THE UNIVERSITY OF CALGARY

Shop Floor Scheduling and Control with the Object-Oriented Analysis and Design Approach

by

William O

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN

PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF MECHANICAL

AND MANUFACTURING ENGINEERING

CALGARY, ALBERTA

JUNE, 2000

0-612-65007-3

ABSTRACT

Researchers have proposed various control architectures for shop floor manufacturing systems, and each has its characteristics, advantages and disadvantages. Therefore when designing a control system, it is important to obtain a natural decomposition of the control algorithms so that the software control model can be decoupled from any preconceived control structures. This provides researchers with the flexibility to implement certain control algorithms in various control forms, and an objective comparison of these alternative control methodologies can then be made.

In this research, we will first discuss why the object-oriented analysis and design approach can be used to help achieve the above-mentioned objective. Some problems in existing research in distributed control systems will be investigated and discussed. As well, the COM/DCOM technology will be used to build some platform independent software control modules that can be easily distributed to and implemented by other researchers.

ACKNOWLEDGEMENTS

Science is nothing but
trained and organized common sense.

Thomas H. Huxley

First of all, I would like to express my special thanks to my supervisor, Dr. Robert W. Brennan for the professional guidance and inspirations that he provided me throughout this work, and for all his efforts in helping me correct the grammatical errors of this thesis.

As well, I would also like to thank the Faculty of Graduate Studies, the Department of Mechanical Engineering, and the Division of Manufacturing Engineering for their generous financial support. And I am grateful to the support staff of the Department of Mechanical Engineering for all their helps.

Finally, I would like to express my thanks to Dr. Robert C. Kremer and Dr. Douglas H. Norrie for being on my examining committee, and to Dr. Norrie for his inspiring teaching on the Object-Oriented technology. I also greatly appreciated all the support from everyone in my family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Researchers have proposed various control architectures for shop floor manufacturing systems, ranging from traditional centralized control to distributed control based on the emerging distributed artificially intelligence (DAI). Recent researchers in intelligent manufacturing systems have proposed to apply the agent technology, which is based on the object-oriented control paradigm (Baker 1997), to shop floor control in order to achieve objectives such as:

1) To improve the control system's adaptability and fault-tolerance against disturbances such as machine failures or rush orders etc.

2) To reduce the control software's complexities so as to simplify the software development, modification and maintenance.

Distributed scheduling algorithms have been used in many researches for distributed control, but there is a general confusion regarding 'decompose and distribute the scheduling responsibilities' and 'eliminate the scheduling function'. In control systems that still generate a pre-production schedule, it is important to loosen the coupling between the control function and the scheduling function. This is an issue that has not been discussed in most research that implements distributed contrel algorithms in control systems. As a result, some of these distributed control systems might still have difficulties with enhancing adaptability against disturbances in a stochastic manufacturing environment. The market-based auction-bidding scheduling approach has been commonly used by many researchers for distributed scheduling, but this approach is mainly for exploring routing flexibility. To achieve certain global performance objectives, job-sequencing mechanisms should also be considered and incorporated into the control algorithms.

As mentioned above, researchers have proposed various control architectures, and each has its characteristics, advantages and disadvantages (Dilts et al. 1991). Research on distributed control has primarily focused on control architecture issues (e.g., hierarchical vs. heterarchical), but has not addressed the relationship between control algorithm and control architecture. As a result, it is important to obtain a natural decomposition of the control algorithms so that the software control model can be decoupled from any preconceived control structures. This provides researchers with the flexibility to implement certain control algorithms in various control forms, and an objective comparison of these alternative control methodologies can then be made.

In this chapter, we will first describe the main control activities in shop floor control systems. Then a brief discussion of current research in manufacturing control architectures will be given. In the last section, an overview of the structure of this thesis will be presented.

1.1 Control Activities in the Shop Floor Control Systems

A shop floor control system mainly embodies decision-making responsibilities such as part scheduling, part routing and resource allocations (Dilts et al. 1991). Bauer et al. have defined the three main elements for shop floor control as (Bauer et al. 1994):

1) Scheduling – To develop a plan based on timely knowledge and data which will ensure all the production requirements are fulfilled.

2) Dispatching – To implement that plan taking into account the current status of the production system.

3) Monitoring – To monitor the status of vital components in the system during the dispatching activity.

In shop floor control systems, scheduling, dispatching, monitoring, and machine execution (i.e., loading code, and initializing, running, and stopping processes) are functional activities, while control encompasses processes and procedures that ensure that the functional activities are carried out in the desired manner (i.e., to ensure that the functional activities take place in appropriate relationship to each other). Depending on the control architecture of the system, functional activities such as scheduling, dispatching and monitoring can be done by one entity in a system or can be distributed over many structures. Similarly, control monitoring can be done by one entity in a system or can be distributed over many structures. Figure 1.1 shows the main functional activities for the shop floor control.

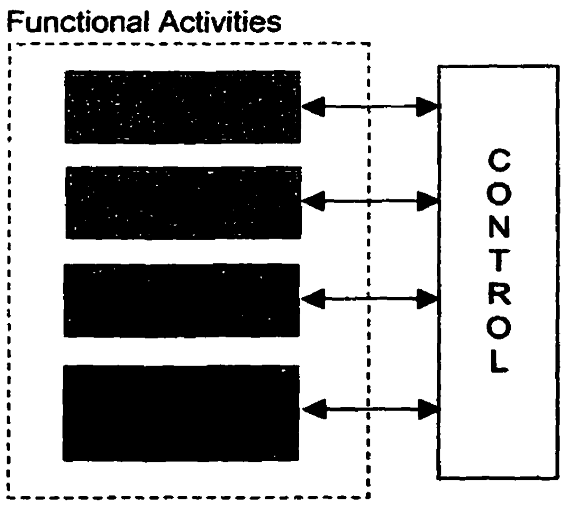


Figure 1.1: The main functional activities for shop floor control.

As mentioned above, shop floor control is concerned with processes and procedures that ensure that the functional activities take place in appropriate relationship to each other. As a result, messages will pass between control and functional entities as is illustrated in Figure 1.2 below.

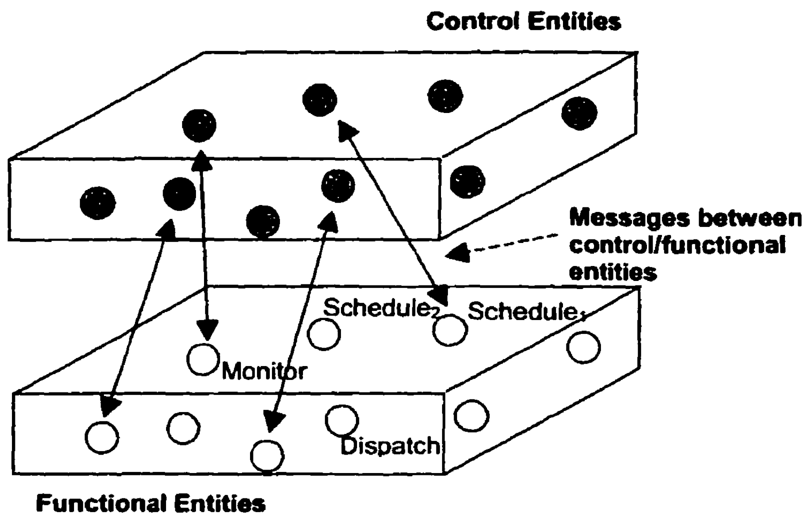


Figure 1.2: Messages passing between control entities and functional entities.

As well, depending on the control architecture that is implemented, the control and functional entities in a shop floor control system may be organized in various configurations. Figure 1.3 below shows one of the possible configurations. The shop floor control architectures will be further discussed in §1.2 below .

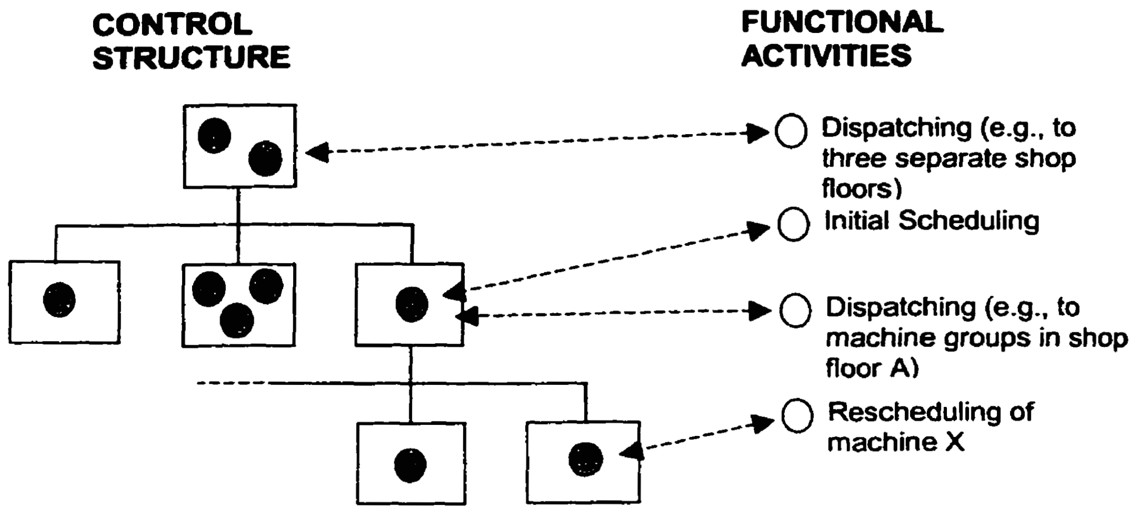

CONTROL STRUCTURE
FUNCTIONAL ACTIVITIES
Dispatching (e.g., to three separate shop floors)
Initial Scheduling
Dispatching (e.g., to machine groups in shop floor A)
Rescheduling of machine X

Figure 1.3: One of the possible configurations for the shop floor control system.

1.2 Control Architectures

While reviewing the evolution of control architectures for the automated manufacturing systems, Dilts et al. (1991) have identified the four basic forms of control architecture, namely, centralized, proper hierarchical, modified hierarchical, and heterarchical as shown in Figure 1.4. In the figure, the boxes represent control components, circles represent manufacturing entities.

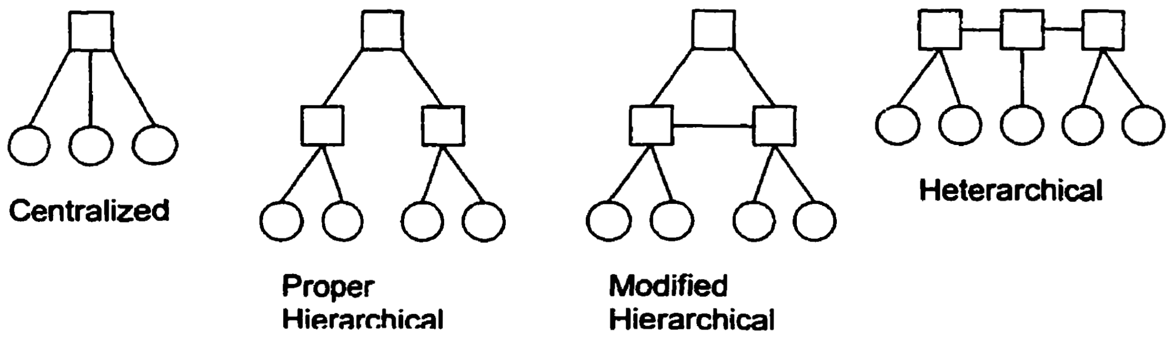


Figure 1.4: Four basic forms of control architecture (Dilts et al. 1991).

The centralized control architecture has a single control unit responsible for all planning, control, and information processing functions. Under this control structure, overall system status information can be retrieved from a single source. Though the central control unit's accessibility to complete global information makes optimization a more readily achievable prospect, the centralized control structure has drawbacks such as slow and inconsistent speed of response (when the system gets larger), difficult to modify control software, and the system's survival relies totally on the reliability of a single control unit (Dilts 1991).

The proper hierarchical control architecture is based on the concept of levels of control, wherein several control components are arranged in a tree structure and strict master/slave relationships are established between decision levels. Commands flow top-down along the hierarchy tree while feedback information flow bottom-up and data are aggregated at each level. The static and deterministic nature of the hierarchical control architecture allows it to work well under environments of certainty and stability. But the rigidity and highly-coupled decision levels of the control structure gives it the disadvantages of having difficulties of making future unforeseen modifications and dealing with dynamic adaptive control (Dilts 1991), and low response time and robustness against disturbances in manufacturing system (Parunak 1993, Valckenaers et al. 1997a, Bongaerts et al. 1998).

The modified hierarchical control architecture was introduced in an attempt to overcome some of the shortcomings of the proper hierarchical structure. In modified hierarchical form, the subordinates are granted some degree of local autonomy. So by cooperating with some other peer subordinates, the subordinates might be able to release some rudimentary responsibilities from the supervisory control level, and thus enable the supervisor to respond more readily to subordinate requests. Also, by having the subordinates act "as an autonomous subsystem within the hierarchy, in the sense that they do not require continuous supervision (they) are characterized by some degree of robustness with respect to random disturbances" (Dilts 1991). The drawbacks of the modified hierarchical form are that it still bears most of the disadvantages of the proper hierarchical form and has the connectivity problems with peer-to-peer communication (Dilts 1991).

The heterarchical control architecture is a highly distributed form of control. Every entity in the system has full local autonomy and there is no centralized or explicit direct control (supervisor/subordinate) existing in the system. Control decisions are reached via peer-to-peer co-operation and mutual agreement among the participating entities, and information is exchanged freely among them. The claimed advantages of heterarchical control architecture include: 1) enhancement of the control system's robustness, flexibility and expandability (Saad et al. 1997), 2) reduced software complexity and development cost, improved fault tolerance, and higher maintainability and modifiability due to enhanced modularity and self-configurabiltiy (Duffie & Prabhu 1994), 3) high robustness against disturbances in manufacturing (Bongaerts et al. 1998). However, due to the fact that in a heterarchical control system, entities use purely localized information and all forms of hierarchy are eliminated, heterarchical control turned out to have problems with global optimization and predictability of system behaviors (Valckenaers et al. 1997a, Bongaerts et al. 1998).

1.3 Thesis Overview

The objectives of this research are mainly concerned with identifying and investigating some of the problems in the existing research in the shop floor control methodologies. These problems include:

- the decomposition approaches used for designing the control systems,
- confusions regarding the concepts of the 'distributed scheduling' and 'real-time distributed control',
- the impact of the routing flexibility and the job sequencing control mechanisms on the performance of a multi-agent heterarchical control system, and

- how to enhance the collaboration between researchers in developing alternative control methodologies.

In Chapter 2, we will review some of the above mentioned problems in the existing research in manufacturing system control. In Chapter 3, more detailed description of the research objectives will be given. Also, the research approach and the anticipated contributions of this study will be discussed. In Chapter 4, we will use the object-oriented analysis and design approach to develop an experimental testbed that will be used for conducting various experiments in the following chapters. The reason for using the object-oriented methodology to design the control system used in the experimental testbed will be explained in Chapter 4 as well. Then in Chapter 5, experiments will be conducted to investigate and identify the role of the 'control' algorithm in the control systems that use the distributed scheduling approach to perform pre-production schedules. These experiments are used for clarifying the confusion in the existing research regarding the concepts of 'distributed scheduling' and 'real-time distributed control'. In Chapter 6, experiments will be conducted to identify the impact of the job routing and job sequencing control mechanisms on the performance of a multi-agent heterarchical control system in different manufacturing environments.

In Chapter 7, an attempt will be made to explore the opportunity of enhancing the collaboration between researchers by building some platform independent (COM/DCOM) software control modules that can be easily distributed to and implemented by other researchers. Finally, in Chapter 8, a conclusion regarding the work in this study in the context of the general research objectives will be given. As well, the contributions of this study and suggestions for future research work are discussed.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we will first discuss some of the problems presented in traditional centralized scheduling and control systems. While some researchers have showed that decoupling the control function from the scheduling function can enhance the shop floor control system's flexibility against disturbances, current research in distributed scheduling and control systems is mainly focused on discussing the distributed scheduling algorithms, but fails to address the control algorithms. As resulted, some of these distributed control systems might still have difficulties with enhancing adaptability against disturbances in a stochastic manufacturing environment. We will further discuss these issues in sections 2.2 and 2.3 below. In §2.4, we will discuss some of the problems regarding the job sequencing and the job routing control mechanisms in current research in the multi-agent heterarchical control systems. Then in §2.5, we will discuss some of the approaches that researchers have used to design shop floor control systems, and see why using the object-oriented analysis and design approach to structure the control algorithms can help decouple the logical control model from any preconceived control architectures.

2.1 Centralized Shop Floor Control System

Conventional shop floor control systems are centralized and usually implemented with the control structures as shown in Figure 2.1 below. In Figure 2.1 (a) a, a central computer plays both the roles of a scheduler and a controller, while in Figure 2.1 (b), the scheduler and controller responsibilities are assigned to different processors. Both of these control approaches are generally regarded as centralized shop floor control (Duffie et al. 1994). In these approaches, the scheduler is responsible for doing all the decision-makings regarding parts routing, operations sequencing, resources allocating etc., and the controller is responsible for executing the plans generated by the scheduler and feeding back the status of the shop floor to the scheduler when disturbances happen.

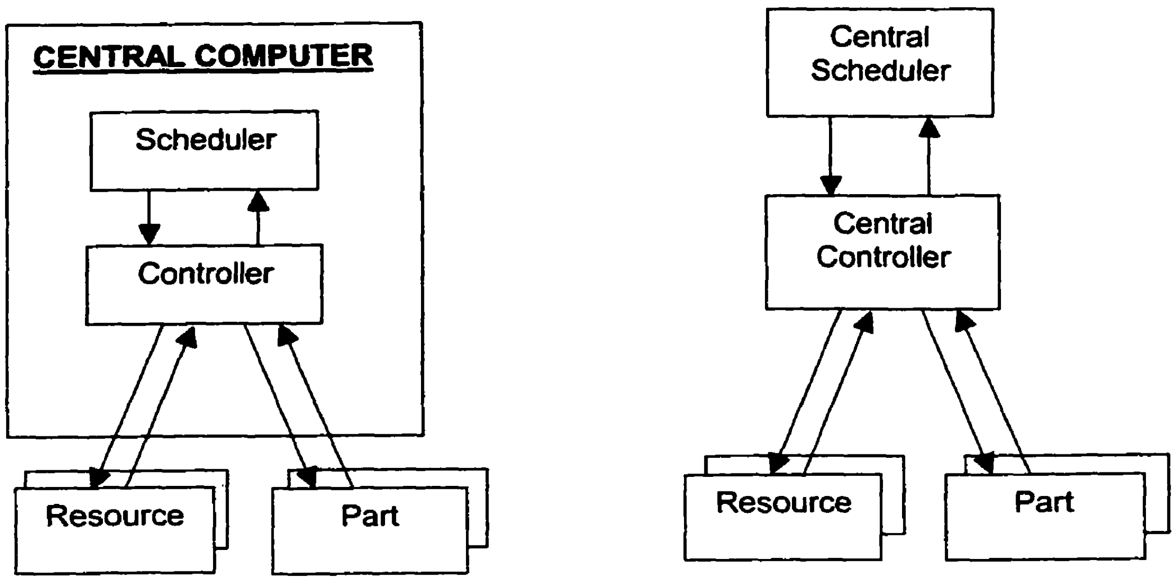


Figure 2.1 (a) Figure 2.1 (b)

Figure 2.1: The centralized shop floor control architecture.

The tightly coupled master/slave relationship between the scheduler and controller results in problems such as:

1) Low fault-tolerance. Failure of the scheduler, controller or central computer can cause the control system to halt.

2) Complex control software and difficulties with dealing with dynamic adaptive control. The centralized control approach "leads to large, complex software systems that are difficult to create, install, and modify, and yields schedules that are vulnerable to changing circumstances on the shop floor" (Parunak 1993). Also, "It is not unusual for schedule generation to take several hours and for shop floor conditions to change significantly before the schedule is completely generated" (Duffie et al. 1994), and this affects the control system's response speed/adaptability against disturbances.

2.2 Decoupling the Control Function from the Scheduling Function

Different agent-based shop floor scheduling and control approaches have been proposed by a number of researchers to tackle the problems associated with the centralized control system. These alternative approaches differ widely on what is represented as an agent. In some approaches, an agent is assigned to each control node in the hierarchical (centralized) control system (Parunak et al. 1998a, Bongaerts 1998), while in the others, agents are associated with some physical manufacturing resources or parts (Duffie et al. 1994, Sousa et al. 1997, Van Brussel et al. 1998). Shen et al. (2000) have provided some classifications regarding agents in the manufacturing systems.

In control systems that contain a central scheduler, researchers have attempted to enhance the control system's adaptability against disturbances by loosening the coupling between the scheduler and the lower level controllers (The term coupling refers to the strength of the associations between objects or agents that is a result of connections between them (Booch 1994, Shen 2000). This is accomplished by giving the controllers which are responsible for executing the schedule a certain degree of intelligence and autonomy so that they could handle the disturbances in real time. For instance, in (Valckenaers et al. 1997b) (referring to Figure 2.2), "Shop floor control is performed by both an on-line control system that reacts to disturbances immediately and a reactive scheduler that does not react as fast, but uses this larger time span to adapt the existing schedule to optimize global performance".

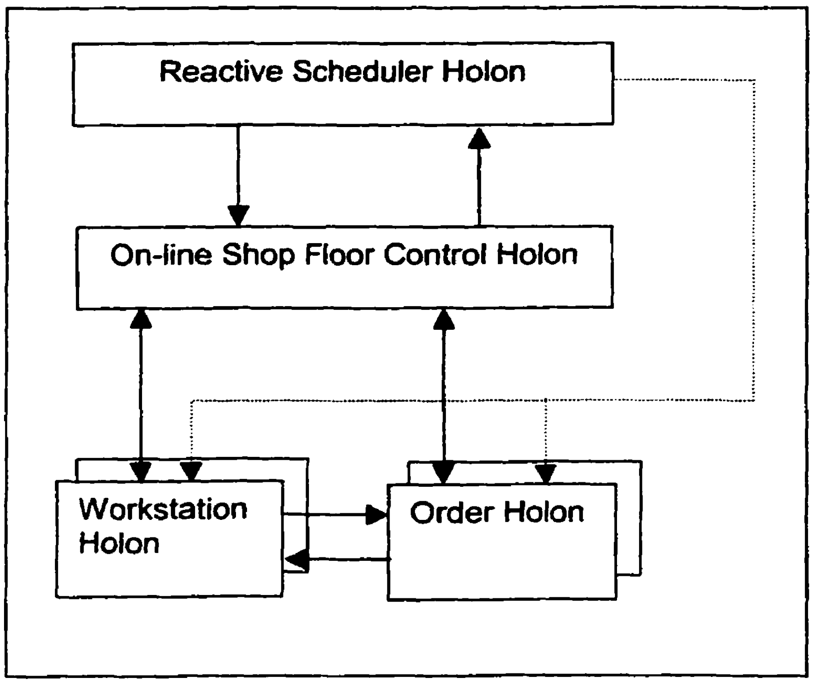


Figure 2.2: A holonic architecture for scheduling and on-line shop floor control (Valckenaers et al. 1997b).

In Figure 2.2, each of the control entities in the system is represented by a holon, which is defined as:

"An autonomous and cooperative building block of a manufacturing system for transforming, transporting, storing, and/or validating information and physical objects. The holon consists of an information processing part and often a physical processing part. A holon can be part of another holon" (Valckenaers et al. 1997a).

In this approach, scheduling is performed by the central (reactive) scheduler. The local autonomy of the lower level control holons is mainly used for dispatching/control purpose. That is, when disturbances happen and the (old) production schedule is no longer feasible or the scheduler is not available or is busy in generating a new schedule, the workstation, order or on-line control holons can make some local decisions to react to disturbances in real time. This can enhance the control system's fault-tolerance and adaptability against disturbances.

2.3 Distributed Scheduling Algorithms and Real-time Distributed Scheduling

Distributed scheduling algorithms have been used by many researchers (especially multi-agent control systems) to generate a production schedule, but few have addressed how the control agents should react in response to disturbances in real time. This might be because of the confusion regarding the use of a distributed scheduling algorithm to generate pre-production schedules and its use to implement real-time dynamic scheduling. In the latter case, the scheduler role (or scheduling function) in the control system is totally eliminated, and production decisions (resources allocation, part routings, etc) are made in real time. In such an approach,

> "Dynamic scheduling implies moving scheduling decisions from production pre-planning to the FMS control system. Shifting these decisions into real-time greatly increase over-all system performance by introducing the capability to handle disturbances such as system component failures without operator interaction" (Duffie et al. 1986).

But if the distributed scheduling algorithm is used to generate a pre-production schedule, even in a heterarchical control system, there is still a tight coupling between the local controllers and the generated schedule (referring to Figure 2.3).

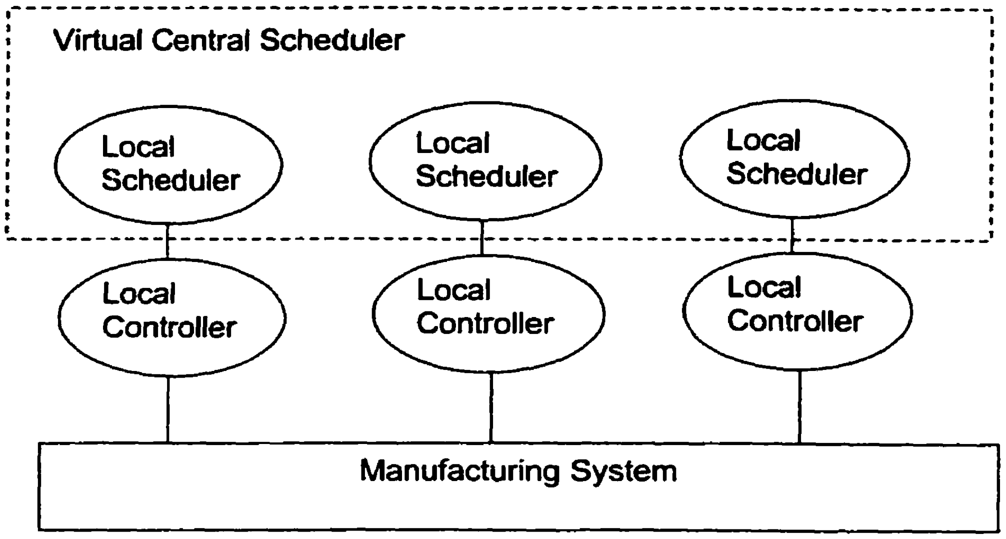


Figure 2.3: Heterarchical manufacturing system scheduling and control, modified from (Duffie et al. 1986).

This is because a production schedule serves as a virtual contract that binds all the participating control entities together to accomplish a certain tasks. For instance, in control systems that use auction bidding scheduling algorithms (Duffie et al. 1994, Sousa et al. 1997) to generate a schedule, once a schedule is generated, each part and resource agent is committed to part of the common schedule (future plans). That is, a resource agent is committed to provide its capacities to certain parts at certain time intervals, and a part agent is committed to have its operations processed by certain resources at certain times. When unexpected disturbances happen, if the affected control agents do not have the responsive mechanism to react to these disturbances in real time so that the production plans will be less affected, a rescheduling process will usually be invoked (Duffie et al. 1994) for the control entities to generate a new schedule. Even using distributed scheduling algorithms, a new schedule might still take some time to be generated. In the meantime, the affected control agents (or production resources) have to wait for the new schedule to be generated. Also, in the scheduling/rescheduling processes, all the control entities have to take part in the process, and this might increase the coupling and communications between the control entities. Therefore, relying on rescheduling to react to disturbances might contradict some design principles of the heterarchical control system, such as (Duffie et al. 1994):

- Time-critical responses should be contained within entities and should not be dependent on time-critical responses from other entities.

- Scheduling and control functions should be contained within the same entity to avoid time-critical response requirements and decrease communication requirements.

Distributed scheduling can enhance the scheduling performance "through parallel computing and through the elimination of the processing bottleneck caused by global scheduler" (Dilts et al. 1991). Localizing the scheduling and control functions within the same control entity can enhance the control software's modularity, reduce control system complexity, and increase flexibility and fault-tolerance (Duffie et al. 1986). But as mentioned above, in the distributed control system, it is also important that the control entities have the local intelligence/autonomy (or reactive mechanism) to respond to disturbances in real time so as to "avoid time-critical response requirements and decrease communication requirements" (Duffie et al. 1986).

2.4 Jobs Sequencing and Dispatching Routing Decision in Multi-agent Heterarchical Control Systems

While reviewing the implementation of dispatching rules in the multi-agent heterarchy, Baker (1997) has mentioned that:

"It is most common to dispatch the routing decision in these architectures, assuming sequencing can then be done at each resource... In the case of the routing decision, a great deal of agent research has been with having the agents making this decision by collecting bids from potential machines to which the job can be routed".

Most of the work on the distributed scheduling and control only dealt with dispatching the routing decisions, and ignored the job sequencing issues. As a result, most of these scheduling approaches performed scheduling by jobs on a First-Come-First-Serve basis (Veeramani & Wang. 1998), and this approach sometimes will compromise certain global performance objectives.

Some attempts have been made to incorporate the sequencing mechanism into the distributed scheduling algorithm in order to optimize the global system performance. For instance, in (Ramaswamy et al. 1995), an offline central computer is used to solve the static optimal characterizations of the scheduling problem and forward the generated Lagrange multipliers to the online control agents (machine and part agents). The online control agents will then make use of this (global) information to decide the allocation and sequencing of both processing and material handling tasks. Although the Lagrangian relaxation technique seems very promising for heterarchical agent scheduling, "current work in this area is only a beginning of the work which needs to be performed to pursue these concepts fully" (Baker 1997).

In (Duffie et al. 1994), an attempt to incorporate the job sequencing mechanism into the distributed scheduling algorithm is implemented by the cooperative scheduling mechanism. In their control system, part agents generate local plans, which are then evaluated by a central agent. The evaluated global performance measure will then pass back to all the control agents. The part agents will then alter their local plans accordingly and the afore-mentioned procedures are repeated until "a set of local schedules is found that collectively achieves the global goals" (Duffie et al. 1994). Jobs sequencing is accomplished in the manner that while a number of parts are in contention to reserve a resource, the part agents with the looser due-date will change their local plans to delay the reservation so that the parts with the tighter due-date can use the resource first. Because lack of global information knowledge, the part agents do not know which part is in most need of a particular resource. As a result, the part agents have to alter their local plans on a trial-and-error basis to see how their changed plans will affect the global performance. This trial-and-error approach also causes 5 out of 12 parts to be tardy, even though in their experimental model, there was enough lead times for the production of all the parts.

"The most common scheduling approach to job shops is to use priority dispatching rules" (Sipper & Bulfin 1997). "For many practical applications, shop floor control is dominated by heuristic dispatching, in which a simple decision rule determines the next job to be processed at a given workstation" (Parunak 1994). A considerable number of dispatching rules have been developed by researchers to help the control system in achieving certain global performance objectives, and some of these dispatching rules are simple and widely adopted by the manufacturing industry.

As described above, the 'part-driven' approach tends to have difficulties in fulfilling the job sequencing responsibilities efficiently because of lack of global information knowledge, and other more efficient approaches have yet to be developed/studied. In (Duffie et al. 1994), it was mentioned that "future work should also compare them with traditional dispatching rules and scheduling heuristics".

In a distributed control system, while the part agents can use the bidding mechanism to explore the routing or process sequencing opportunities, it is possible for the resource agents to use the dispatching rules to sequence the jobs to help improve certain global performance objectives. Having the resource agents responsible for job sequencing will not compromise the design principles of the heterarchical system, as the routing and sequencing decisions are made by job and resource agents, respectively, locally. Although some researchers (Saad et al. 1997) have incorporated dispatching rules in the market-based scheduling approach and have shown that this approach could improve the system performance with respect to certain global performance measures, few have discussed how the opportunistic behavior of the part agents and the adoption of the dispatching rules will affect the commitments between the control agents, the overall system performance, and the communication requirements. For instance, when the resource agents submit bids to the part agents, the part agents will make the routing/process sequencing decision based on the bids. But if the resource agents use the dispatching rules to sequence the jobs, the resource agents might violate some of the commitments they made to some of the part agents. When that happens, should the affected part agents be informed so that they can explore some other routing opportunities? And what impact will that make on the system performance and communication requirements?

2.5 Shop Floor Control Architectures

Different control architectures have been proposed for shop floor scheduling and control, ranging from traditional centralized control to the distributed multi-agent systems. Although quantitative results are becoming available for these alternative methodologies, it is difficult to evaluate these alternative control architectures from the existing literature alone since different approaches tend to use different control algorithms. As a result, one can not be certain if it is the control architecture or the control algorithm that is being compared. In designing a control system, it is important to obtain a natural decomposition of the control algorithms so that the resultant software control model can be decoupled from any preconceived control architectures. This provides the researchers with the flexibility to implement the software control modules in various control forms, and thus an objective comparison of alternative control methodologies can then be made.

Figure 2.4 shows some of the control architectures that have been proposed for shop floor manufacturing systems. Figure 2.4(a) represents the centralized shop floor control architecture, wherein all the manufacturing scheduling and control are characterized by a centralized computer and database (Parunak 1993). Figure 2.4(b) represents the hierarchical control structure, wherein structural (functional) analysis approach is used to functionally decompose the control activities into different control modules. While reviewing the hierarchical control architecture, it was described in (Dilts et al. 1991) that:

> "The refinement process of breaking down aggregated decisions and the concept that 'sensory information at the higher levels is more abstract and requires the

integration of data over longer time intervals' (Simpson et al. 1982) implies that aggregated database will be found at each level"

Therefore, from centralized to hierarchical control, the design approach is shifted from developing a monolith control program to modularizing different control functions and their corresponding database in different control modules. "Software development can be easily managed and modifiability can be easily achieved due to the modularity of the hierarchical structure" (Dilts et al. 1991)

Figure 2.4 (d) represents the general multi-agent control architecture. Some mutli-agent control systems use only the resource and part agents to perform the scheduling and control (Duffie et al. 1994), while other control systems may involve some staff roles (central scheduler or mediator, etc.) to perform the scheduling and control (Bongaerts et al. 1998, Shen et al. 1999). These staff agents are represents as agents X and Y in Figure 2.4 (d). In multi-agent systems, the scheduling and control responsibilities are distributed among some loosely-coupled cooperative control entities. The multi-agent control architecture can enhance the modularity of the control systems. "The reduced coupling between (software) modules reduces complexity and simplifies development and maintenance (modifications and extensions)" (Dilts et al. 1991).

Most research in shop floor control developed the control systems from a "top-down" approach, wherein control architectures were first determined, then the control algorithms were structured to fit in the preconceived control architectures. But this approach limits the flexibility and increases the difficulty of changing a control system's control structure. Sometimes the physical environment (communication network's availability, computational limitations of certain control entities) of a control system might favor certain control architecture, but this environment might change overtime. Therefore, it is important that there is a natural decomposition of the control algorithms so that the logical control model can be decoupled from any preconceived control structure, and this provides the flexibility of implementing the logical control modules in various control forms.

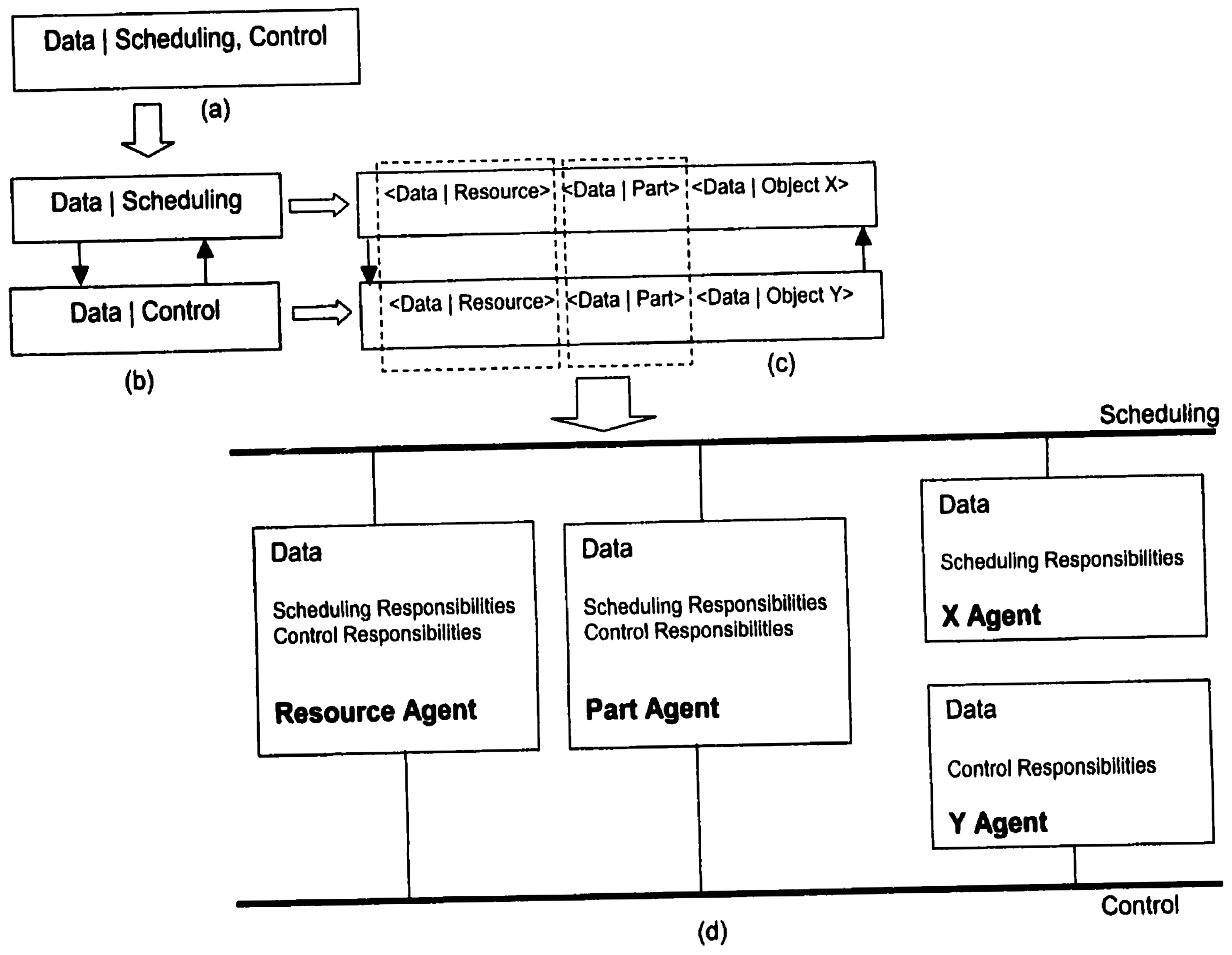


Figure 2.4: The decomposition approaches for control systems with centralized, hierarchical and heterarchical control form, respectively.

2.5.1 Structuring Control Algorithms with the Object-oriented Analysis and Design Approach

To obtain the natural decomposition of the control algorithms so that the logical control model can be decoupled from any preconceived control architectures, object-oriented analysis and design methodology appears to be the appropriate approach for decomposing the control algorithms. Figure 2.4 (c) represents the software control model wherein the object-oriented decomposition approach has been applied to structure the scheduling and control algorithms.

Referring to Figure 2.4 (c), by applying the object-oriented decomposition approach to structure the scheduling algorithm, roles that are involved in the process can be identified, and their corresponding responsibilities and states can be encapsulated. It would not be surprising that production entities such as machines, parts etc. will be identified in most scheduling algorithms since scheduling is mainly concerned with allocating operations to resources. It should be noted that objects identified from the scheduling algorithms are not limited to objects that have a physical correspondence in the real manufacturing system. Other objects may also be identified for certain design purpose or to help accomplish the scheduling tasks.

> "In analysis, we seek to model the world by discovering the classes and objects that form the vocabulary of the problem domain, and in design, we invent the abstractions and mechanisms that provide the behavior that this model requires" (Booch 1994).

The control algorithms (for the control function) are usually structured around some control roles that have a physical correspondence in the production system. These control roles are responsible for executing the production plan and monitoring the production activities of their physical correspondence. "One cannot make all the logical design decisions before making all the physical ones, or vice versa; rather, these design decisions happen iteratively" (Booch 1994).

As shown in Figure 2.4 (c), some roles (resource, part) in different control functions are related to a particular entity. These roles (and their corresponding responsibilities and data) can then be modularized in a single control component so that this component will be responsible for the control behaviors of a particular entity, and thus increase the cohesion of the control component ("Cohesion measures the degree of connectivity among the elements of a single module" (Booch 1994)). With this approach, roles in different control functions can be modularized into some low-coupling, high-cohesion control components. These control components can then be implemented in a distributed (multi-agent) control structure as shown in Figure 2.4 (d) to fulfill the required control functions of the system.

As illustrated above, by applying the object-oriented methodology to decompose the control functions in a control system (Figure 2.4 (c)), the resultant control modules can be arranged to fit into various control architectures. For instance, one can retain the functional structure in Figure 2.4 (c) to implement the hierarchical control (or even implement all the control functions in a single computer to form the centralized control). Or one can further modularized the control roles in different control functions to form some low-coupling, high-cohesion control components, and implement these components in a distributed control structure. Therefore, applying the object-oriented methodology to structure the control functions can help achieving the objective of decoupling the control algorithms from the control architectures. Also, by using the object-oriented analysis and design approach to structure the control functions, notation (for example, UML-Unified Modeling Language (Larman 1997)) for modeling systems using object-oriented concepts can be used to capture (document) the static and interaction models of the control processes. These artifacts will be helpful when modifications have to be made to the logical control software in the future (or to reconfigure certain control roles into different control modules).

CHAPTER 3

MOTIVATION FOR THIS RESEARCH

In this chapter, we will first describe the objectives of this research, and the methodologies that will be used to investigate the problems in this study. Then the anticipated contributions of this research will be given.

3.1 Research Objectives

The objectives of this research are mainly concerned with identifying and investigating some of the problems in the existing research in the shop floor control methodologies. These problems include:

1) Control system decomposition approaches – Various control methodologies have been proposed for shop floor control, but most research tends to decompose the control system from a 'top-down' approach. In this research, we will analyze some of these approaches and try to use a natural decomposition approach to structure the control algorithms so that the resultant software control model can be decoupled from any preconceived control architectures.

2) Confusions regarding 'Scheduling' and 'Control' functions in the distributed shop floor control systems – Distributed scheduling has been used in many researches, some distribute the scheduling function to generate pre-production schedules, while others distribute the control function to implement real-time distributed production

control (dispatching and monitoring). In this research, we will clarify this confusion and identify some problems resulted from this confusion.

3) Routing flexibility and job sequencing – Most research using the market based scheduling approach has not considered the job sequencing issue. As a result, resource allocations are usually done by jobs on a first-come-first-serve basis, which can compromise certain global performance measures. In this research, we will investigate and identify the role of the job routing and job sequencing control mechanisms in a multi-agent distributed control system, and discuss how these control mechanisms will affect the performance of the control system in various manufacturing environments.

4) Software reuse and distribution – As some researchers have proposed that different researchers should compare their control algorithms on a common testbed, it will be helpful if researchers can built their control modules as some platform-independent software components that can be easily distributed across the network and integrated into some other control systems. In this research, we will explore this opportunity by trying to use the COM/DCOM technology to build some control modules and distributed them across the network to implement the simulated distributed shop floor control system.

3.2 Research Approach

As mentioned above, the research conducted in this thesis is mainly concerned with the problems related to the shop floor manufacturing systems; particularly in distributed shop floor scheduling and control. Since a lot of work has been done for shop floor scheduling and control, some researchers have seen the need to develop a benchmark framework for manufacturing control so that researchers can validate and assess the performance of their algorithms on a common testbed. A recent formed research group, BENCH-MAS

(Cavalieri et al. 1999), has developed two benchmark proposals for a generic machining system and a flexible assembly system, respectively. The generic machining system proposed by (Cavalieri et al. 1999) is intended to serve as a testbed for comparing multi-agent control systems, and will be adopted in this research as a common platform for evaluating a variety of test scenarios.

The production system consists of four types of machines, and two machines per type are present. Although it is proposed that the transport system is modeled as a set of serial transporters (AGVs), these AGVs are assumed always available and transport times are set equal to zero. Therefore in this research, the transportation entities and transport times are not modeled in order to simplify the system. Two types of job shop problems are proposed. For the first problem, products have a fixed process plan constituted by four non-preemptive operations (one for each machine type), and the third machine to be visited is the bottleneck resource (long-lasting operation). The second problem is similar to the first one, except that in this problem, routing flexibility is introduced into the system. That is, products have a flexible processing order of their operations (a bottleneck resource is still present, but not necessary the third one to be visited). More detailed descriptions of the production system and performance measures are covered in chapter 5.

As mentioned in §3.1, current research in distributed scheduling and control systems mainly focuses on discussing the distributed scheduling issues, and ignored the control issues. In this research, we will use the first job shop problem to investigate how different control algorithms in the distributed control system will affect the control system's performance in a stochastic manufacturing environment. A heuristic scheduling algorithm will be used to generate the production schedules. Although the heuristic methods may not guarantee to generate an optimal schedule, these methods guarantee a solution in a reasonable amount of time, and by far the most commonly used techniques in industry (Baker 1997).

In the previous research on distributed (multi-agent) control systems, control responsibilities are usually assigned to certain control agents (such as the part or machine agents) that have a physical correspondence in the real manufacturing system. The scheduling algorithm is then structured around these control agents (Duffie et al. 1986, 1994). Instead of using this 'top-down' approach to design the control agents for the control system, in this research, we will first use the object-oriented analysis and design approach to structure the scheduling algorithm so that roles (objects) that are involved in the scheduling processes can be identified. Such an approach can allow us to identify a broader set of possible agent candidates, and to design the logical scheduling model that can be decoupled from any preconceived control structure. As would be expected, some of the objects identified may have a correspondence in the physical production system, since scheduling is about allocating jobs to resources. Since we are trying to investigate control problems in a distributed control system, control responsibilities will then be added to some of the objects identified in the scheduling decomposition process. With the scheduling and control responsibilities, these objects will then behave as the control agents for certain production entities. It should us noted that control agents are not limited to the 'physical' control agents (Shen et al. 2000). Other agents may also be created to help accomplishing the scheduling and control functions of the control system. An experimental model will be constructed around the identified control agents. Different control responsibilities will be introduced to the control entities, and the control system performance will be tested under different stochastic disturbance scenarios.

As mentioned in §2.4, most of the work on multi-agent heterarchical control systems only dealt with dispatching the routing decisions, but ignored the job sequencing issues. Since routing flexibility is introduced to the products in the second proposed job shop problem, we will use this problem to investigate the role of the job sequencing and job routing control mechanisms in a distributed (multi-agent) control system. Experiments will be conducted to test and evaluate the impact of the job sequencing and job routing control mechanisms on the control system's performance under various manufacturing environments.

As the benchmark framework (Cavalieri et al. 1999) is intended for different researchers to compare their control methodologies on a common testbed, it would be helpful if the control modules can be built into some platform independent software components that can be easily distributed across the network and/or be integrated into other researchers' (software) control models for validation or testing. In the last section of this research, we will try to explore this opportunity by using the COM/DCOM technology (Bates 1999, Sing et al. 1998) to implement the control entities in a multi-agent control system. These component objects will then be distributed across the network for implementing the distributed control.

3.3 Anticipated Contributions of this Research

Through the work in this research, it is expected that the following objectives can be achieved:

1) To provide some insights regarding the decomposition approaches for various control methodologies, and the importance of decoupling the control algorithms from the control architectures.

2) Clarify the confusing concepts regarding 'distributed scheduling' and 'real-time distributed control'. Identify the role of the 'control' algorithm in the control systems that use the distributed scheduling approach to perform pre-production scheduling.

3) To provide some insights regarding the role and importance of the job sequencing and dispatching routing decision control mechanisms in the multi-agent distributed control system (in various manufacturing environments).

4) To explore the opportunity of enhancing the collaboration between researchers by building some platform independent software that can be easily distributed to and implemented by other researchers.

CHAPTER 4

EXPERIMENTAL MODEL DEVELOPMENT

As mentioned in the last chapter, the objectives of this research are mainly concerned with investigating the control problems related to the distributed control systems. In the current research in distributed scheduling and control systems (Duffie et al. 1994, Sousa et al. 1997), production entities such as machines, workstations, and parts etc. are usually associated with a corresponding control agent, which will control the production activities of these entities. Scheduling algorithms are usually decomposed/structured around these control agents. In this research, we will use the object-oriented analysis and design approach to decompose the scheduling algorithm to identify the roles and responsibilities that are involved in the scheduling processes. This will give us the opportunities to explore other possible agent candidates (other than the pre-determined 'physical' (Shen et al. 2000) control agents).

> "In selecting our agents, we want to begin with the broadest possible set of candidates. While some entities may prove unnecessary, it's easier to cast the net broadly and leave some as stubs than to build an architecture into which omitted entities cannot easily be added later" (Parunak et al. 1998a).

As would be expected, some of the roles (objects) identified in the scheduling algorithm may have a physical correspondence in the real manufacturing system, since scheduling is about allocating operations to resources. The scheduling and control responsibilities of these roles can then be modularized into some individual control components. This, together with other agents that might be identified from the scheduling algorithm, allows us to then implement a multi-agent (distributed) scheduling and control structure.

Another advantage of using the object-oriented approach to decompose the scheduling algorithm is that the software solution can be decoupled from any preconceived control structure. For instance, in a distributed control system wherein the control responsibilities are distributed among certain control agents, we can still implement the (scheduling) software solution in an individual control module to implement a centralized scheduling, distributed control structure. Or as described above, we can incorporate the software solution into the distributed control system to implement a distributed scheduling, distributed control system.

In this chapter, we will first describe the production model and the scheduling algorithm that will be used in our experimental model. The object-oriented analysis and design approach will then be used to decompose the scheduling algorithm so that the roles that are involved in the scheduling processes can be identified. Some of these roles may be the possible agent candidates, and the corresponding control responsibilities will then be added to these roles so that a distributed scheduling and control system can be built. Based on this distributed control model, we will then build an experimental model for simulating the shop floor manufacturing and control activities. This experimental model will be used in the later chapters for testing different control methodologies.

4.1 Characteristics of the Manufacturing System

The followings are some of the characteristics and assumptions that were made about the manufacturing system.

1) The system contains a number of workstations/stations. Each station has a queuing buffer (queue) and a number of resources/machines.

2) Each station can offer a single type of operation. That is, machines of the same station have same function.

3) Resource (stations, machines) information, process plans for the jobs and order details are stored in resource, production and order database, respectively. The scheduling application has to access the corresponding database to retrieve the relevant information.

4) Set-up time for each operation and transportation times for moving jobs between stations are ignored.

5) No preemption. Once a machine starts processing a job, it will continue until it completes the operation (that is, no machine breakdown will occur when the machine is in operation).

6) When disturbances (like machine breakdowns or new order arrivals) happen, the scheduling application might be invoked to do the rescheduling.

4.2 Scheduling Algorithm

In this research, the Giffler-Thompson (French 1990) algorithm will be used as the scheduling algorithm. This algorithm is a heuristic scheduling algorithm for generating non-delay schedules for the general job shop problem, n/m/G/B. Non-delay schedules are schedules "where no machine is kept idle when it could start processing some operations" (French 1990). For the general job shop scheduling problem, n/m/G/B, n = number of jobs, m = number of machines, G = the general job-shop case, and B = the performance measure. Although the heuristic methods may not guarantee to generate an optimal schedule, these methods guarantee a solution in a reasonable amount of time, and by far the most commonly used techniques in industry (Baker 1997). Also, there are strong empirical reasons for using the non-delay scheduling approach. For example, in the empirical studies conducted by (Conway et al. 1967), the results showed that in most cases, the non-delay schedules far outperformed the active schedules (active schedules are schedules in which an optimal solution is guaranteed in the set of schedules) in terms of mean flow time. The Giffler-Thompson scheduling algorithm is described as follows (French 1990):

"In the algorithm we shall schedule operations one at a time. We shall say that an operation is schedulable if all those operations which must precede it within its job have been already been scheduled. Since there are nm operations, the algorithm will iterate through nm stages. At stage t, let

P_t – be the partial schedule of the (t-1) scheduled operations;

S_t – be the set of operations schedulable at stage t, i.e. all the operations that must precede those in S_t are in P_t.

σ_k – be the earliest time that operation o_k in S_t could be started;

ϕ_k – be the earliest time that operation o_k in S_t could be finished, that is, $\phi_k = \sigma_k + p_k$, where p_k is the processing time of operation o_k;

...It is an easy matter to modify this (Giffler and Thompson) algorithm so that it produces non-delay schedules."

Algorithm 4.1 below shows the steps for generating a non-delay schedule.

Algorithm 4.1 (French 1990):

Step 1 Let $t = 1$, P_1 being null. S_1 will be the set of all operations with no predecessors, in other words, those that are first in their job.

Step 2 Find $\sigma^* = \min_{o_k \text{ in } S_k} \{\sigma_k\}$ and the machine M* on which σ^* occurs. If there is a choice for M*, choose arbitrarily.

Step 3 Choose an operation o_j in S_t such that

(1) it requires M*, and

(2) $\sigma_j = \sigma^*$.

Step 4 Move to next stage by

(1) adding o_j to P_t so creating P_{t+1};

(2) deleting o_j from S_t and creating $S_{t+1}$by adding to S_t the operation that directly follows o_j in its job (unless o_j completes its job);

(3) increment t by 1;

Step 5 If there are any operations left unscheduled ($t <= nm$), go to Step 2. Otherwise, stop.

4.3 Analysis and Design of the Scheduling Application with the Object-oriented Approach

There are a number of proven approaches for analysis that are relevant to object-oriented systems. These approaches include the classical approaches, behavior analysis, domain analysis, use-case analysis, CRC cards etc. (Booch 1994). In this thesis, since we are trying to identify the objects that are involved in the scheduling processes, we will use an approach that is similar to the behavior analysis and the use-case analysis approaches, as these approaches are used for identifying roles / objects that are involved in certain business processes. The followings are the descriptions for the behavior and use-case analysis approaches.

Behavior Analysis – "Rubin and Goldberg offer an approach to identifying classes and objects from system functions. As they suggest, 'the approach we use emphasizes first understanding what takes place in the system. These are the system behaviors. We next assign these behaviors to parts of the system, and try to understand who initiates and who participates in these behaviors.... Initiators and participants that play significant roles are recognized as objects, and are assigned the behavioral responsibilities for these roles' (Rubin et al. 1992)" (Booch 1994).

Use-case Analysis – Use-case analysis was first formalized by Jacobson (Jacobson et al.1992). This approach is typically used "to enumerate the scenarios that are fundamental to the system's operation. These scenarios collectively describe the system functions of the application.... As the team walks through each scenario, they must identify the objects that participate in the scenario, the responsibilities of each object, and how those objects collaborate with other objects, in terms of the operations each invokes upon the other" (Booch 1994).

In this research, we will first walk through each of the steps (scenarios) in the scheduling algorithm to identify the objects that are involved in the scheduling processes. Since the scheduling algorithm will be used to allocate operations to resources in the simulated manufacturing system, we will interpret the scheduling algorithms in terms of some of the physical production entities that are present in the manufacturing system. It should be noted that objects identified in the scheduling algorithm are not limited to the objects that have a physical correspondence in the manufacturing system. Other objects might also be identified and in together, all these objects will collaborate with each other to accomplish the scheduling processes.

"In analysis, we seek to model the world by discovering the classes and objects that form the vocabulary of the problem domain, and in design, we invent the abstractions and mechanisms that provide the behavior that this model requires" (Booch 1994).

4.3.1 Scheduling Algorithm Walkthrough

In this section, we will first walkthrough a part of a non-delay schedule generation example to see how the scheduling algorithm actually works. In the example, the system has 2 stations; Station 1 & 2. Station 1 has 2 machines, M1 and M3, and station 2 has 1 machine, M2. 6 jobs have to be scheduled and their process plans are listed in Table 4.1 below. The process plans of the jobs are fixed in sequence: Operation 1 then Operation 2. Table 4.2 shows part of a non-delay schedule generation example.

Job	Operation 1	Operation 2
J1	6/1	8/2
J2	4/1	1/2
J3	8/2	6/1
J4	5/2	10/1
J5	3/1	4/2
J6	2/1	4/2

Table 4.1: Process plan for the jobs (for the operation, x/y means x time units at station y).

The first 4 stages of the scheduling processes are shown in Table 4.2 below.

At stage t=1,

Step 1. The schedulable operations in S_t are: J1 to be processed in workstation WS1 for its first operation, J2 to be processed in workstation WS1, J3 to be processed in workstation WS2 for its first operation...etc. (i.e. all the 'operation 1' entries in Table 4.1)

Step 2. $\sigma^*=0$, and since σ^* occurs in both WS1 and WS2, so we just select one (WS1) arbitrarily.

Stage	Station next free at: Station1		Station2	Schedulable Operation					Scheduled Operation
t	M1	M3	M2	o_k in S_t	σ_k	ϕ_k	σ^*	Priority	o_j in P_t
1	0*	0	0	J1, WS1	0	6	0	14	J1, WS1
				J2, WS1	0	4		5	
				J3, WS2	0	8		14	
				J4, WS2	0	5		15	
				J5, WS1	0	3		7	
				J6, WS1	0	2		6	
2	6	0*	0	J2, WS1	0	4		5	
				J3, WS2	0	8		14	
				J4, WS2	0	5		15	
				J5, WS1	0	3	0	7	J5, WS1
				J6, WS1	0	2		6	
				J1, WS2	6	14		8	
3	6	3*	0	J2, WS1	3	7		5	
				J3, WS2	0	8		14	
				J4, WS2	0	5	0	15	J4, WS2
				J6, WS1	3	5		6	
				J1, WS2	6	14		8	
				J5, WS2	3	7		4	
4	6	3*	5	J2, WS1	3	7		5	
				J3, WS2	5	13		14	
				J6, WS1	3	5	3	6	J6, WS1
				J1, WS2	6	14		8	
				J5, WS2	5	9		4	
				J4, WS1	5	15		10	
5	6	5*	5	J2, WS1	5	9		6	

Table 4.2: Part of a non-delay schedule generation example.

Step 3. Here we have 4 jobs, J1, J2, J5 and J6 that all satisfy: 1) it requires M* (WS1), and 2) $\sigma_j = \sigma^*$. The choice is made by applying the heuristic selection or priority rules such as, SPT (shortest processing time), EDD (earliest due date), MOPNR (most operations remaining) and MWKR (most work remaining) etc. MWKR is used in this example and the priority of each job at each stage is listed in Table

4.2 (the priority is calculated as a job's total processing time remaining). Which priority rule to use is dependent on the measure of schedule desirability. For example, SPT might be a good candidate for minimizing mean flow times, while EDD would be a reasonable rule to use for minimizing tardiness. If there is still a tie among some jobs after a priority rule is applied, a job is selected randomly. In this case, J1 is selected for WS1 since it has the highest priority (14).

Step 4. (J1, WS1) is added to the scheduled operations set P_t, and deleted from S_t. The next operation of J1 is (J1, WS2) and is added to S_{t+1}. t is incremented by 1.

At stage t=2;

Step 2. Since WS1 has 2 machines, so after a job, J1, is added to one of its machines, say M1, it still has one machine, M3 being available at time 0. Therefore, the earliest time that J2, J5 and J6 could be started is still 0. Again, $\sigma^*=0$ and it occurs in both WS1 and WS2, we arbitrarily select one (WS1).

Step 3. J2, J5 and J6 all satisfy: 1) it requires M* (WS1), and 2) $\sigma_j = \sigma^*$. By applying the MWKR priority rule, J5 is selected.

Step 4. (J5, WS1) is added to the scheduled operations set P_t, and deleted from S_t. The next operation of J5 is (J5, WS2) and is added to S_{t+1}. t is incremented by 1.

At stage t=3:

Step 2. Now both machines of WS1 are assigned to a job, and the earliest one to be free is M3 at time=3. Jobs waiting to be processed in WS1 can start their operation no sooner than time =3. Therefore, the earliest start time for J2 and J6 has to be changed from 0 to 3, and their corresponding earliest finish time (ϕ_k) is changed to 7 and 5, respectively. So at this stage, $\sigma^*=0$ and it occurs in WS2.

Step 3. Both J3 and J4 satisfy the conditions. By applying the priority rule, J4 is selected.

Step 4. (J4, WS2) is added to the scheduled operations set P_t, and deleted from S_t. The next operation of J4 is (J4, WS1) and is added to S_{t+1}. t is incremented by 1.

At stage t=4:

Step 2. Here WS2's next available time will be at time=5, so J3's earliest start and finish time have to be changed to 5 and 13, respectively. In this stage, $\sigma^*=3$ and it occurs in WS1.

Step 3. Both J2 and J6 satisfy the conditions. By applying the priority rule, J6 is selected.

Step 4. (J6, WS1) is added to the scheduled operations set P_t, and deleted from S_t. The next operation of J6 is (J6, WS2) and is added to S_{t+1}. t is incremented by 1.

At stage t=5:

.

.

.

The scheduling process continues until stage t=12 (n*m, where n=6 jobs, m=2 machines). The Giffler and Thompson algorithm disclosed in (French 1990) was made under the following assumptions:

1) Each job has m distinct operations, one on each machine;

2) There is only one of each type of machine;

3) Machines never breakdown and are available throughout the scheduling period.

But in our system, we can have multiple machines with same function in a station (to provide some functional redundancies in case some machine breaks down during the scheduled period). Therefore, instead of having only one type of each type of machine, we have only one type of each type of station. So steps 2 and 3 of the algorithm stated in the problem specification are modified as,

Step 2 Find $\sigma^* = \min_{o_k \text{ in } S_k} \{\sigma_k\}$ and the workstation WS* on which σ^* occurs. If there is a choice for WS*, choose arbitrarily.

Step 3 Choose an operation o_j in S_t such that

(1) it requires WS*, and

(2) $\sigma_j = \sigma^*$.

Our next step is to investigate how we can interpret the scheduling algorithm stated before from a production system's perspective. Referring to the example, we see that at stage t=1, S_1 contains (J1, WS1), (J2, WS1), (J3, WS2), (J4, WS2), (J5, WS1) and (J6, WS1). In a production system, when the workstation that a job needs for processing its next operation currently has no machining resource available, the job usually waits in the queue of the workstation for its turn to be processed. So when viewed from a production system's perspective, what S_1 really contains are jobs 1, 2, 5 and 6 from (the queue of) workstation 1, and jobs 3 and 4 from (the queue of) workstation 2.

This means that what we did for step 1 was to send all the jobs to queue in the station which corresponds for their first operation. Then we found a job with the minimal earliest start time and select the workstation that holds the job. In the example, since all jobs have the same earliest start time (0), we selected WS1 arbitrarily. From WS1, since there was more than one job with the same minimal earliest start time, we applied the MWKR priority rule to select J1 and assigned it to one of WS1's machines, M1, and added the operation record of (J1, WS1) to P_t. We then moved on to stage t=2.

At stage t=2, we see that (J1, WS2) was in S_t, which means that after the job J1 was assigned to the machine (M1) of WS1 at stage t=1, it was sent to the workstation (WS2) which corresponds to its next operation. We then repeated the same procedures that we did in stage t=1, and assigned J5 to M3 of WS1 and added the operation record of (J5, WS1) to P_t. We then moved on to stage t=3.

At stage t=3, we can see that in addition to the fact that (J5, WS2) was already added to S_t, the earliest start time of jobs 2 and 6 in workstation WS1 were also changed. A workstation's earliest available time is equal to $\min_{k=1\text{-}m}$ {earliest available time of machine k}, m= total number of machines the workstation has. So after a job is assigned to a machine in a workstation, the workstation's earliest available start time might be changed. As it was at stage = 2, after the selected WS1 had assigned a job to its machine (M3), its earliest available time was changed (from 0 to 3), so the queuing jobs (J2, J6) in WS1 also had to change their earliest possible start time accordingly. Again, similar procedures (as what we did in the previous stages) were carried out in stage 3 and the following stages until a complete schedule was generated.

Referring to the above example, we can re-interpret the scheduling algorithm 4.1 from a production system's perspective as the Algorithm 4.2 listed below:

Algorithm 4.2:

Step 1. Send the jobs to the stations that correspond to their first operation.

Step 2. From all the workstations, find a queuing job that has the minimal earliest start time σ^*, select the workstation WS* that holds that job. If 2 or more queuing jobs from different workstations have the same σ^*, choose one workstation arbitrarily.

Step 3. From the selected workstation WS*, find a job whose earliest start time is σ^*. If there are more than one jobs that have the earliest start time as σ^*, use priority rule to choose one. If there is still a tie after the priority rule is applied, a second priority rule can be used or the job can be chosen randomly.

Step 4. Assign the job to the first available machine of the workstation WS*. Send the job to the workstation that corresponds to its next operation. Add the scheduled operation record to P_t. Update the earliest start time of the workstation WS*'s remaining queuing jobs accordingly. Increase t by 1.

Step 5. If there are any operations left unscheduled, go to Step 2. Otherwise, stop.

4.3.2 Conceptual Model for the Scheduling Problem

The first step in the analysis process is to develop the conceptual model for the scheduling problem so that the objects that are involved in the scheduling processes, and the associations of these objects can be identified.

> "The quintessential object-oriented step in analysis or investigation is the decomposition of the problem into individual concepts or objects—the things we are aware of. A conceptual model is a representation of concepts in a problem domain" (Larman 1997).

"One widely used technique for identifying objects in an object-oriented systems analysis (Rumbaugh et al. 1991) is to extract the nouns from a narrative description of the desired system behavior" (Parunak et al. 1997). By applying this approach, objects that we identified from the scheduling algorithm 4.2 are:

<u>Job</u> <u>Operation</u> <u>Machine</u> <u>Partial Schedule</u> <u>Queue</u>
<u>Scheduled Operation</u> <u>Unscheduled Operation</u> <u>Station</u>

Referring to Algorithm 4.2, we need to have some staff roles to help accomplish the scheduling steps and coordinate the activities of the other objects. For instance, in the scheduling processes, because the resource (station, machine) and job objects do not have the knowledge of the global time, we need to have someone to coordinate the activities of these objects. Therefore, a new concept, the 'System Mediator' (Maturana & Norrie 1996), is added to the conceptual model of the scheduling problem. Also, since each machine can only perform one operation type, we will add the concept 'Operation Type' to the conceptual model to help describe the static model of the scheduling problem. As resulted, the conceptual model will also include the following objects:

<u>System Mediator</u> <u>Operation Type</u>

The conceptual model for the scheduling problem is shown in Figure 4.1, and is represented in the UML (Unified Modeling Language) notation. The Unified Modeling Language is a notation for modeling systems using object-oriented concepts (Larman 1997). In Figure 4.1, it states that:

- 1 job has 1 or many unscheduled operation(s)
- 1 job has 0 or many scheduled operation(s)
- 1 job consults 1 system mediator
- An unscheduled operation is an operation
- A scheduled operation is an operation
- 1 scheduled operation is processed by 1 machine
- 1 scheduled operation is processed in 1 station
- 1 partial schedule records 1 or many scheduled operation(s)
- 1 or many scheduled operation(s) belongs to 1 job

- 1 station is responsible for 0 or many scheduled operation(s)
- 1 station has 1 or many machine(s)
- 1 station has 1 queue
- 1 queue holds 0 or many job(s)
- 1 system mediator provides station info to 1 or many job(s)
- 1 system mediator coordinates the (scheduling) activities of 1 or many station(s)

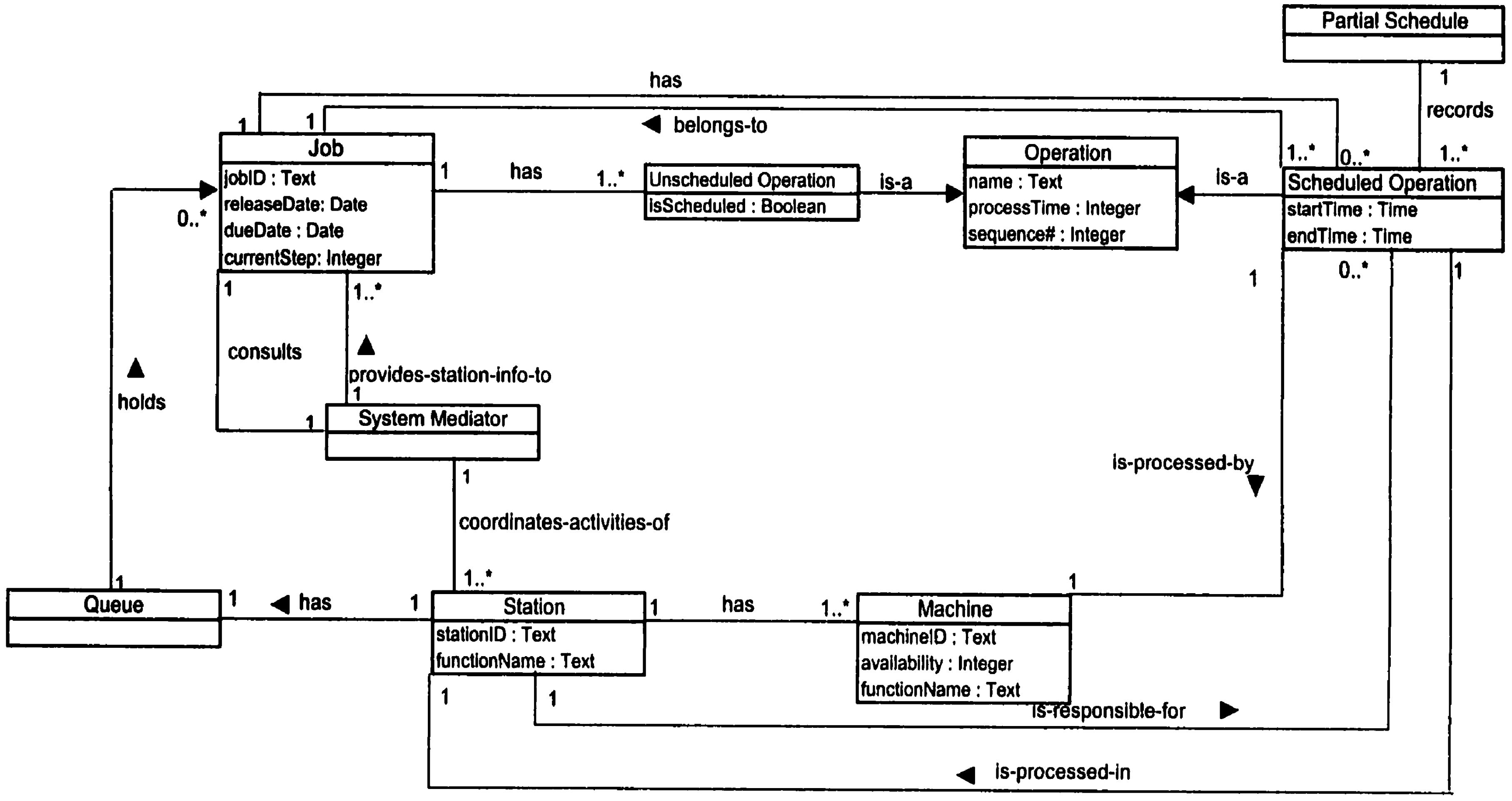
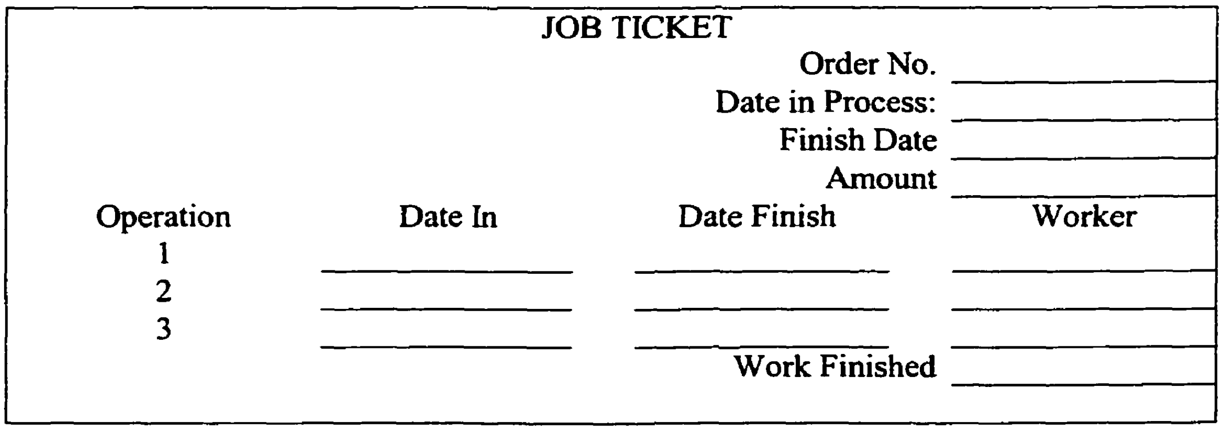


Figure 4.1: The conceptual model for the scheduling problem.

As shown in the conceptual model, we identified some of the attributes and associations of the objects involved in the scheduling processes. Attributes/associations of the objects are identified mostly from examples in the real world. For instance, Figure 4.2 shows a job ticket that can usually be found in the real manufacturing system (Voris 1966). In the job ticket, we can see that the data related to the jobs include the identification of the job, the release date and due-date of the job, and the operations that the job contains etc., and data that are related to the operation include the name of the operation, the start and finish time of the operation, and the worker/machine that processed the operation etc. These data are shown as attributes or associations of the job and operation objects in Figure 4.1. Attributes and associations of the other objects are also identified with similar approach.



JOB TICKET

		Order No.	
		Date in Process:	
		Finish Date	
		Amount	
Operation	Date In	Date Finish	Worker
1			
2			
3			
		Work Finished	

Figure 4.2: Job ticket to record progress (Voris 1966).

4.3.3 Interaction Models for the Scheduling Problem

Before we proceed to design the interaction models for the scheduling problem, we will first briefly introduce some of the GRASP patterns that we might apply in assigning responsibilities to objects. GRASP stands for General Responsibility Assignment Software Patterns. These patterns describe fundamental principles of assigning responsibilities to objects (Larman 1997). The patterns that are most relevant to our application are:

1) Expert – "Assign a responsibility to the information expert- the class that has the information necessary to fulfill the responsibility" (Larman 1997).

2) Creator - "Assign class B the responsibility to create an instance of class A if one of the following is true: 1) B aggregates A objects, 2) B contains A objects, 3) B records instances of A objects, 4) B closely uses A objects, 5) B has the initializing data that will be passed to A when it is created" (Larman 1997).

3) Low Coupling – "Assign a responsibility so that coupling remains low. Coupling is a measure of how strongly one class is connected to, has knowledge of, or relies upon other classes. A class with low (or weak) coupling is not dependent on too many other classes" (Larman 1997).

4) High Cohesion – "Assign a responsibility so that cohesion remains high. Cohesion is a measure of how strongly related and focused the responsibilities of a class are. A class with highly related responsibilities, and which does not do a tremendous amount of work, has high cohesion" (Larman 1997).

To design the interaction models for the scheduling problem, we will go through each of the steps listed in the scheduling algorithm 4.2 to determine how the objects involved in the scheduling processes can collaborate to fulfill all the scheduling steps. Responsibilities will then be assigned to the objects accordingly.

STEP 1

To send a job to the station for its next operation, we first have to know what is the job's next operation. Referring to Figure 4.1, a job contains a number of unscheduled operations, and thus has the knowledge of its own process plan. Therefore, by applying the 'Expert ' pattern, we assign the job the responsibility to send itself to the station that corresponds to its next operation.

For a job to go to the station, first it has to know which station is responsible for processing its next operation. One way to do this is to have the job broadcast to all stations in the system to ask who can process its next operation, and have the corresponding station responded to the job. If a broadcasting facility is not implemented in the system, another way is to have the system mediator responsible for answering the job's question. This can be accomplished by having the station objects registered with the system mediator when the scheduling system starts. The system mediator will then have the knowledge of what stations are contained in the system, and what function each station can perform.

After the job knows which station can process its next operation, it has to find out the earliest possible start time for the operation. A job's next operation's earliest possible start time is dependent on 2 factors, 1) the end time of the job's previous operation, and 2) the earliest available time of the station that is responsible for its next operation. So a job has to ask the corresponding station what its earliest available time will be, then it must check the end time of its previous operation (if it's the job's first operation, the job might check its release time), then decide when the earliest possible start time of its next operation will be.

When a job enters a workstation, the station might insert the job into its queue in accordance to some ranking rules. Referring to steps 2 and 3 in Algorithm 4.2, a station will choose a job with the minimal earliest possible start time or with the highest priority to be processed first. So the station will rank the arriving jobs by their earliest possible start time. And if there is a tie, the jobs will be ranked by their priority, and if there is still a tie, the jobs will be ranked by first-come-first-serve rule. Therefore, when a job arrives at a workstation, the station needs to know about the job's earliest possible start time and priority to insert the job into its queue.

Figure 4.3 below shows the interaction model for sending a job to the station for its next operation. As is shown in the figure, in Step 1, the jobs will be asked to move to the stations for their first operation. Upon receiving the NextProcess() message, the job will,

1) Find out its next operation (op) from its collection of unscheduled operations.

2) Ask the system mediator which station (st) is responsible for its operation.

3) Ask the station about its earliest available time (aTime).

4) Find out the priority (prior) for its operation.

5) Find out the earliest possible start time for its operation.

6) Add itself to the station.

6.1) The station inserts the job to its queue.

7) Mark the unscheduled operation (op) as scheduled.

8) Record (create) the scheduled operation.

9) Add the scheduled operation (op1) to its collection of scheduled operation.

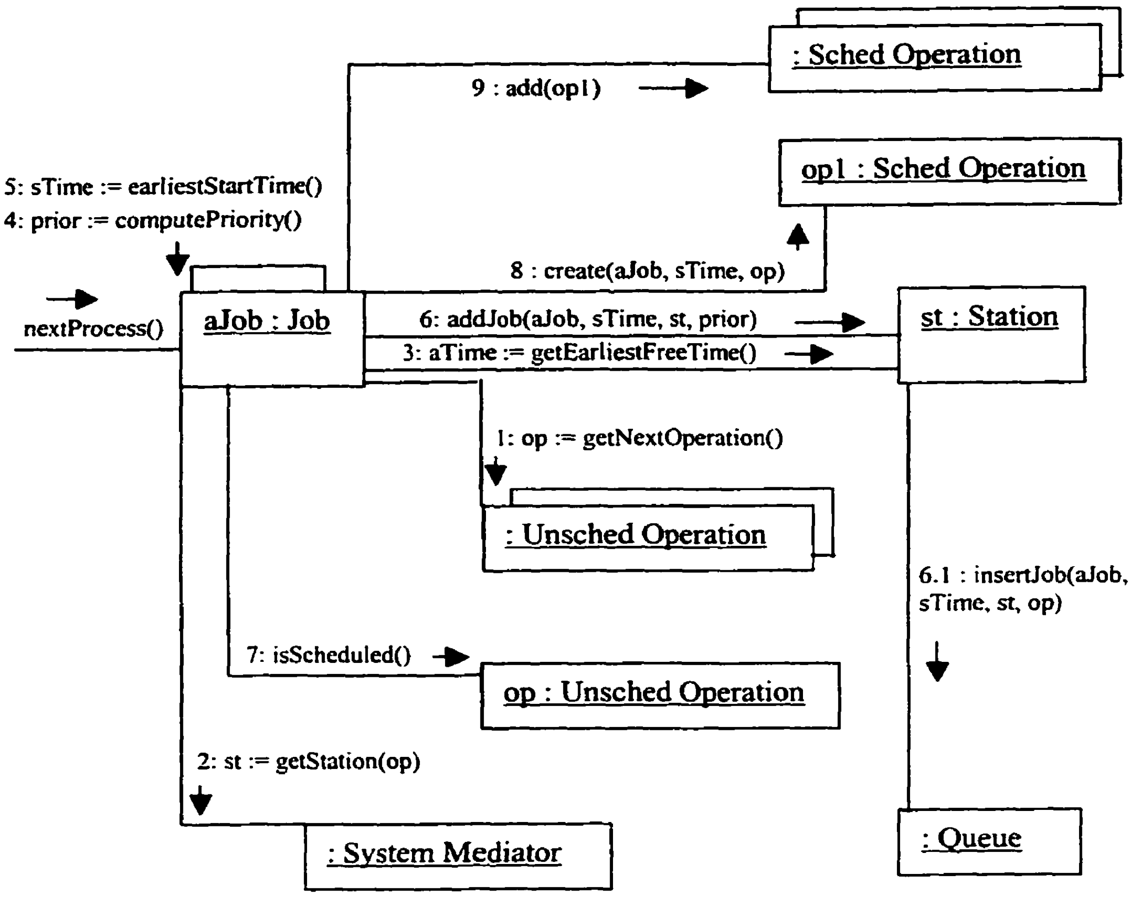

: Sched Operation
9 : add(op1)
op1 : Sched Operation
5: sTime := earliestStartTime()
4: prior := computePriority()
8 : create(aJob, sTime, op)
nextProcess()
aJob : Job
6: addJob(aJob, sTime, st, prior)
3: aTime := getEarliestFreeTime()
st : Station
1: op := getNextOperation()
: Unsched Operation
6.1 : insertJob(aJob, sTime, st, op)
7: isScheduled()
op : Unsched Operation
2: st := getStation(op)
: System Mediator
: Queue

Figure 4.3: The collaboration diagram for sending a job to the station for its next operation.

STEP 2

To find the job with the minimal earliest possible start time, the system mediator can ask each of the jobs about their earliest start time, then choose the one that has the minimal earliest possible start time. The system mediator can then ask the selected job which station is responsible for its operation. Upon receiving the answer, the system mediator can then instruct the selected station to carry out the tasks for steps 3 and 4 in Algorithm 4.2 (to be explained later).

The other way to find out the σ^* and WS* is to have the system mediator ask each station about the earliest start time for its next job. We should note that in step 1, when a job arrives at a station, the station ranks the job by its earliest possible start time and priority. Therefore, the first job in the station's queue will have the minimal earliest start time among all the jobs in the queue. So upon receiving the system mediator's request, the station will answer the system mediator with the earliest possible start time that belongs to its first job in queue. After collecting the answers from all stations, the system mediator can then select the station that has the minimal earliest possible start time and instruct the station to carry out the tasks for steps 3 and 4.

The second method will be used in this thesis because the production system usually will have much smaller number of stations than jobs. As a result, it would be faster for the system mediator to find the job with minimal earliest possible start time by consulting all the stations than by consulting all the jobs. Moreover, the first method will couple the system mediator to the knowledge of the jobs in the system. And by the 'Low Coupling' pattern, the second method is chosen. Figure 4.4 and 4.5 show the collaboration diagrams for method 1 and 2, respectively. In Figure 4.5, it shows that:

1) The system mediator needs to find out which station has the job that has the minimal earliest possible start time.

1.1) The system mediator consults each station about the earliest possible start time for their first job in queue (1.1.1).

2) The system mediator informs the selected workstation to start the tasks for steps 3 and 4 in Algorithm 4.2 (to be explained next).

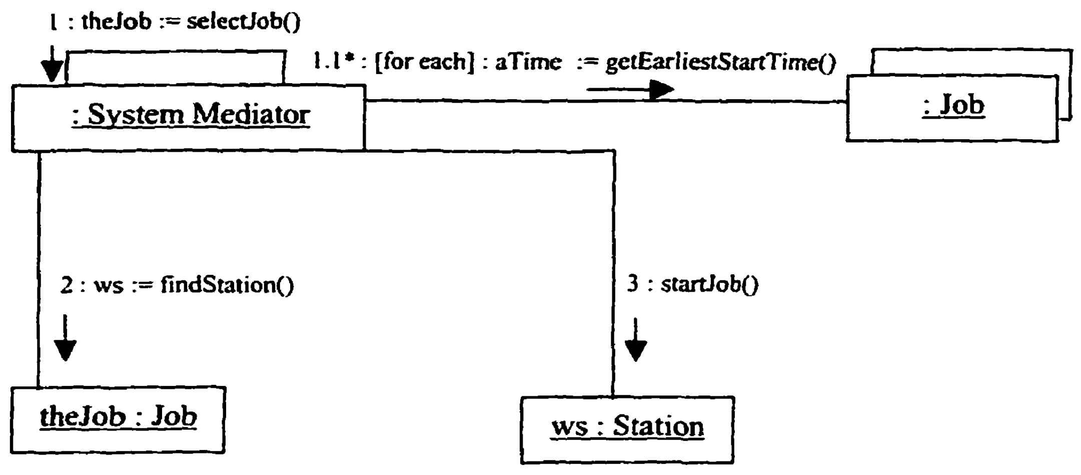


Figure 4.4: The collaboration diagram for method 1.

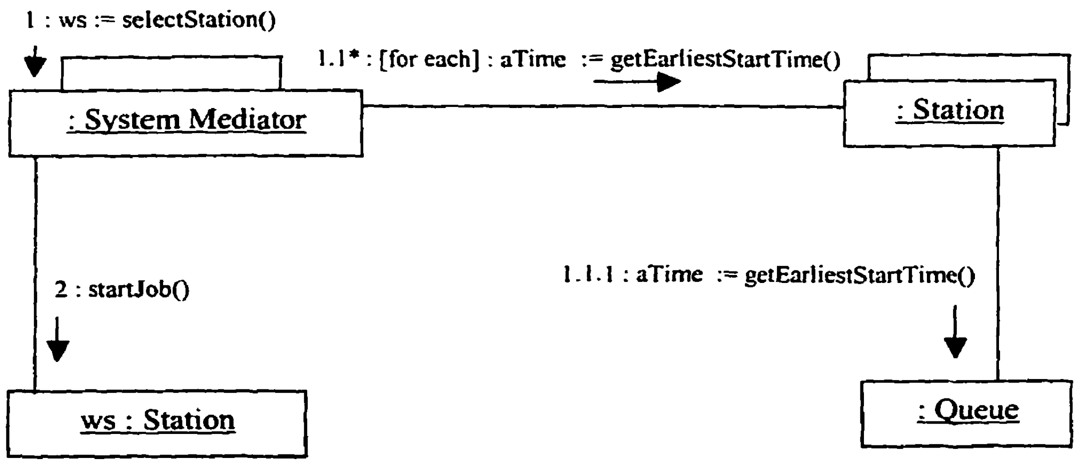

1 : ws := selectStation()
1.1* : [for each] : aTime := getEarliestStartTime()
: System Mediator
: Station
2 : startJob()
1.1.1 : aTime := getEarliestStartTime()
ws : Station
: Queue

Figure 4.5: The collaboration diagram for method 2.

STEP 3

As described in the previously steps, when a job enters a workstation, the workstation will insert the job in its queue in according to their earliest possible start time and priority. Therefore, to find a job in its queue with the minimal earliest possible start time (or highest priority), the station just picks the first job in its queue.

STEP 4

The station finds the first available machine and then assigns the selected job to that machine. After that, the station will inform the partial schedule about the scheduled operation. Also, the station will record the scheduled operation that it will process. The station will then update its earliest available time and have the remaining jobs in its queue update their earliest possible start time accordingly. Finally, the station will notify the selected job to proceed to its next operation, and instruct the queue to re-rank the remaining queuing jobs. The reason for re-ranking the remaining queuing jobs is that, as in the example described in §4.3.1, in stage 3, we can see that in workstation 2, J3 and J4's earliest possible start time = 0, and J5's earliest possible start time = 3, so J5 will queue behind J3 and J4. But after J4 is selected and assigned to machine M2, WS2's earliest available time changed from 0 to 5, so J3 and J5 will all update their earliest possible start time accordingly (change the time from 0 to 5). In the example, because J3 has higher priority than J5, so J5 is still queued behind J3. But if J3's priority were lower than J5, then after updating their earliest possible start time, J5 would have queued in front of J3, instead of queuing behind it. Figure 4.6 below shows the collaboration diagram for steps 3 and 4 in Algorithm 4.2.

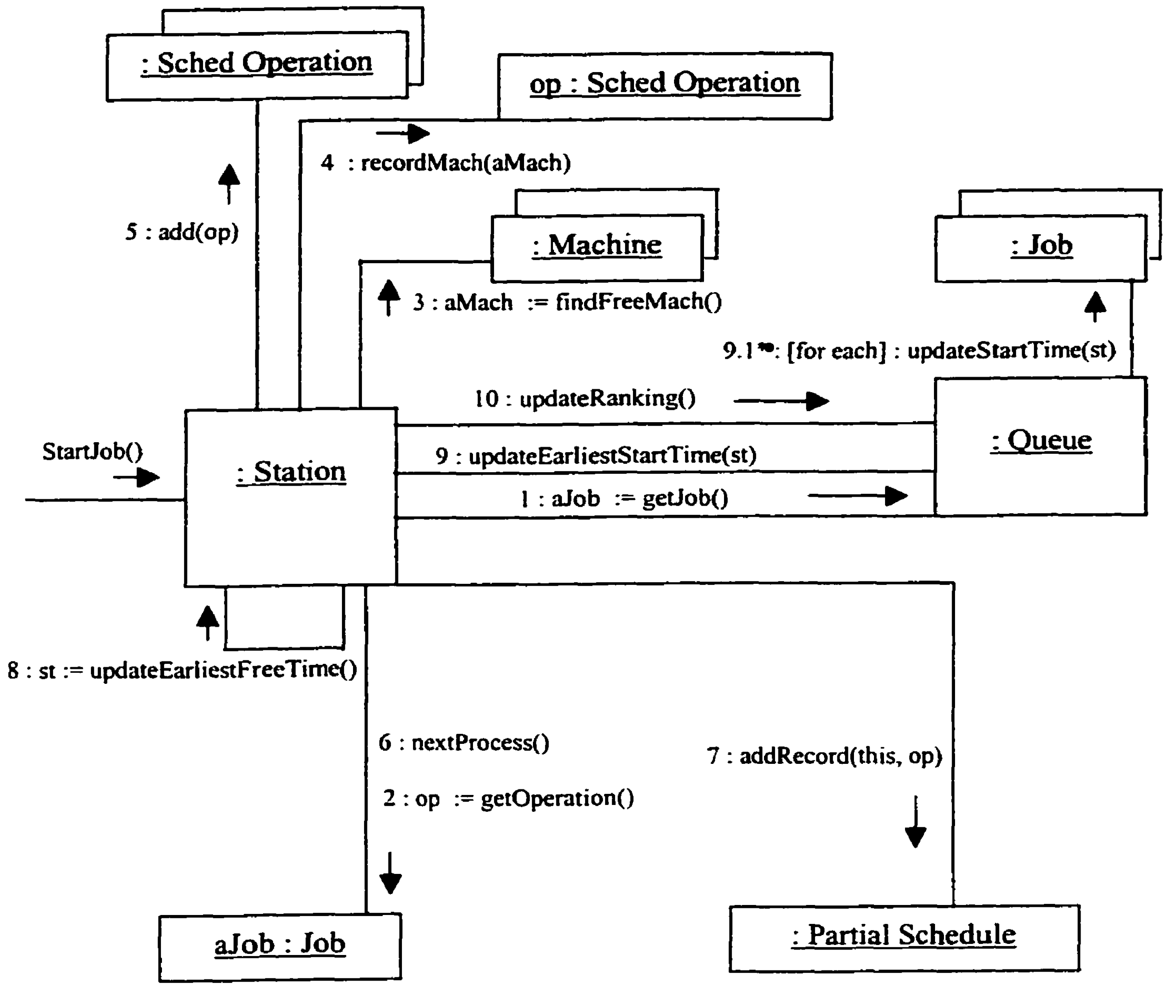


Figure 4.6: Collaboration diagram for steps 3 and 4 in Algorithm 4.2.

Figure 4.6 shows that, upon receiving the message StartJob() from the system mediator,

1) The station selects a job (aJob) from its queue.

2) The station gets the operation (op) information from the job.

3) The station finds out which of the machines (aMach) that it contains is available.

4) The scheduled operation (op) records the machine that will process the operation.

5) The station records the scheduled operation (op) that it will process.

6) The station instructs the selected job (aJob) to proceed to its next operation.

7) The station notifies the partial schedule about the scheduled operation. The partial schedule will store the related information about the scheduled operation (e.g., the operation's start/end time, the machine/station that will process the operation, etc.) in a permanent record (e.g., a relational database, a file record, etc.). This record will then be used to generate the production schedule.

8) The station updates its earliest available time (st).

9) The station asks the queue to have its queuing jobs updated their earliest possible start time (9.1).

10) The station asks the queue to re-rank its remaining queuing jobs after they have updated their earliest possible start time.

STEP 5

The system mediator increments the stage t by one. Steps 2~4 are repeated until no schedulable operations are left.

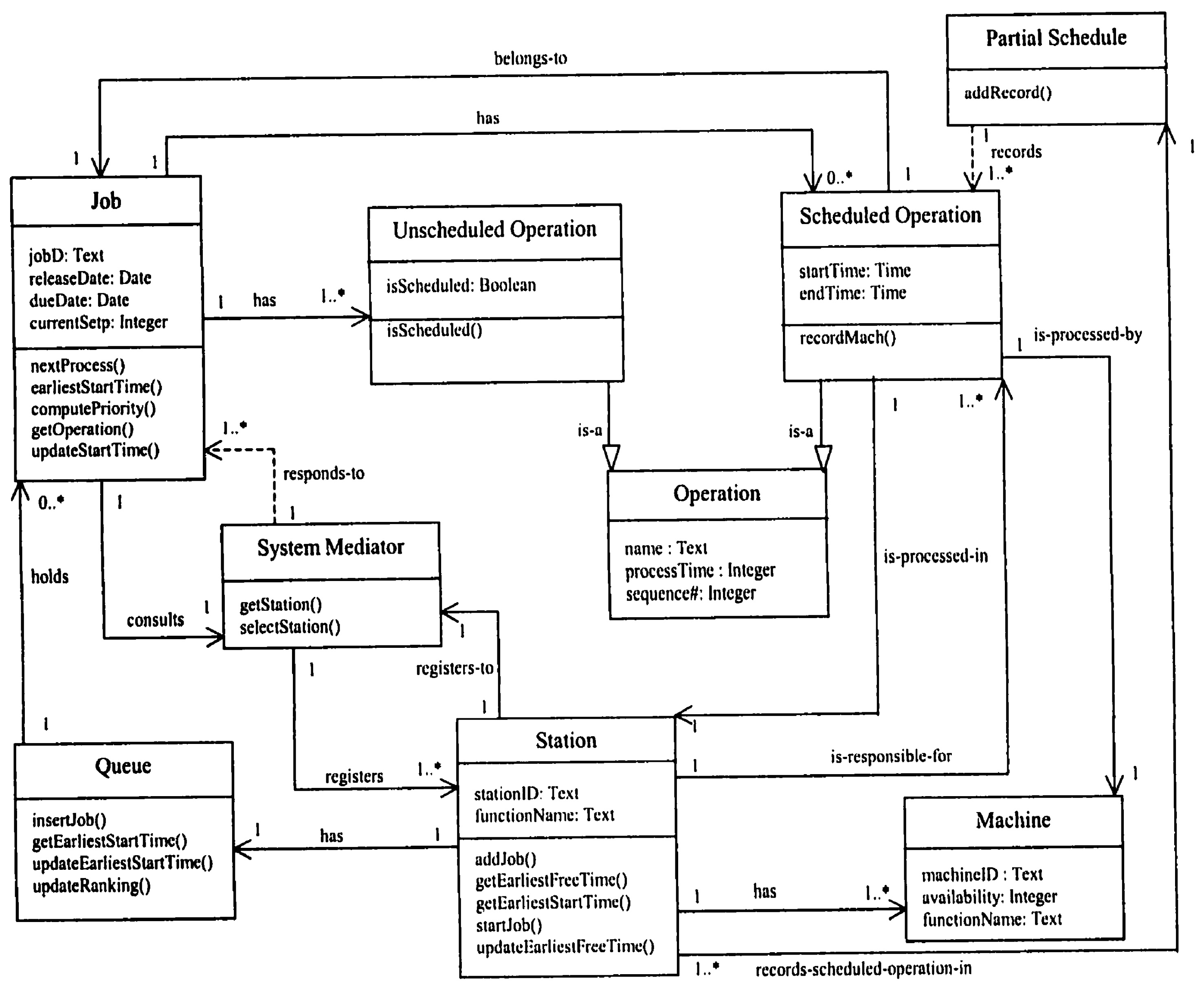


Figure 4.7: The class diagram for the scheduling problem.

With the completion of the interaction diagrams for the scheduling problem, we can then proceed to create the design class diagrams to "identify the specification for the software classes which participate in the software solution" (Larman 1997). The class diagram for the scheduling application is shown in Figure 4.7. The class diagram not only shows the attributes and methods of the classes, it also shows the associations, navigability and dependency relationships between the classes.

> "Each end of an association is called a role, and in the design-oriented diagrams the role may be decorated with a navigability arrow.... The usual interpretation of an association with a navigability arrow is attribute visibility from the source to target class.... The UML includes a general dependency relationship which indicates that one element has knowledge of another element. It is illustrated with a dashed arrowed line. In class diagrams the dependency relationship is useful to depict non-attribute visibility between classes: in other words, parameter, global, or locally declared visibility" (Larman 1997).

For example, referring to Figure 4.7, for the scheduled operation role, it needs to have the knowledge of when the operation is scheduled to start, the process time of the operation etc. In addition to that, the scheduled operation needs to know which job it belongs to, which station it will be processed in, and by which machine. Therefore, the scheduled operation role has the attribute visibility to the job, station and machine roles. And for the job role, for instance, a job needs to have the attribute visibility to the system mediator, since jobs have to ask the system mediator about the stations that can process their operations. But the system mediator does not need to have attribute visibility to the jobs. When a job sends a message to the system mediator, it will tell the system mediator who it is (as parameter). The system mediator will then return the answer to that job. Therefore, the system mediator only needs to have the parameter visibility (dependency relationship, represented as dashed arrowed line) to the job role. The classes in the class diagram, their attributes, methods, and associations etc. are mainly identified from the conceptual model and interaction diagrams that we have developed in the previous sections. After we have built the software solution for the scheduling problem, we will discuss the design of the distributed scheduling and control agents in next section.

4.4 Identifying Control Agents for the Distributed Control System

As mentioned before, the objectives in this research are mainly concerned with the control problems related to the distributed (multi-agent) control systems. To build the distributed control system, instead of determining the control agents first and then structuring the scheduling algorithm around these agents, we use the object-oriented methodology to explore the possible agent candidates from the scheduling algorithm. In §4.3, we used the object-oriented analysis and design approach to identify the objects that participate in the scheduling processes (scheduling algorithm 4.2), and the associations between these objects. We then assigned responsibilities to these objects so that they can collaborate to fulfill all the required scheduling procedures.

Some researchers have used a similar approach to explore the possible agent candidates while developing the agent architecture for shop floor scheduling and control. For instance, in (Parunak et al. 1998b), it was described that:

> "Previous research on agent-based factory control and scheduling (including our own) differs widely on what is represented as an agent.... In selecting our agents, we want to begin with the broadest possible of candidates.... As described in (Parunak 1995), we identify candidate agents by constructing a set of declarative sentences describing the domain. The nouns in such a sentence are candidate agent instances, and their cases represent agent classes".

Also, in (Wooldridge & Jennings 1999), it was mentioned that:

> "We expect an agent-oriented view of software to complement – not replace – the object-oriented view. Developers will typically implement agents using object-

oriented techniques, and there will usually be fewer agents in the system than objects".

In our model, some of the roles (objects) are potential agent candidates, while others are only (software) objects for software design purposes. For instance, referring to Figure 4.7, we can see that there are some roles such as station, machine, and job that have a physical correspondence in the real manufacturing system. These physical entities usually have a control agent associated with them (in a distributed control system). In Figure 4.7, the station, machine and job roles only have the scheduling responsibilities. However, control responsibilities (to be discussed in next chapter) can also be added to these roles so that they can become the control agents that are responsible for the scheduling and control behaviors of their physical correspondences in the real world. Other objects might not have a physical correspondence in the real world, but they still can be agent candidates because they can provide some distinct services (behaviors) or play an active role in the scheduling process. For instance, the system mediator not only provides station information to the jobs, it is also responsible for initiating some scheduling steps during the scheduling processes. Therefore, the system mediator can act as a staff agent.

For objects such as 'scheduled operation', 'unscheduled operation', 'partial schedule', these objects in our model are only used for capturing certain data and providing the accessing methods (Get(), Set()) to the data they stored. Therefore, they will be served as software objects only. For instance, 'scheduled operation' stores the information regarding an operation's process time, start time, the job it belongs to, and the station/machine that will process the operation. After the schedule is generated, the job control agent will contain a collection of the 'scheduled operation' objects so that for each operation, the job agent can determine which station to visit at what time to have its operation processed. The station control agent will also contain a collection of 'scheduled operation' objects as its task list, so that it will know that it will know when and where to load each job to start processing the operation. Since both the job and station agent need to have scheduled operation knowledge, making the 'scheduled operation' an object is for software reusability purpose and can save some coding. It should be noted that whether certain roles (objects) are agent candidates or not, sometimes it is dependent on the control algorithms that are adopted. For instance, in (Parunak et al 1997, 1998b), in their AARIA agent architecture, the operation is modeled as the 'Unit Process' agent, and is assigned with the responsibilities so that it can initiate activities and is responsible for marshaling the inputs and resources needed to execute an operation.

In distributed (multi-agent) control systems, part and resource control roles are usually used to control the production activities of their physical counterparts in the production plant. Since most of these roles are also identified in our scheduling algorithm, we will then implement these roles as the control agents that are responsible for the scheduling and control for their physical correspondence in the production system. Therefore, in our distributed control system, we will have control agents such as workstation agents, job agents and machine agents. Also, we will have the staff agent, the system mediator, to help accomplish the scheduling tasks. In the next section, we will implement these control agents in an experimental testbed to carry out the scheduling and control responsibilities for a simulated shop floor production system. The experimental testbed will be used to test/evaluate the system performances under different control strategies.

4.5 The Experimental Testbed

Figure 4.8 shows the experimental model that will be used for testing and evaluating different control methodologies. In our experimental model, a simulated production system was set up in Arena (a discrete-event simulation software). The resource, order and production databases were created in the format of MS-Access ODBC database. The Microsoft Visual C++ block served as a control module.

In the experimental model, we have entities like workstations, machines and jobs set up in Arena to represent the production entities in the real production system. The workstation, machine and job entities in the production system (Arena model) are

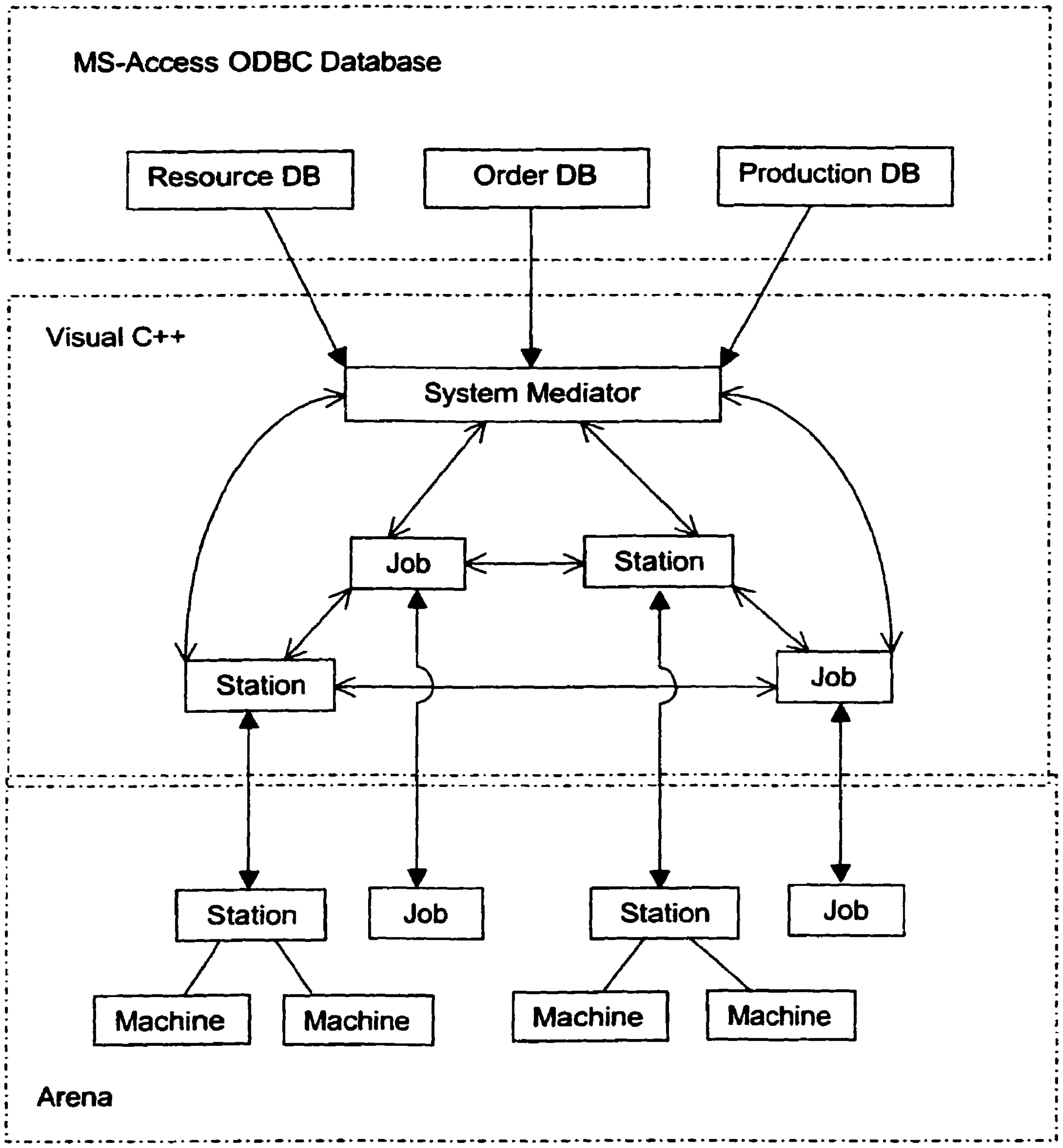


Figure 4.8: The control structure for the experimental model.

controlled by the corresponding control agents residing in the Visual C++ control module. When the production system starts, a system mediator will be created in the C++ module. The system mediator will then access the resource, order and production databases to create and initialize the workstation and job control agents. (The advantages of having the system mediator access different databases to retrieve the necessary information are that: 1) this practice can more likely reflect the situation in a real manufacturing system, wherein the order and production information might be created by different departments and stored in different databases, and 2) we don't need to modify the experimental model's software program when we make changes to the manufacturing system's physical configuration, the orders or the production plans of the jobs.) These agents will then be responsible for the control of their counterpart entity in the Arena model.

From hereon, entity X in the C++ module will be referred as X agent, and its counterpart in the Arena model will be referred as X. After all the agents in the C++ module are created and initialized, the three essential elements of the shop floor control will be implemented as follows:

1) Scheduling

To do the scheduling, the system mediator, workstation agents and job agents will interact with each other, in the same way as the system mediator, workstation and job objects that were defined in §4.3 interacted, to generate a production schedule for all the jobs. Referring to the example described in §4.3.1, after the scheduling process is done, workstations WS1 and WS2 will have a scheduled task list as shown in Figure 4.9 and each job will have a scheduled operation record (processing plan) as shown in the Table 4.3.

J1	Operation 1	Operation 2
Start Time	0	13
Finish Time	6	21
Station	WS1	WS2
Machine	M1	M2

Table 4.3: The scheduled operation record (processing plan) for job J1.

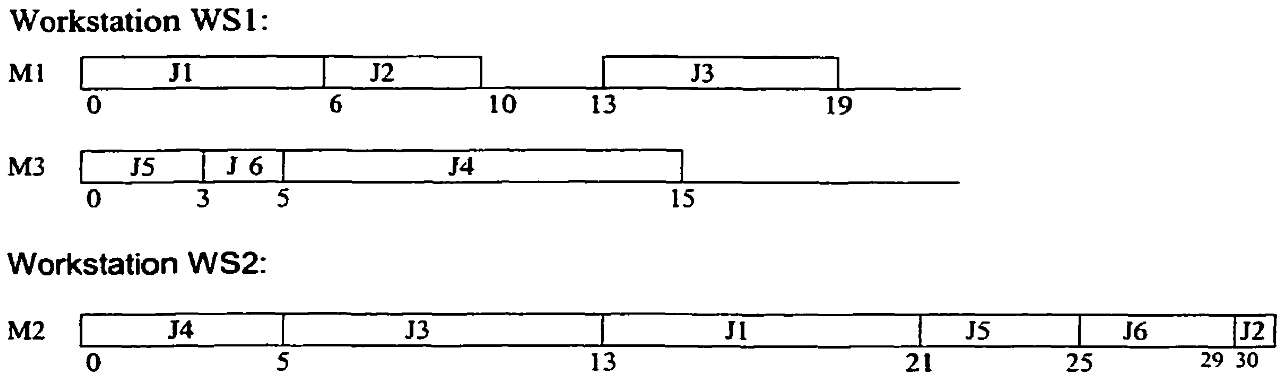


Figure 4.9: The scheduled task lists for workstations WS1 and WS2.

2) Dispatching

To do the dispatching, each workstation agent will follow the scheduled task list and instruct its counterpart workstation in the Arena model to perform the operations accordingly. For example, referring to Figure 4.9, WS1 agent will instruct workstation WS1 to have machines M1 and M3 start operating jobs J1 and J5, respectively, at time 0. So at time=0, WS1 will look for jobs J1 and J5 in its queue and load them to machines M1 and M3, respectively. And for the job agents, each agent will follow their scheduled operation record and instruct their counterpart in Arena model to go to the scheduled workstation for its next operation. For example, referring to Table 4.3, J1 agent will have job J1 in the Arena model to go (and wait) to workstation WS1 for its first operation. After the operation is done, J1 agent will then instruct job J1 to go to workstation WS2.

3) Monitor

When a machine starts or finishes an operation, it will inform its workstation, and the workstation will inform its counterpart control agent about the start/end time of the operation. The informed workstation agent will in turn pass that information to the job agent of the job being processed. For example, referring to Figure 4.9, when machine M1 starts processing job J1 at time 0, WS1 agent will be informed about the start time of the operation, and WS1 agent will record the data and pass that information to job J1 agent, who will also record the data. These procedures are needed so that the workstation and job agents will have the knowledge of the actual work progress of their counterparts in the production system. And when disturbances happen, these progress records will be used for rescheduling purpose. Also, this kind of data can be used to aid in securing labor costs for the job, and for tracing the source of some quality problems (Voris 1966). A workstation will also monitor the state of its machines (to see if there's any breakdown or restoration from failure). When disturbance happens, the workstation agent will then take an appropriate action to handle the situation, based on what control scheme is implemented in the system. When a workstation agent receives the message from its counterpart workstation, it will pass the data to its corresponding machine agents. Each workstation agent controls its contained machine agents. The main responsibilities of these machine agents are to record the start/finish time of each operation, and the status of its counterpart machine in the production system.

CHAPTER 5

CONTROL ALGORITHMS IN DISTRIBUTED SCHEDULING AND CONTROL SYSTEMS

Previous research on distributed scheduling and control has implemented scheduling algorithms using various approaches. Some researchers have implemented the distributed scheduling algorithms for real-time distributed scheduling (Duffie et al. 1986), wherein there are no pre-production schedules generated, and the distributed scheduling algorithms are used for making production decisions in real-time (dispatching). Other researchers have used the distributed scheduling algorithms to generate schedules in advance (Sousa et al. 1997), or used the algorithms and simulations to generate look-ahead schedules (Duffie et al. 1994).

Control systems that implement the real-time distributed scheduling approach and also control systems that generate advance schedules have different planning horizons (and also different degrees of control agents' commitments to the future plans). In control systems that generate advance schedules, control agents are committed to a common future plan that spans a certain planning period. As a result, "a schedule adds a level of rigidity to a manufacturing system" (Baker 1997) and limits the control system's flexibility and adaptability against disturbances. In control systems that implement real-time distributed scheduling (dispatching), production decisions are made in real-time and the control agents make no commitments to any future plan. As a result, these control systems are more flexible and adaptable to disturbances (Dilts et al. 1991, Bongaerts et al. 1998).

In this chapter, we will first use some experimental examples to demonstrate how control agents' commitments to different planning horizons can affect the control system's flexibility and adaptability against disturbances. Then experiments will be conducted to investigate and test the performance of a distributed scheduling and control system under various control strategies in a stochastic manufacturing environment. Results of the experiments will be discussed, and a conclusion will be presented in the last section of this chapter.

5.1 Control System Flexibility against Disturbances for Different Planning Horizons

In this section, we will develop three different models to test the flexibility against disturbances of the control systems that perform scheduling with different planning horizons. Table 5.1 shows the process plans of the jobs to be produced in the experimental models, and the disturbance will be modeled by having job J1 delayed in arriving in the production system by 16 minutes. Figure 5.1 shows the production schedule for the production system without any disturbance. The schedule was generated by using the non-delay scheduling algorithm (Algorithm 4.2) described in Chapter 4 with the Most-Working-Remaining (MWKR) priority rule. The experimental testbed described in §4.5 will be used for the tests.

Job	Operation 1	Operation 2
J1	6/1	8/2
J2	4/1	1/2
J3	8/2	6/1
J4	5/2	10/1
J5	3/1	4/2
J6	2/1	4/2

Table 5.1: Process plan for the jobs (for the operation, x/y means x time units at station y).

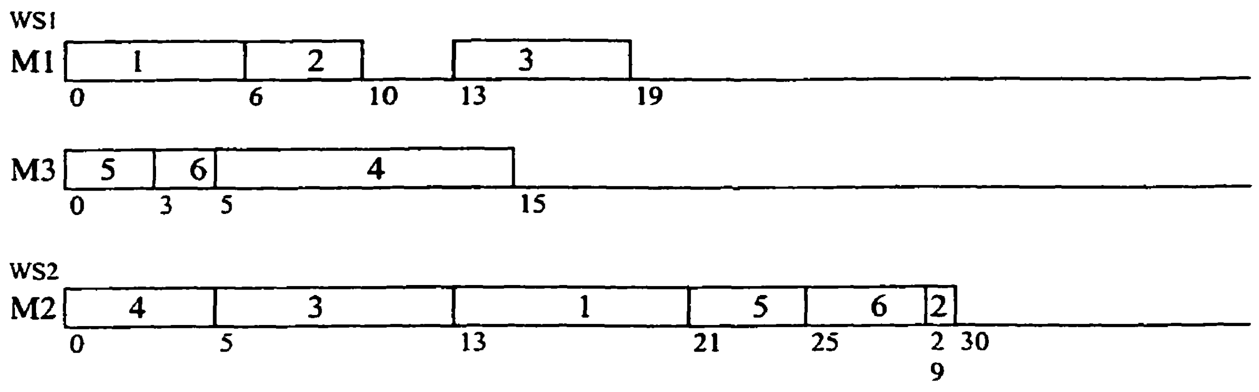


Figure 5.1: Gantt chart for the production system without any disturbance.

MODEL 1 – Advanced Scheduling

In this model, the control agents will first co-operate to generate a pre-production schedule. Then the control agents will execute the schedule in a strict order. That is, workstation agents will process the jobs in the exact order as described in their scheduled task list, and job agents will visit the workstations in the exact order as described in their scheduled operation list. Figure 5.2 shows the Gantt Chart that describes the actual production times for the jobs in this model. As shown in the figure, we can see that when job J1 is late in arrival by 16 minutes, workstation WS1 will have machine M1 wait for the job and start the operation at time = 16. The delay of job J1's arrival also affects its next operation's start time in workstation WS2. As a result of job J1's arrival, all jobs in workstations WS1 and WS2 that are scheduled behind job J1 are affected.

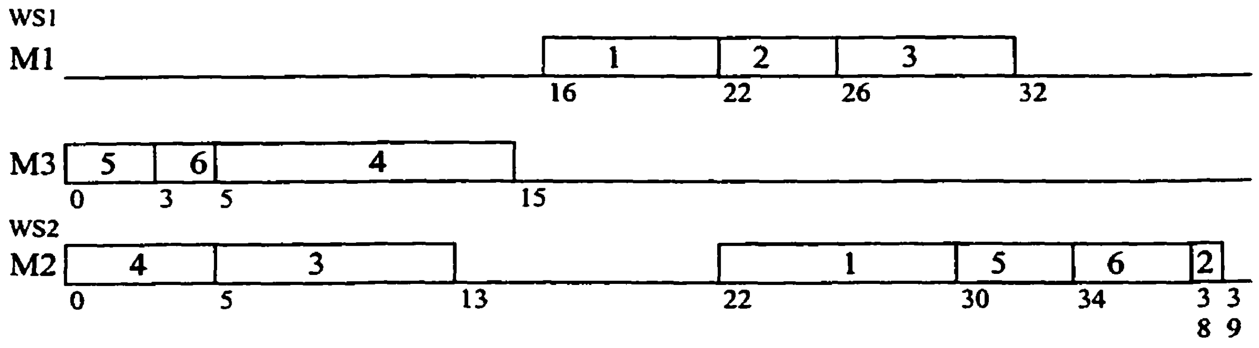


Figure 5.2: Gantt chart for model 1.

MODEL 2 – Scheduling One-step Ahead

In this model, instead of scheduling all the operations of the jobs at one time, a job's operation will be scheduled one step ahead. That is, the planning horizon for the jobs' operations is reduced. When a machine starts the operation of a job, the job agent will contact the workstation agent that corresponds to its next operation to reserve a resource. For example, referring to Figure 5.1, when machine M1 starts processing job J1 at time = 0, the job agent will contact workstation WS2 agent to add it to its queue (J1 agent tells WS2 agent that it will arrive at time = 6). Based on the arrival time that job J1 agent told it, WS2 agent will insert job J1 to its queue accordingly. And when the scheduled processing time is reached and the scheduled job hasn't arrived yet, the corresponding workstation will wait for the job.

Figure 5.3 shows the Gantt Chart that describes the actual production times for the jobs in this model. As shown in the figure, we can see that even though job J1 is late in arrival for 16 minutes, before time = 16, job J1 didn't make any reservation for the resource in workstation WS2. This keeps workstation WS2 agent from committing any resource (machine M2's time slot 13~21 (referring to figure 5.1)) to job J1. As resulted, we can see that with shorter planning horizon, the control system's flexibility against disturbance can be enhanced, and the impact of the disturbance on the system performance is minimized.

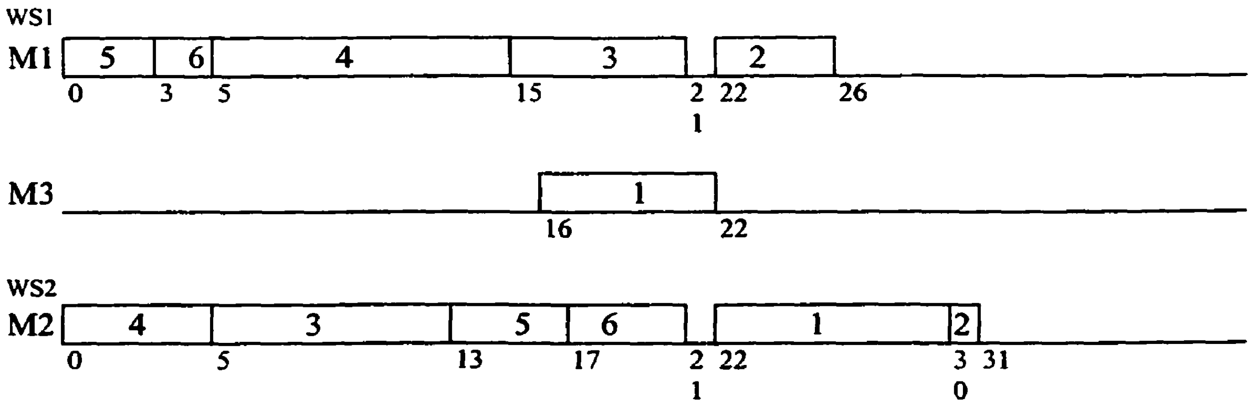


Figure 5.3: Gantt chart for model 2.

MODEL 3 – Real-time Distributed Scheduling

In this model, the production decisions are made in real-time, and the control agents do not commit to any pre-production plan /schedule. When a job arrives, the workstation agent will rank the job by some adopted priority rules. Whenever there is a machine available and there is a job in queue, the workstation agent will retrieve the first job in queue and allocate it to be processed in the first available machine. Figure 5.4 shows the Gantt Chart that describes the actual production times for the jobs in this model. As shown in the figure, we can see that the disturbance caused by job J1's late arrival is transparent to both workstations WS1 and WS2. In model 2, although workstation WS2 is not affected by job J1's late arrival, workstation WS1 is. This is because when the production system started, job J1 agent assumed that J1 would be available at time = 0, and thus contacted workstation WS1 agent to reserve a machine for its operation. Referring to Figure 5.3, WS1 agent then allocated one of its machines, M3 to job J1. This caused it to hold machine M3 from processing other jobs until it had processed job J1. As a result, we can see that in control systems that implement the real-time distributed scheduling approach, the control agents make no commitment to any future plan, and thus can enhance the control systems' adaptability against disturbances, in comparison to control systems that generate pre-production schedules.

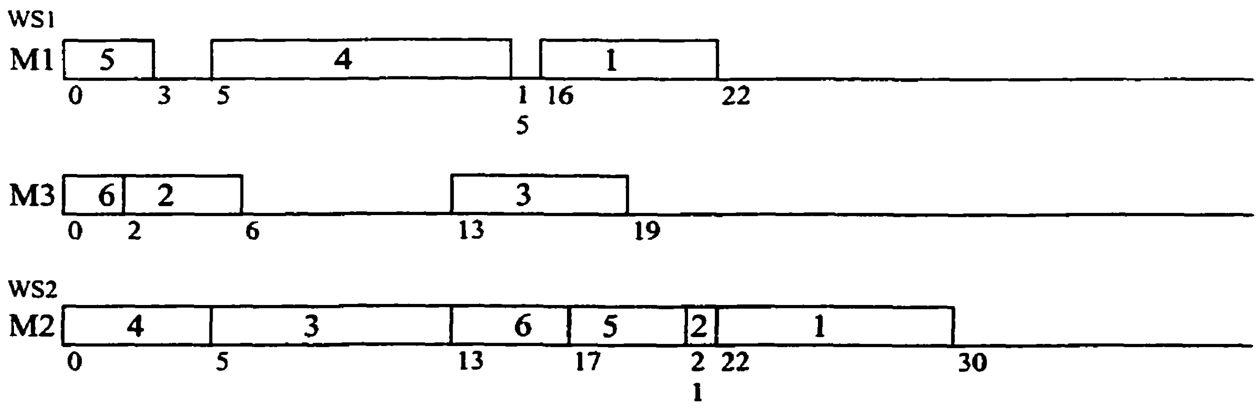


Figure 5.4: Gantt chart for model 3.

Referring to model 1 (Figure 5.2), we can see that in control systems that implement the distributed scheduling approach to generate the pre-production plans, the schedule still imposes a level of rigidity to the control system (Baker 1997). Therefore, in order to enhance the control system's flexibility against disturbances, it is important that the control agents have some local reactive mechanisms to respond to disturbances in real-time. Current research in distributed scheduling and control is mainly focused on scheduling issues, and few researchers have addressed these control issues. In the following sections, we will develop some experimental models to test and evaluate the performance of a distributed scheduling and control system under various control strategies in a stochastic manufacturing environment.

5.2 The Performance of the Distributed Scheduling and Control System under Various Control Strategies

In this section, we will first introduce the characteristics of the production systems that will be used in our experimental models. Then various experiments will be conducted to test the performance of the distributed scheduling and control system's performance under various control strategies in the stochastic manufacturing environment, and the results of these experiments will be discussed.

5.2.1 Production Model

The production system used in our experiments will be similar to the generic machining system proposed in (Cavalieri et al. 1999) for the benchmark job shop scheduling problem. The characteristics of the production system are summarized as follow:

- The production model
  - The layout of the production system is shown in Figure 5.5. Each type of machine can execute a single type of operation. 2 machines per type are present.
  - Transportation times, set-up times are ignored.

- Process plan
  - Products have a fixed process plan and are constituted by 4 non-preemptable operations, one for each type of machine.
  - The third machine to be visited by the products is a bottleneck resource.

- The manufacturing scenarios
  - Stochastic variability on the machining times.

- Measure of performance
  - The minimization of mean flow times.

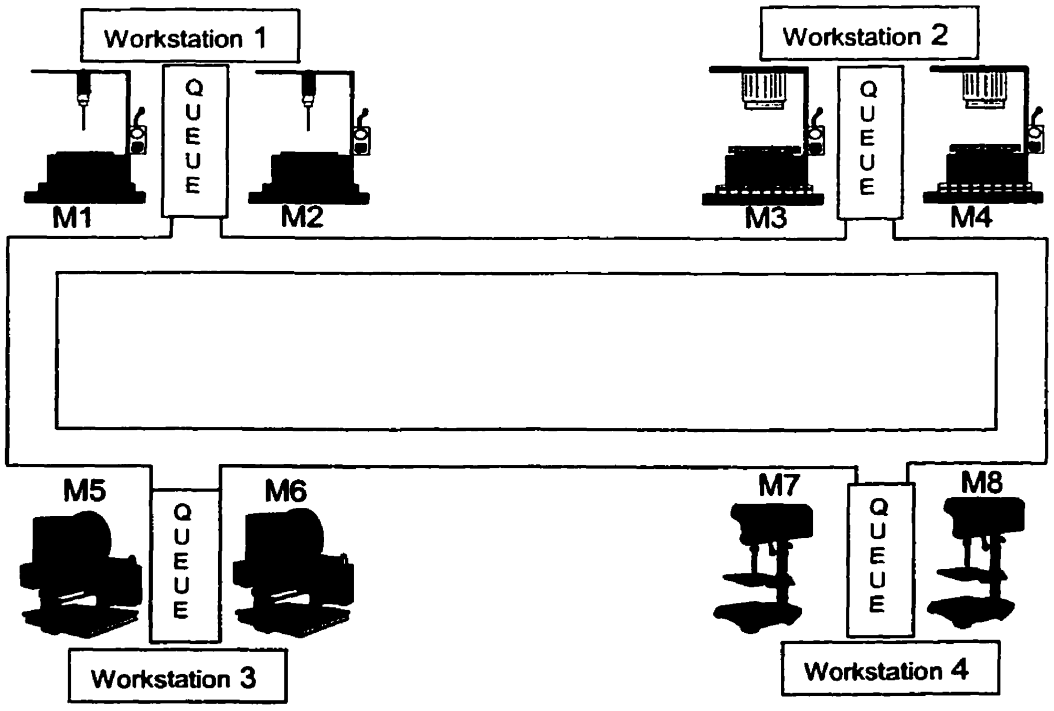


Figure 5.5: The layout of the manufacturing system.

5.2.2 Experiments and Results

In this section, we will investigate how the control agents' reactions to the uncertain machining time disturbance will affect the overall system performance. We choose minimizing mean flow time as the measure of system performance since in (French 1990), it has been proved that minimizing the mean flow time of jobs also minimizes their mean completion time, mean waiting time and mean lateness. Also, this is the performance measure that is proposed in (Cavalieri et al. 1999). The stochastic manufacturing scenario in the experiments is modeled by having some machines always delayed in finishing the jobs with a delay time generated by the triangular distribution function. We will use the experimental testbed described in §4.5 to carry out the experiments.

5.2.2.1 Control Strategies

In the experiments, the control agents (referring to Figure 4.8) will first cooperate to generate a pre-production schedule. After the production schedule is generated, the workstation agents will process the jobs in accordance with their scheduled task list. But when a machine delays in finishing a job, this will affect some of the workstation agents' production plan. For example, referring to Figure 5.6, if machine M2 delays in finishing job J3 at time = 13, workstation WS1 cannot have machine M2 start processing job J2, and workstation WS2 cannot have machine M3 start processing job J3 at time = 13 as scheduled.

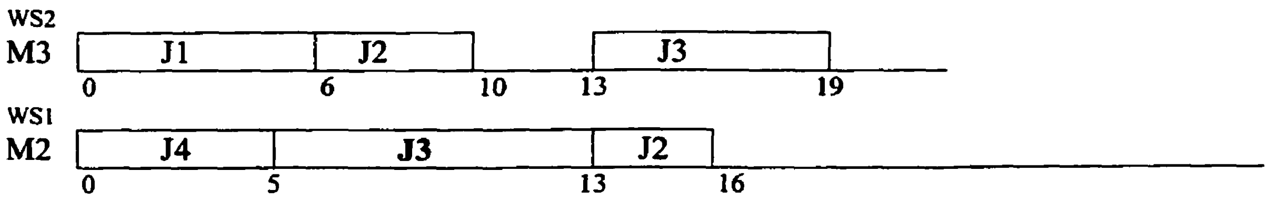


Figure 5.6: An example Gantt chart.

When this disturbance happens, workstations WS1 and WS2 agents can react in one of the following 3 ways:

1) WS2 agent will do nothing and wait for J3 to arrive, then load J3 to machine M3 as scheduled. WS1 agent will do nothing and wait for J3 to be finished, then have machine M2 processing job J2 after J3 is done, as scheduled.

2) WS1 or WS2 agent can contact the system mediator to request a rescheduling immediately.

3) The workstation agents will decide whether to wait or call for rescheduling immediately based on some criteria (to be explained later).

Before we get to option 3, we need to address the rescheduling problem first. To do the scheduling/rescheduling, we need to know when a machine will be free and when a job's operation will be finished, so that we can schedule a next job to a machine and schedule the job's next operation, respectively. But in the case of uncertain machine delay times, we do not know when the machine will finish the job. To overcome this problem, we will let the delayed machine's corresponding control agent estimate the time that it will finish its current job while doing rescheduling. But if a machine is prone to delay in finishing jobs and the delay time is uncertain and varies from job to job, then we need the machine's control agent to make a good guess about its delay time while doing rescheduling. This is because under or over estimation about the delay time can cause some extra rescheduling or machine idle times, respectively. Therefore in our experiments, during the production processes, each machine agent will learn from its delay records and calculate the mean delay time and use it for estimating the current job's finish time while doing rescheduling.

Regarding option 3 and referring to Figure 5.7, lets assume that machine M1 delays in finishing job J3 at time = 13. In our experimental models, after the scheduling process is complete, each workstation agent will compute the latest possible start time LPST for all of its jobs. The computation steps for the LPST are as follows (referring to

WS1 and jobs J2 and J3, whose first operation OP1 is done in WS1, and second operation OP2 is done in WS2):

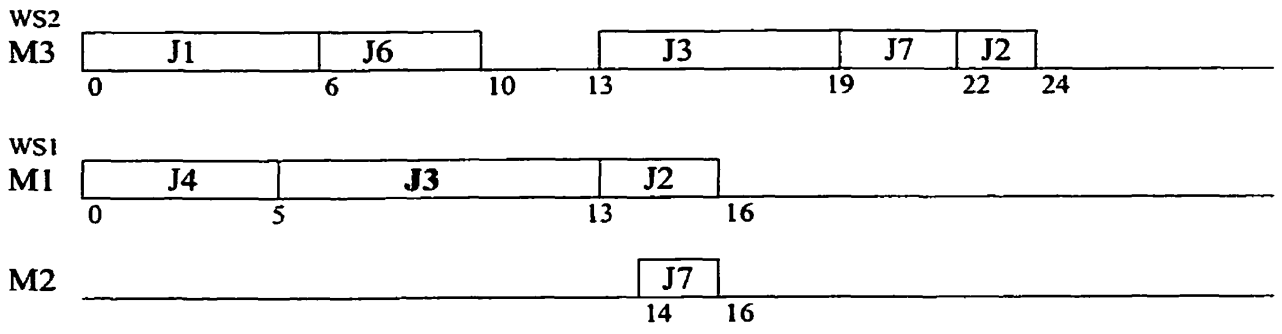


Figure 5.7: An example Gantt chart.

1) WS1 agent will consult each of its jobs' control agents about the latest start time LST for its operation. The latest start time LST of a job's current operation is equal to: LST = start time of next operation – processing time of current operation. For example, J3's OP1's processing time is 8 minutes, and the start time for its next operation OP2 is at time = 13. So its latest start time LST for OP1 is (13-8) at time = 5. And for J2, its OP1's processing time is 3 minutes, and the start time for its next operation OP2 is at time = 22, so its latest start time LST for OP1 is (22-3) at time = 19. This means that WS1 can postpone the processing of J2 until time = 19 and still will not interrupt the start time of J2's next operation. (If a job's current operation is its last operation, then its LST is equal to: due-date – processing time of current operation. For example, for J3 in WS2, say J3's due-date is at time = 30, then its LST for OP2 is equal to (30- 6 = 24.)

2) WS1 agent will then figure out the times that it can delay processing a job without affecting its next job's operation. We will call this delay time as LST'. The LST' for a job is equal to: LST' = next job's LST – current job's current processing time. For example, the LST' for J3 in WS1 is equal to the LST of job J2 (19) – the processing

time (8) of J3's operation OP1. Therefore, the LST' for J3 is equal to (19-8)= 11. And since J2 is the last job in machine M1, it doesn't have a LST' (LST' = ∞).

3) The latest possible start time LPST for a job is equal to: LPST = MIN(LST, LST'). For job J3, its LPST in WS1 is equal to MIN(5, 11) = 5. And the LPST for J2 is equal to MIN(19, ∞) = 19.

The latest possible start time LPST of a job represents the latest time that a workstation can delay processing a job and without causing an impact on the remaining operations of the job and the jobs that are scheduled behind that job.

For workstation WS2, at time = 13, when job J3 is late in arrival, WS2 agent will contact J3 agent to see when it will arrive. J3 agent will in turn contact WS1 agent to see when its machine M1 will finish J3's operation OP1. WS1 agent will ask its corresponding machine agent M1 to estimate its delay time DT and the job's finish time FT. Machine agent M1 estimates its delay time DT by calculating the mean from its past delay time records, and job J3's finish time is generated by adding job J3's start time + J3's processing time + estimated delay time (DT) = job J3's estimated finish time (FT). After receiving the response from M1 agent, WS1 agent will then forward the answer (FT) to job J3 agent. J3 agent will in turn pass that answer to WS2 agent. Upon receiving the answer, WS2 agent will then see if the FT is less than or equal to J3's LPST. If it is, then WS2 agent will decide to wait for J3. And if it's not, then WS2 will contact the system mediator to call for the rescheduling immediately. If WS2 agent decides to wait and J3's latest possible start time LPST is reached and J3 still hasn't arrived (M1 agent underestimated its delay time), WS2 agent will contact the system mediator to call for rescheduling. The collaboration diagram of the afore-mentioned processes is shown in Figure 5.8 below.

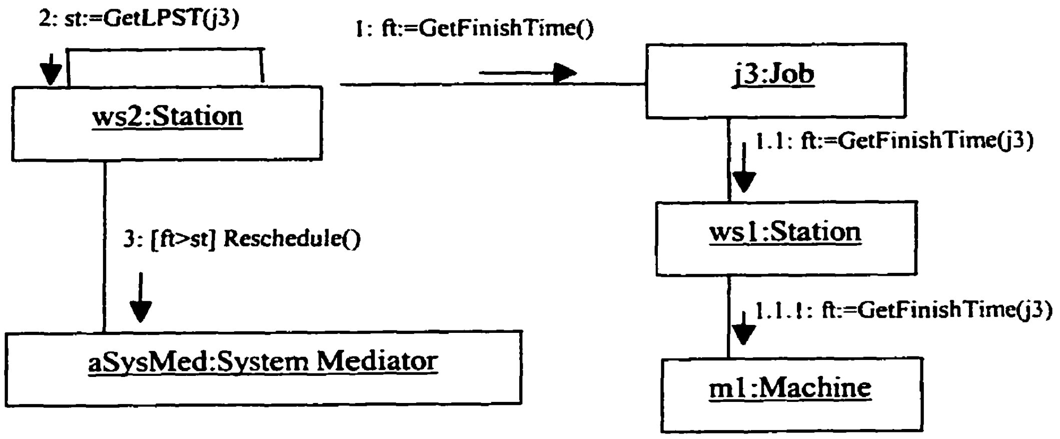


Figure 5.8: The collaboration diagram regarding WS2 agent's decision to call for rescheduling.

For workstation WS1, at time = 13, WS1 agent finds out that it cannot load job J2 to machine M1 because M1 is not done with job J3 yet. WS1 agent will then take the following actions:

1) See if there's another machine available at that time. For example, if machine M2 has no job scheduled to it after time = 13, then WS1 agent will load job J2 to machine M2 instead.

2) If as shown in Figure 5.7, that machine M2 has a job (J7) scheduled to it after time = 13, then WS1 agent will determine job J7's LPST. If the current time + J2's OP1's processing time is less than or equal to job J7's LPST, then WS1 agent will decide to load job J2 to machine M2 instead.

3) If the above 2 options are not feasible, then WS1 agent will have its corresponding machine agent M1 estimate its delay time DT and job J3's finish time FT. WS1 agent will then see if FT is less than or equal to job J2's LPST. If it is, WS1 agent will decide to wait for job J3's completion and load job J2 to machine M1 as scheduled. If it is not, then WS1 agent will call for rescheduling immediately. When J2's latest

possible start time LPST is reached and if machine M1 still hasn't finished job J3 yet (M1 agent has under estimated the its delay time), WS1 agent will then call for the rescheduling. The collaboration diagram of the afore-mentioned processes is shown in Figure 5.9 below.

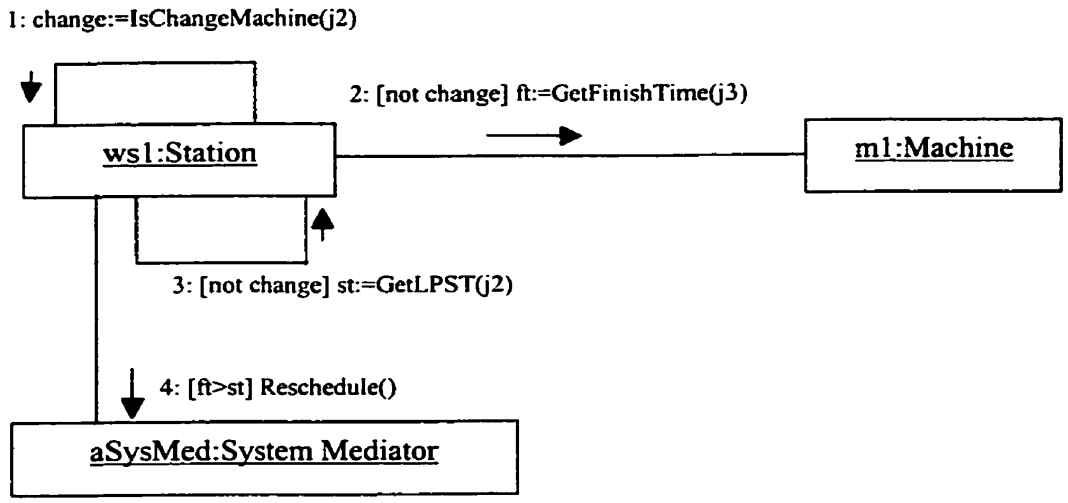

1: change:=IsChangeMachine(j2)
2: [not change] ft:=GetFinishTime(j3)
ws1:Station
m1:Machine
3: [not change] st:=GetLPST(j2)
4: [ft>st] Reschedule()
aSysMed:System Mediator

Figure 5.9: The collaboration diagram regarding WS1 agent's decision to call for rescheduling.

Before we proceed to the next section, we will summarize the control strategies that will be used in the experiments as follows:

NO_WAIT – represents the control system wherein the affected workstation agents will call for a rescheduling immediately whenever a machine delay disturbance happens.

WAIT – represents the control system wherein the affected workstation agents will just wait for the disturbance to pass and never call for rescheduling whenever a machine delay disturbance happens.

WAIT_INTEL – represents the control system wherein when a machine delay disturbance happens, the affected workstation agents will decide whether to wait or call for a rescheduling immediately, based on the criteria described in option 3 above (i.e., based on the collaborations described in Figures 5.8 and 5.9).

DIST – represents the control system wherein production decisions are made in real time. The scheduling algorithm is implemented for the real-time distributed scheduling purpose.

Although the above four control approaches use the same scheduling algorithm, unlike the WAIT, NO_WAIT and WAIT_INTEL control approaches, the DIST approach has no pre-production schedule generated. With the DIST approach, jobs are sent to the corresponding station queue for their first operation, then the scheduling algorithm is used to determine the processing order in real time.

5.2.2.2 Experiments

In this section, experiments will be conducted to test the performance of the aforementioned control strategies in various stochastic manufacturing scenarios. The stochastic manufacturing scenarios are modeled by providing machines in workstation WS1 with uncertain machining times. We will test and evaluate the performance of the alternate control methodologies in the manufacturing models with varying levels of uncertainty. The uncertainty level of the manufacturing model is constituted by 2 factors: 1) the disturbance frequency (i.e. the number of times that the machines in workstation WS1 will delay in finishing the jobs in production), and 2) the processing variability (i.e. the variation of the delay-time). 40 jobs of 3 job types will be produced in the manufacturing models. The process plan and the number of jobs for each job type are shown in Table 5.2 below.

Operation (process time (min) / operation type)					
Job ID	1	2	3	4	# of Jobs
J01	6/1	8/2	13/3	5/4	12
J02	3/4	6/2	15/1	4/3	14
J03	5/2	6/1	13/3	4/4	14

Table 5.2: Process plans and the number of jobs for the experimental models.

5.2.2.2.1 Experiment 5A: Performance of Different Control Strategies in Manufacturing Systems with Various Disturbance Frequencies

In this experiment, the machines M1 and M2 in workstation WS1 might delay in finishing the jobs with the delay time generated by a triangular distribution function TRIA (1,3,8). That is, the machine delay time varies from 1 minute to 8 minutes, with a mean delay time of 3 minutes. To model the various disturbance-frequency scenarios, the following test scenarios will be used:

1) The machines will always finish the jobs on time. That is, there is a 0% chance that the machines will delay in finishing the jobs.

2) During the production of the 40 jobs, there is a 60% chance that the machines will delay in finishing some of the jobs.

3) The machines will always (100%) delay in finishing the jobs.

The discrete probability function is used to generate the various disturbance frequencies for machines M1 and M2 in the Arena model (the simulated production system). The results of the tests are shown in Table 5.3 below. The results show the mean flow time (minute) of each control approach in different test scenarios. For the results of the WAIT_INTEL and NO_WAIT control methodologies, the number inside the parenthesis shows the 'total number of rescheduling' instances. Figure 5.10 shows the graphical interpretation of these results.

Disturbance Probability / Control Strategy	0%	60%	100%
WAIT	169	185	197
WAIT_INTEL (Reschedule #)	169	171 (4)	172 (10)
NO_WAIT (Reschedule #)	169	172 (31)	176 (54)
DIST	169	170	172

Table 5.3: The mean flow time (minutes) of various control approaches in different 'disturbance frequency' test scenarios.

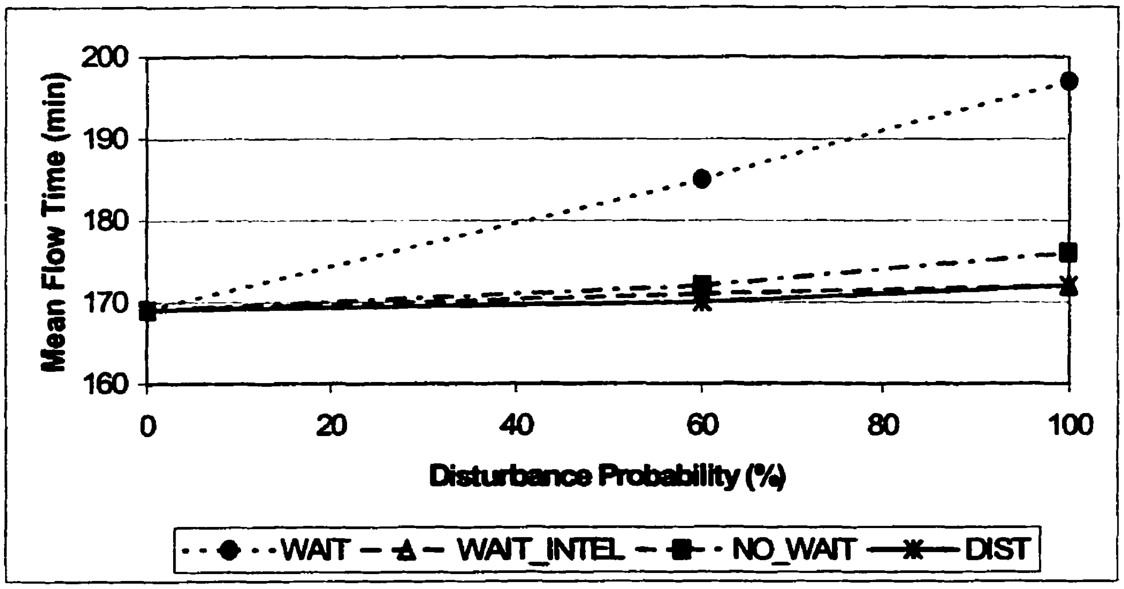
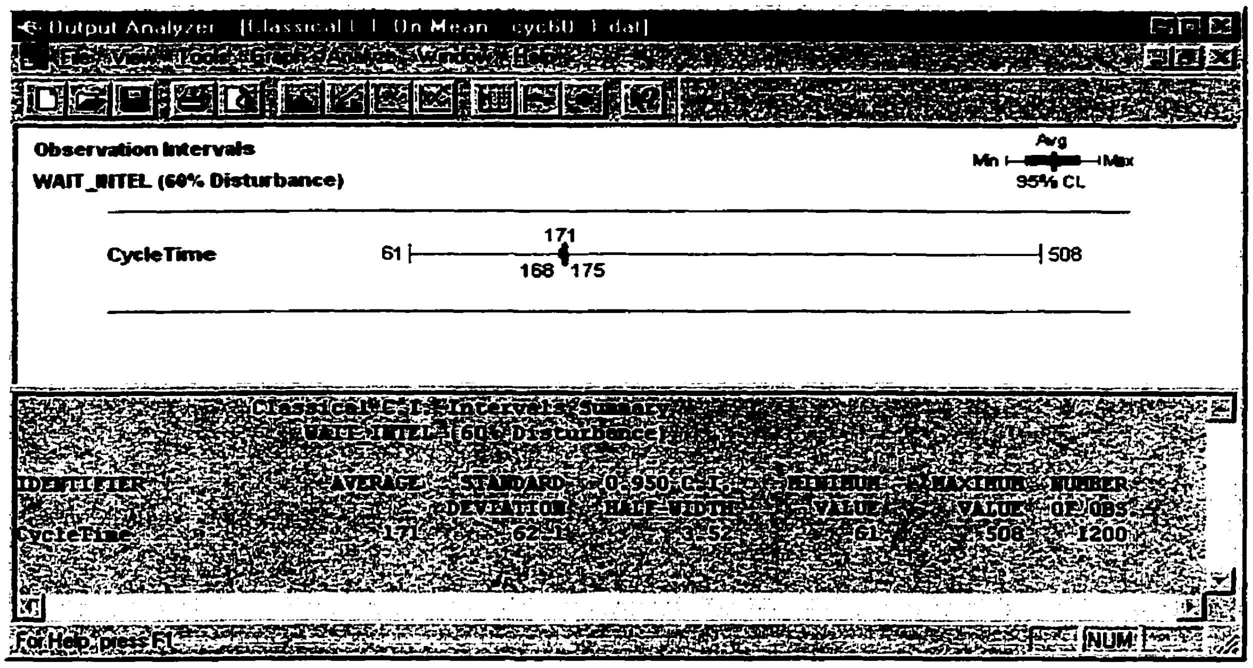


Figure 5.10: The mean flow time (minutes) of various control approaches in different 'disturbance frequency' test scenarios.

To ensure the consistency of each test result (that is, to make sure that a result does not represent an extreme instance caused by the randomness in the machine delay-time or disturbance frequency input data), each simulation is run for 30 replications. With a 95% confidence interval, the variation of each result (half-width) is about ±2% or less. For example, Figures 5.11 shows the 95% confidence interval results of the WAIT_INTEL control approach in the 60% disturbance test scenario.



```
                Classical C.I. Intervals Summary
                 WAIT_INTEL (60% Disturbance)

IDENTIFIER        AVERAGE   STANDARD   0.950 C.I.   MINIMUM   MAXIMUM   NUMBER
                            DEVIATION  HALF-WIDTH   VALUE     VALUE     OF OBS
CycleTime         171       62.1       3.52         61        508       1200
```

Figure 5.11: 95% confidence interval cycle time result for the WAIT_INTEL control approach in the 60% disturbance test scenario.

Referring to Table 5.3 and Figure 5.10, we can see that in control systems that generate the pre-production schedules (i.e., WAIT, WAIT_INTEL, NO_WAIT), when there is no disturbance, all the control methodologies have the same performance. But when there are disturbances, control systems (WAIT) that do not have the reactive mechanisms to response to disturbances in real time have inferior performance (in terms of mean flow time) than control systems (WAIT_INTEL and N0_WAIT) that can react to disturbances in real time. This is because when 'machining-time-delay' disturbances happen, with the NO_WAIT and WAIT_INTEL control approaches, available jobs in other workstations can be re-scheduled to be processed first instead of waiting for the delay job to arrive. But with the WAIT control approach, when a machine delays in finishing a job, the workstation that corresponds to the delay job's next operation might have to wait for the job, and this might delay the processing of the other scheduled jobs, which in turn might cause the processing of other jobs in other workstations to be delayed. As resulted, the accumulated delay times can greatly affect the performance of the control system. And the performance of the control system becomes worse as the disturbance frequency increases.

Regarding the WAIT_INTEL and NO_WAIT control approaches, we can see that the WAIT_INTEL control approach far outperforms the NO_WAIT approach in terms of 'rescheduling frequencies' in all the tests (as shown in Figure 5.12). Referring to Figures 5.10 and 5.12, we can see that in the manufacturing systems with disturbances, the rescheduling frequency of the NO_WAIT approach is significantly higher that that of the WAIT_INTEL approach (especially in the 100% disturbance case). This high rescheduling frequency also causes the NO_WAIT approach to have worse performance than the WAIT_INTEL approach. This is because as mentioned in §5.3.1, when doing the rescheduling, the machines with uncertain processing time have to estimate their delay times. While under-estimation of the delay times may cause extra rescheduling, over-estimation of the delay times may cause extra machine idle times, which will affect the overall system performance. Control systems with higher rescheduling frequencies are more likely to have this kind of estimation error, since higher rescheduling frequency means more opportunities to generate faulty schedules. Also, in control systems wherein the machines have large processing variability, it is more difficult to estimate the machine delay time while doing the rescheduling. As a result, (larger) estimation faults are usually incorporated in the new schedules, and thus will affect the system performance (we will further discuss this issue in the next experiment).

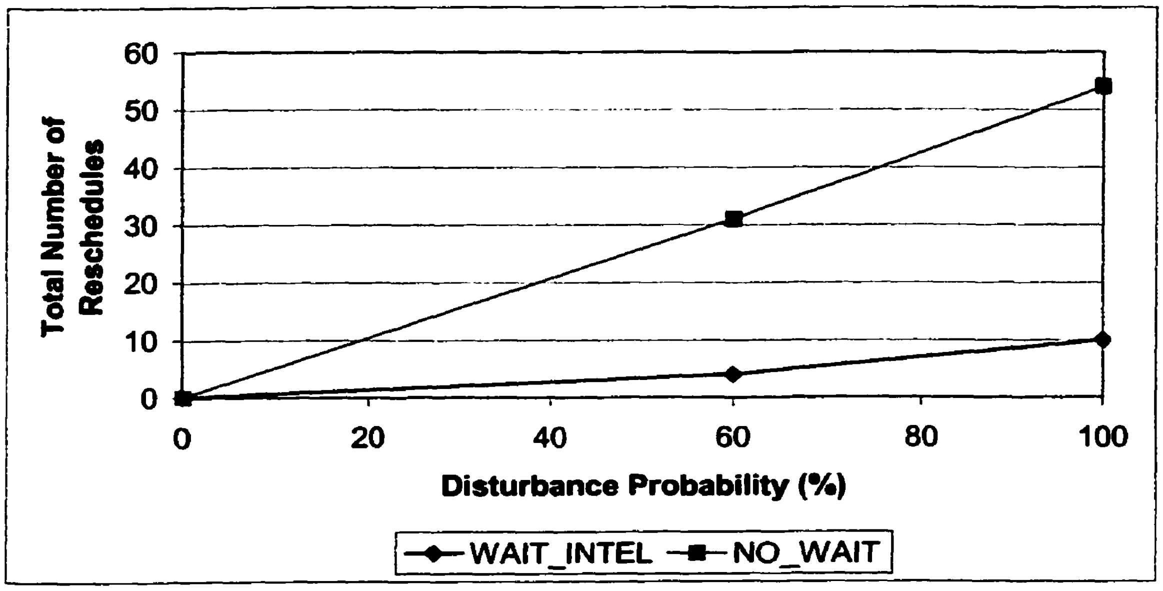


Figure 5.12: Results of the rescheduling frequencies of the WAIT_INTEL and NO_WAIT approaches in various test scenarios.

Referring to Figure 5.10, theoretically, the NO_WAIT and the DIST approaches should have similar (if not the same) performance, since both approaches use the same scheduling algorithm to allocate resources to the jobs. As well, in the NO_WAIT approach, rescheduling is done whenever processing-delay disturbances happen. Therefore, the results of both approaches represent the (non-delay) production schedules of the control system with the machining delay times incorporated in them. But in practice, with the NO_WAIT approach, rescheduling may incur some estimation faults in the new schedules, and the DIST approach can avoid this kind of error. This is because in DIST control, resources do not have to follow any pre-production schedules (production decisions are made in real-time), and thus machines do not have to make any delay estimates (for rescheduling purpose). As a result, the DIST approach always has better performance.

5.2.2.2.2 Experiment 5B - Performance of Different Control Strategies in Manufacturing Systems with Various Processing Variabilities

In each of the test scenarios in this experiment, the machines in workstation WS1 will have different processing variabilities, and the probability of the processing-delay disturbance occurrence is 100%. The uncertain delay-time for the machines in each test scenario is given as follows:

1) Zero processing variability – The machines will always finish the jobs on time (no processing delays).

2) TRIA(0,3,5) – The machines will always delay in finishing the jobs. The delay-time uncertainty is modeled by the triangular distributed function, which will generate a delay-time between 0 – 5 minutes, with a mean of 3 minutes.

3) TRIA(0,3,8) – The machines will always delay in finishing the jobs. The delay-time uncertainty is modeled by the triangular distributed function, which will generate a delay-time between 0 – 8 minutes, with a mean of 3 minutes.

4) TRIA(0,3,12) – The machines will always delay in finishing the jobs. The delay-time uncertainty is modeled by the triangular distributed function, which will generate a delay-time between 0 – 12 minutes, with a mean of 3 minutes.

Table 5.4 shows the results of the performance (the mean flow time) of the 4 control approaches mentioned in §5.2.2.1 in each test scenario. Figure 5.13 shows the graphical interpretation of the results in Table 5.4. Each simulation is run for 30 replications and the variation of each result is about ±2% or less.

Experiment # / Control Strategy	ZERO	TRIA(0,3,5)	TRIA(0,3,8)	TRIA(0,3,12)
WAIT	169	188	194	204
WAIT_INTEL (Reschedule #)	169	170 (4)	172 (8)	176 (14)
NO_WAIT (Reschedule #)	169	171 (52)	175 (54)	182 (56)
DIST	169	170	171	174

Table 5.4: The mean flow time (minutes) of various control approaches in different 'processing variability' test scenarios.

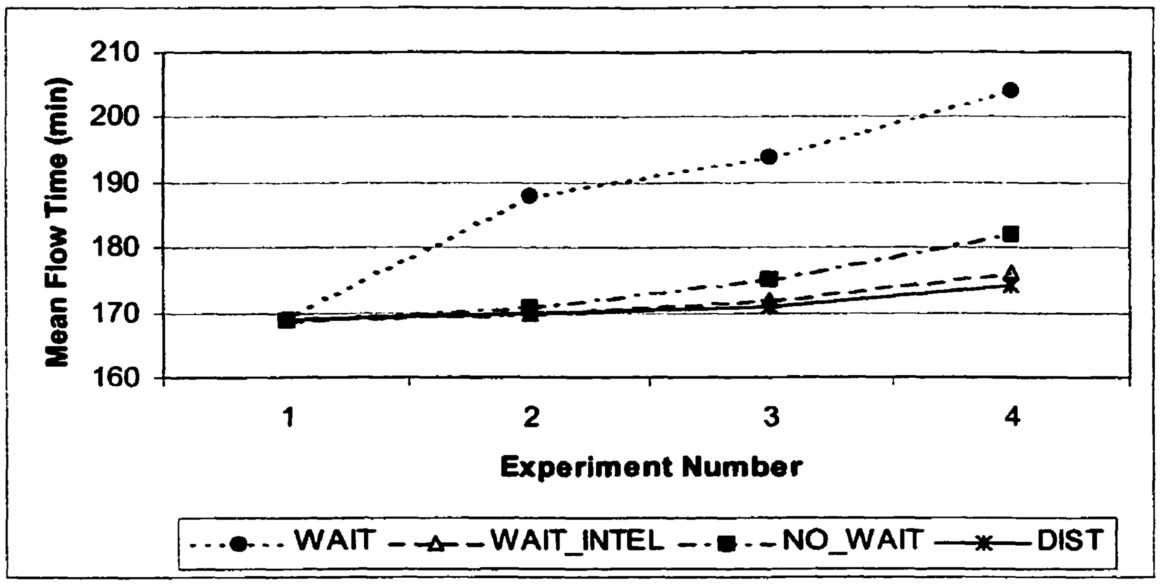


Figure 5.13: The mean flow time (minutes) of various control approaches in different 'processing variability' test scenarios.

Referring to Table 5.4 and Figure 5.13, the increase of the 'processing variability' of the machines in workstation WS1 imposes 2 kinds of impacts on the performance of the alternative control approaches:

1) Higher 'processing variability' implies that some of the machines' processing delay-times will be increased. As resulted, this will increase the processing time of some of the jobs, and thus increase the cycle time of the production. This impact is reflected in the results shown in Figure 5.13. In the figure, we can see that the mean flow time of all the control approaches increase as a result of the increase of the processing variability. When considering the control systems that generate pre-production schedules, when there are disturbances, control systems that do not have the reactive mechanisms to response to disturbances in real time (WAIT) have inferior mean flow time performance than control systems that can react to disturbances in real time (WAIT_INTEL and N0_WAIT).

2) With higher 'processing variability', it is more difficult for the machine agents to estimate their delay-times while doing the rescheduling. As mentioned in the last section, while under-estimating the delay-times may cause extra rescheduling, with higher 'processing variability', there is a higher probability that the machine agents will over-estimate their delay times. Higher 'processing variability' may also cause larger estimation errors. As resulted, this will cause extra (larger) machine idle times, and thus increase the cycle time of the production. This impact is reflected in the results shown in Figure 5.13. In the figure, we can see that with increasing 'processing variability', the performance of the NO_WAIT control approach deviates from the performance of the DIST approach with greater angles.

A paired-t zero rejection hypothesis test was performed to evaluate the differences between the performance of the WAIT_INTEL and DIST control approaches. Even though this test showed these differences to be statistically significant, the differences are very small as can be seen in Figure 5.3. This is because whenever the machining delay disturbances happen, instead of calling for rescheduling immediately, the workstation control agents in the WAIT_INTEL approach will decide to call for rescheduling or not based on certain rescheduling-invocation criteria. As resulted, the rescheduling frequency of the WAIT_INTEL approach is significantly lower than that of the NO_WAIT approach in all test scenarios. Higher rescheduling frequency means more opportunities for generating faulty schedules. And as mentioned above, higher 'processing variability' might cause larger machining delay-time estimation errors, and thus cause extra (larger) machines idle times. Therefore, with increasing 'processing variability', the performance of the NO_WAIT approach deviates from the performances of the WAIT_INTEL and the DIST approaches.

5.2.2.2.3 Experiment 5C: Unpredictability of Heterarchical Control Systems

In experiments 5A and 5B, the results of the control system (DIST) that implements the distributed scheduling algorithm for real-time control always has superior mean flow time performance over other control approaches. This is because with the DIST control approach, no pre-production plan is generated. As a result, the DIST control approach can be more flexible to react to disturbances in real-time, and can avoid generating faulty schedules (machining delay-time estimation errors generated during the rescheduling processes).

Even though the DIST control approach uses the same scheduling algorithm as the other control approaches to allocate jobs to resources, in some cases, the adopted dispatching rule that is used in the scheduling algorithm might be violated in the DIST control systems. As resulted, the performance of the control system becomes unpredictable. For instance, in the DIST control systems, since there is no pre-production plan, the workstation agents will process the arriving jobs in the order that is based on some adopted dispatching priority rules. But in situations wherein there is an idle machine in the workstation, and two jobs (say job A and job B) with different priority arrive at the workstation at the same time but in different sequence. If job A has lower priority than job B but arrives at the workstation first, the workstation agent will load job A to the idle machine immediately. Then when job B arrives, it has to wait in the queue, even though its priority is higher than job A's. But in control systems with pre-production schedules, since both jobs will be available at the workstation at the same time, the job with the higher priority will be scheduled to be processed first. So in such control systems, while executing the production schedule, the workstation control agent will always process job B (higher priority, as scheduled) first, regardless of the arriving sequence of jobs A and B.

To model this situation, we use the production data shown in Table 5.5 below to conduct the tests.

	Operation (process time (min) / operation type)				
Job ID	1	2	3	4	# of Jobs
J1	6/1	8/2	13/3	5/4	4
J2	4/1	3/2	8/3	3/4	4
J3	3/4	6/2	15/1	4/3	4
J4	5/2	6/1	13/3	4/4	4
J5	5/1	3/2	8/4	4/3	4

Table 5.5: Process plans and the number of jobs for the experiment 5C.

In this experiment, we will produce 20 jobs, 4 of each job type listed in Table 5.4. In the table, we can see that jobs with job types J1, J2, and J5 all need to have their first operation processed in workstation WS1. In our scheduling algorithm (for the WAIT, NO_WAIT, and WAIT_INTEL approaches), the Most-Work-Remaining (MWKR) priority rule is used for ranking the jobs. Therefore, in the production schedule, at time = 0, since all jobs are available at that time, jobs with job type J1 will be scheduled to be processed first in workstation WS1, since they have higher priority (for operation 1) than jobs with other job types. But in the DIST control system, at time = 0, it is possible for jobs with other job types to arrive at the workstation WS1 first. And in such cases, workstation WS1 agent will load the first arriving job to the first available machine. As a result, some jobs with lower priority will be processed first.

The stochastic manufacturing scenario in this experiment is modeled by having machines in workstation WS1 have an uncertain processing delay-time generated by the triangular distribution function TRIA(0,2,4). In different test scenarios, we will have jobs with different job types arrive at workstation WS1 in different sequences at the start of the production (time = 0). And jobs with job types J3 and J4 will go to workstations WS4 and WS2, respectively, at time = 0 for the processing of their first operation. The arrival sequences (for workstation WS1) of the jobs in different test scenarios are:

1) (J1, J2, J5) – In the first test scenario, jobs with job type J1 will arrive (at workstation WS1 at time = 0) first, then the jobs with job types J2 and J5 will arrive.

2) (J5, J2, J1) - In the second test scenario, jobs with job type J5 will arrive (at workstation WS1 at time = 0) first, then the jobs with job types J2 and J1 will arrive.

3) (J2, J1, J5) - In the third test scenario, jobs with job type J2 will arrive (at workstation WS1 at time = 0) first, then the jobs with job types J1 and J5 will arrive.

The results of the tests are shown in Table 5.6 below. In test scenario 2 (J5, J2, J1), when the production system starts, jobs with job type J5 were modeled to arrive at workstation WS1 first. In the DIST control system, the workstation WS1 agent will load the first 2 arriving J5-type jobs to its 2 available machines (the remaining J5-type jobs will then be ranked with other late arriving jobs). As resulted, two J5-type jobs that have lower priority than the J1-type jobs will be processed first. Similar situations happened in test scenario 3 (J2, J1, J5). The violation of the dispatching priority rule in test scenarios 2 and 3 had affected the overall system performance of the DIST control system. 30 replications have been run for each test, and the paired-t zero rejection hypothesis test has showed that the differences between the DIST test result in different test cases are statistically significant.

In the WAIT, NO_WAIT and WAIT_INTEL control approaches, since the pre-production schedules are generated in these systems, the workstations will always process the jobs as scheduled, regardless of the arriving sequence of the jobs. As resulted, the performances of these control approaches do not change in all the test scenarios.

Referring to the above experiments 5A and 5B, and 5C, we can see that even though the control systems (DIST) that make the production decisions in real time can be more flexible against disturbances, sometimes the behaviors (or performance) of such systems are hard to predict.

Test Scenario / Control Strategy	(J1, J2, J5)	(J5, J2, J1)	(J2, J1, J5)
WAIT	86	NO CHANGE	NO CHANGE
WAIT_INTEL (Reschedule #)	76.7	NO CHANGE	NO CHANGE
NO_WAIT (Reschedule #)	77.8	NO CHANGE	NO CHANGE
DIST	76.4	82.9	83.8

Table 5.6: The mean flow time (minutes) of various control approaches in different test scenarios for experiment 5C.

5.3 Conclusion

Referring to the experiments described in §5.2, we can see that while implementing the scheduling algorithm for real-time distributed scheduling and control can enhance the control system's flexibility against disturbances, sometimes it is hard to predict the behavior and performance of such control systems. Therefore, sometimes it is still necessary to develop the pre-production plans (advance schedules) to enhance the predictability of the control systems.

Research in distributed scheduling and control systems has claimed that the distributed scheduling approach can enhance the control systems' adaptability against disturbances (Dilts et al. 1991, Duffie et al. 1994, Sousa et al. 1997). But (referring to the experiments described in §5.2) unless the distributed scheduling algorithm is implemented for real-time distributed control, control systems that implement the distributed scheduling algorithms to generate the pre-production schedule can still be very susceptible to the impact of the disturbances. In the distributed scheduling and control systems that generate pre-production schedules, implementing the distributed scheduling approach can achieve advantages such as:

1) The scheduling performance can be enhanced "through parallel computing and through the elimination of the processing bottleneck caused by global scheduler" (Dilts et al. 1991)

2) The control system's fault-tolerance can be improved (can avoid the single point of failure problem of the centralized scheduling system).

3) The control system's reconfigurability and adaptability can be enhanced (Dilts et al. 1991).

It is important to note though, that the control systems should have the proper control algorithms to react to disturbances in real-time. Otherwise, the performance of the control systems can be greatly affected (cf.., the WAIT control approach), especially in the stochastic manufacturing environment.

If the control agents in the distributed control systems do not have the local reactive control mechanisms to react to disturbances in real-time when disturbances happen, the affected control agents have to invoke the rescheduling processes so that the control system can respond to these disturbances (e.g., the NO_WAIT control approach). Some researchers have used the rescheduling approach to help enhance the control system's adaptability against disturbances. For instance, in (Duffie et al. 1994), the control agents in the control system continuously generate new 'look-ahead' schedules via simulations. For each simulation, the current status of the production system is modeled in the simulation model so that the control system can be "adaptable to faults" (Duffie et al. 1994). But while doing the rescheduling, all the control agents in the system have to be involved. Therefore, high frequency of rescheduling means that the control agents will always be engaged in some communication processes and this might violate some design principles of the distributed control systems. For instance, Duffie et al. (1994) mention that:

"The fully distributed heterarchical manufacturing system scheduling and control architecture is comprised of loosely coupled, highly autonomous entities retaining minimal global information".

But in (Smith 1980), the 'loosely coupled' is defined as:

"Loosely coupled means that individual KS's (Knowledge Source) spend most of their time in computation rather than communication".

Therefore, we can see that the 'loosely coupled, highly autonomous' features of the distributed scheduling and control systems require that the communication between the control agents should be minimized. To achieve this goal, it is important that the control agents should have the local reactive control mechanism (autonomy/intelligence) to react to disturbances in real-time, and thus can minimize the rescheduling frequencies.

Referring to the experiments in §5.2, we can see that in control systems (e.g., the WAIT_INTEL approach) wherein the control agents have the proper local reactive control mechanisms to react to disturbances in real time, not only can the desirable system performance can be achieved, but the control system's adaptability against disturbances can be enhanced, and the rescheduling frequencies can be minimized significantly, in comparison to the NO_WAIT approach. As well, when reacting to the machining delay disturbances, unlike the rescheduling approach (wherein all the control agents have to be involved), in the WAIT_INTEL approach, only the affected control agents will be contacted, and the unaffected control agents will not be bothered. As resulted, with the WAIT_INTEL control approach, the "loosely coupled, highly autonomous" features of the distributed scheduling and control systems can be fully realized.

CHAPTER 6

JOB SEQUENCING AND DISPATCHING ROUTING DECISIONS IN THE MULTI-AGENT HETERARCHICAL CONTROL SYSTEMS

Current research in multi-agent heterarchical control systems has commonly used the dispatching routing approach to allocate jobs to resources (Duffie et al. 1986, Baker 1997). In such control approach, resource allocations are accomplished by having the jobs collect bids from the potential machines that can process their operations. "The contents of these bid are usually simple information such as cost, earliest start time, or earliest finish time" (Baker 1997). The contract net auction-bidding protocol (Smith 1980) is commonly used in the multi-agent heterarchical control systems for the part agents to make the routing decisions.

> "Indeed, the preponderance of agent research for manufacturing has developed agent architectures which implement different dispatching rules. It is most common to dispatch the routing decision in these architectures, assuming sequencing can then be done at each resource.... In the case of the routing decision, a great deal of agent research has been with having the agents make this decision by collecting bids from potential machines to which the job can be routed" (Baker 1997).

Most work in multi-agent distributed scheduling and control systems only deals with dispatching the routing decisions, and the job sequencing issues are usually ignored. As resulted, most of these approaches usually perform the scheduling by jobs on a First-

Come-First-Serve basis (Veeramani & Wang 1998), which sometimes compromises certain global performance objectives.

In this chapter, we will investigate the impact of the job routing and job sequencing decisions on the control system's performance and its adaptability against disturbances such as machine failure. The experimental testbed described in §4.5 will be used for testing and evaluating the performance of various control algorithms in a multi-agent heterarchical shop floor scheduling and control system. The experimental results will then be discussed and a conclusion will be presented.

6.1 Experiments and Results

Current research in multi-agent heterarchical control systems usually implement 'part driven' real-time scheduling algorithms, wherein the part agents use the auction-bidding resource reservation protocol to explore the routing or process sequencing flexibility in real-time. Traditional dispatching control systems usually implement 'resource driven' scheduling (dispatching) algorithms, wherein the resource controllers (agents) use dispatching rules to sequence the processing of the arriving jobs, and the routing decisions are usually determined in advance.

> "Most dispatching rule research has been with dispatching which job a resource will work on next. This sequencing decision can be made based on a job's due-date, its customer priority, similar setups, the shortest processing time remaining.... Once a job is released into the factory, or once a job is finished at a resource, the next decision is which resource to route it to next. Often, the routing decision has been made in advance" (Baker 1997).

Although quantitative results are available for the traditional dispatching and the bidding-based control approaches, few researchers have compared the performance of these alternative approaches on a common platform. In (Duffie et al. 1994), wherein the 'part driven' real-time distributed scheduling and control algorithms are implemented, it is proposed that "future work should compare them with traditional dispatching rules and scheduling heuristics" (Duffie et al. 1994).

In this section, we will conduct experiments to investigate the impact of the dynamic job routing and job sequencing decisions on the control system's performance and adaptability against disturbances. The tested control systems will have varying production volumes (to model the production system with looser/tighter schedules) and disturbance frequencies, so that the impact of the job routing and sequencing decisions in various manufacturing environments can be evaluated. In our experimental models, routing flexibility is introduced into the production system by providing jobs with a flexible processing order for their operations. That is, there is no technological constraint on the processing sequence of the operations of the jobs.

6.1.1 Experimental Models

To evaluate the impact of dynamic job routing and job sequencing decisions in various manufacturing environments, the following control strategies will be implemented in our experimental models:

a) AUC_BID (AUCtion-BIDding) - In this control approach, the job control agents will use the contract net auction-bidding protocol to collect bids from the workstations to explore the process sequencing/routing flexibility. Job sequencing will not be implemented in this control approach. That is, to decide which operation to process next, for each of the job's remaining unprocessed operations, the job agent will contact the system mediator to find out which workstation is responsible for that type of operation. Then the job agent will contact the corresponding workstation agent to see when the workstation can start the operation. Since job sequencing is not implemented, the workstation agent will rank the incoming jobs on the First-Come-First-Serve basis, and respond to the job agent with the answer that states the earliest possible start time for that operation. After receiving responses from all the workstations that can process its remaining unprocessed operations, the job agent will evaluate all the responses and pick the operation whose corresponding workstation can start the job soonest to be processed next.

b) JSEQ (Job SEQuencing) – In this control approach, the workstation control agents use the adopted priority dispatching rule to sequence the incoming jobs, and the jobs do not explore the routing flexibility. That is, even though there is no technological constraint for the operations of the jobs, the job agents will not explore the routing flexibility, and will have their operations processed in some predetermined order (the order that is originally stated in their process plan). When a job enters a workstation, the workstation agent will rank the incoming jobs based on some adopted priority rules. In our experiments, the Least Work Remaining (LWKR) heuristic priority dispatching rule will be used. This is because the performance measure of our experiments is the minimization of the mean flow time, and the empirical experimental results conducted by other researchers (Conway et al. 1967) have suggested that the LWKR rule can help minimize the mean flow time.

c) AUC + JSEQ – In this control approach, while the job agents will use the auction bidding mechanism as stated in (a) to explore the routing flexibility, the workstation agents will sequence the incoming jobs based on the dispatching rules as stated in (b). That is, to decide which operation to process next, the job agents will collect bids from the workstations that correspond to its remaining unprocessed operations. Unlike in (a), when a workstation agent receives a bid request from a job agent, instead of quoting the job's earliest possible start time based on the First-Come-First-Serve rule, the workstation agent will try to insert the job into its queue and quote the job with the earliest possible start time that is based on the adopted priority dispatching rule as stated in (b). After receiving the response from all the workstations that correspond to its remaining unprocessed operations, the job agent will evaluate all the responses and pick the operation whose corresponding workstation can start the job soonest to be processed next.

d) COMT+AUC+JSEQ (COMmitmenT + AUC + JSEQ) – One of the problems regarding the control approach stated in (c) is the role of commitment in the auction-bidding processes. In deciding which operation to be processed next, the job agent will make the decision based on the returned 'earliest start time' quote of the workstations that correspond to its remaining unprocessed operations. The returned quoted start time represents that the workstation is willing to commit some of its resource capacities to process the job at certain times. But when the workstation agents use the LWKR rule to sequence the incoming jobs, the workstation agents might violate some of the previous commitments that it has made to some jobs.

For example, referring to Figure 6.1 below, at time = 3, if workstation WS1 agent has responded to the job J2 agent that it can start processing the job at time = 6, and the job J2 agent, after evaluating some other bids, decided to join the workstation WS1. Then at time = 4, a new job J5 that has smaller remaining work processing times than job J2, asks WS1 when it can start processing the job. If workstation WS1 uses the LWKR rule to try to insert job J5 into its queue, it will answer the job J5 that the earliest possible start time that it can process the job is at time = 6. And if job J5 decides to join workstation WS1, then workstation WS1 will violate the quoted start time it sent to the job J2 previously. In this case, should the affected job J2 be notified that its quoted start time has been changed, so that job J2 can explore other routing opportunities to see if other workstations that correspond to its other remaining unprocessed operations can start the job earlier? And how would this opportunistic behavior of the job agents affect the performance and the communication requirements of the control system? The COMT+AUC+JSEQ control approach will be used to investigate these issues. In this control approach, when the workstation agents insert a new job into its queue, the affected jobs will be notified so that they can explore other routing opportunities. For the affected job agents, if no other workstations can start their other remaining operations sooner, then they will decide to stay in the original workstation. Otherwise, they will change the workstation (and the process sequence).

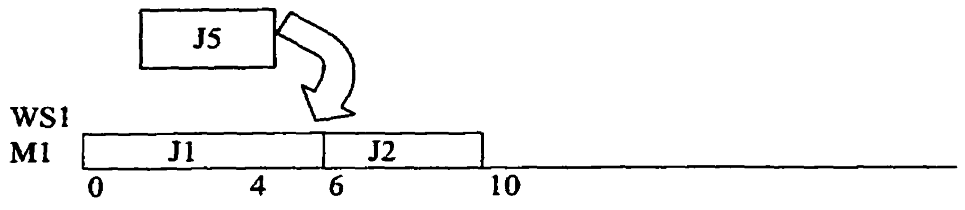


Figure 6.1: An example task list for workstation WS1.

The process plan for the job types that will be used in the experimental models is shown in Table 6.1 below.

Operation (process time (min) / operation type)				
Job ID	1	2	3	4
J1	6/1	8/2	13/3	5/4
J2	4/1	3/2	8/3	¾
J3	3/4	6/2	15/1	4/3
J4	5/2	6/1	13/3	4/4
J5	5/1	3/2	8/4	4/3

Table 6.1: Process plan of the various job types.

6.1.2 Experiment 6A – Zero Disturbances

In this experiment, we will evaluate the performance of the four control strategies described in the §6.1.1 above in control systems with zero disturbances. Each of the four control strategies will be implemented in control systems with varying production volumes, so that the impact of the alternative control approaches in control systems with various degree of tightness of schedules can be evaluated. In each test, equal amounts of each of the job types described in Table 6.1 above will be produced. The results of the tests are shown in Table 6.2 below. Figure 6.2 shows the graphical interpretations of the results shown in Table 6.2.

Number of Jobs / Control Strategy	10 Jobs	15 Jobs	20 Jobs	25 Jobs	30 Jobs	35 Jobs	40 Jobs	50 Jobs	70 Jobs
AUC_BID	33.6	49	62.5	80.5	92.7	113.6	129	159	211
JSEQ	36.8	46.3	57.8	69.4	81.7	94.5	107.4	133.7	180
COMT + AUC + JSEQ	33.4	42	47.5	55.6	64	72.5	81.3	97.6	133.2
AUC + JSEQ	33.8	42	50.4	57.6	69.5	78.2	82.8	103.8	141.4

Table 6.2: The mean flow time (minutes) of various control approaches in different test scenarios for experiment 6A.

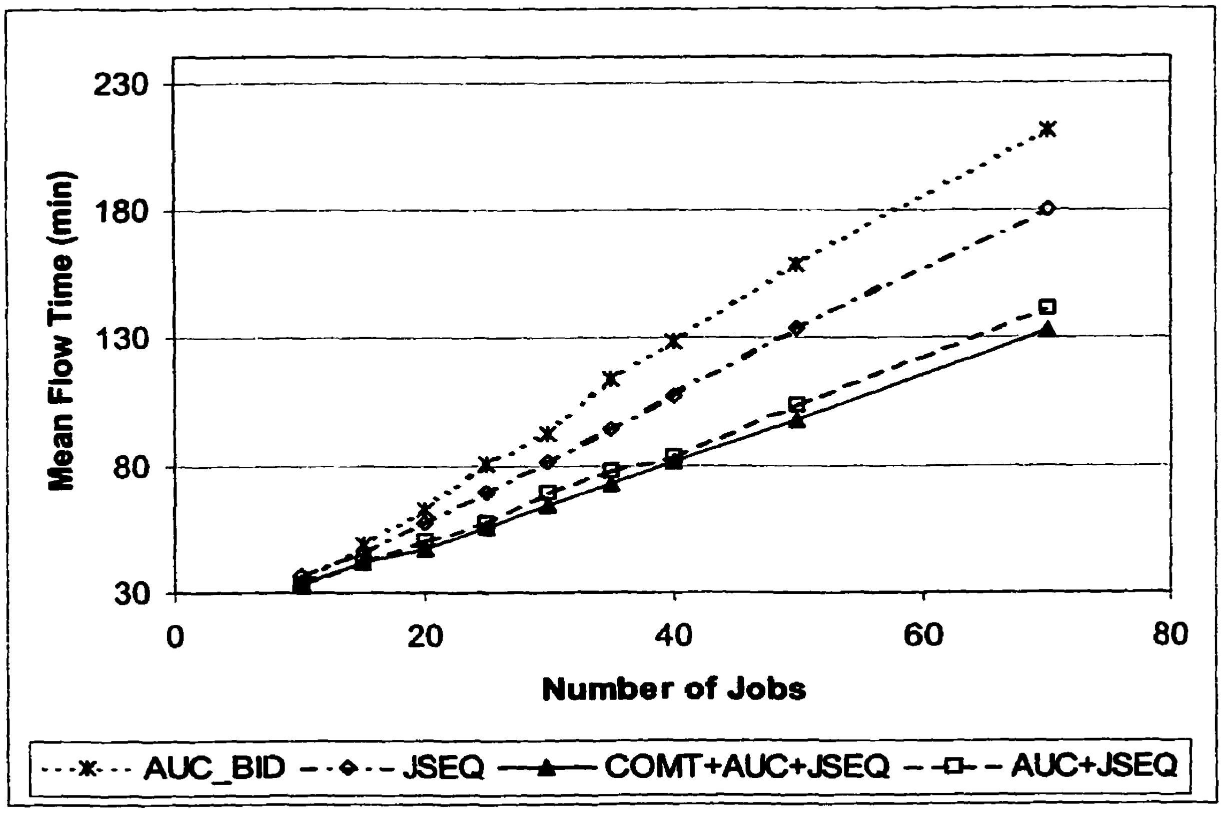


Figure 6.2: The mean flow time (minutes) of various control approaches in different test scenarios for experiment 6A.

6.1.3 Experiment 6B – One Machine Failure Disturbance

In this experiment, we will evaluate the performance of the four control strategies described in the §6.1.1 above in control systems wherein one of the machines in workstation WS1 will be down for 30 minutes after it processes the first job. With the COMT + AUC +JSEQ approach, when the machine failure happens, the affected jobs in workstation WS1 will be notified so that they can explore other routing opportunities. The results of the tests are shown in Table 6.3 below. Figure 6.3 shows the graphical interpretations of the results shown in Table 6.3.

Number of Jobs / Control Strategy	10 Jobs	15 Jobs	20 Jobs	25 Jobs	30 Jobs	35 Jobs	40 Jobs	50 Jobs	70 Jobs
AUC_BID	39	55.9	68	86	102	117	131.7	159.7	218
JSEQ	46.0	59.0	72.2	87.0	96.6	110.2	120.8	144.8	191
COMT + AUC + JSEQ	38.6	45.5	52.7	61.2	68.7	77.7	87.2	104	138
AUC + JSEQ	39.8	47.0	53.3	64.2	75.8	80.6	89	110	148

Table 6.3: The mean flow time (minutes) of various control approaches in different test scenarios for experiment 6B.

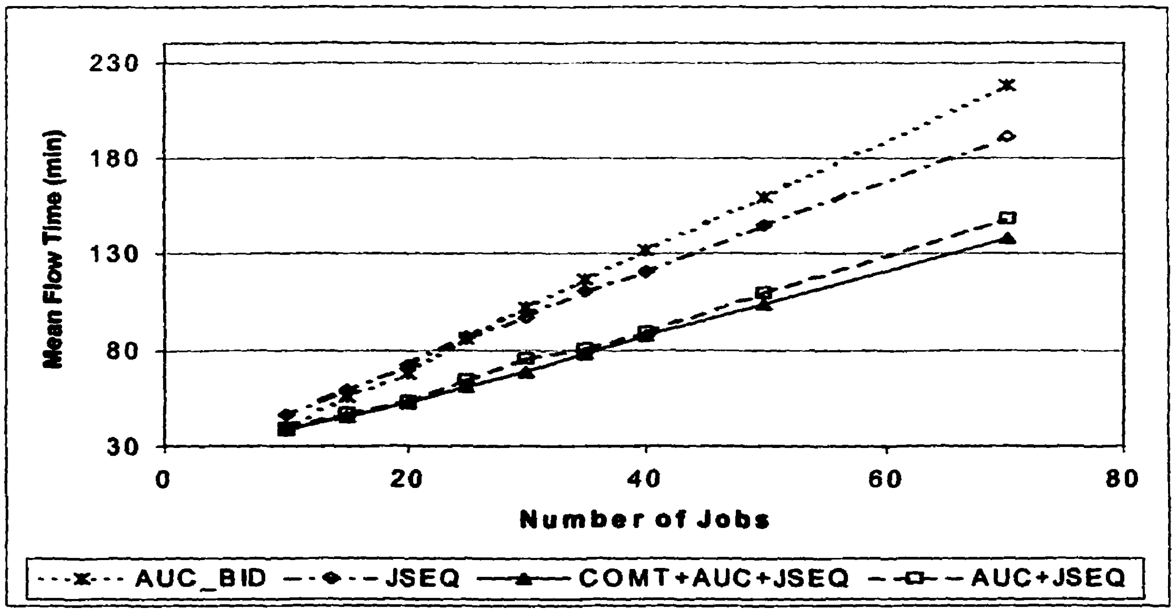


Figure 6.3: The graphical interpretation for the results shown in Table 6.3.

6.1.4 Experiment 6C – Two Machine Failure Disturbances

In this experiment, we will increase the machine failure disturbances by having both the machines in workstation WS1 down for 30 minutes after they have done their first operation. The results of the tests are shown in Table 6.4 below. Figure 6.4 shows the graphical interpretations of the results shown in Table 6.4.

Number of Jobs / Control Strategy	10 Jobs	15 Jobs	20 Jobs	25 Jobs	30 Jobs	35 Jobs	40 Jobs	50 Jobs	70 Jobs
AUC_BID	53.3	64.4	79.7	91.6	110.1	126.8	140	167.2	223
JSEQ	64.2	76.1	91.4	100.6	114	125.2	138	161.4	208
COMT + AUC + JSEQ	51.1	56	64.6	72	81.3	88.4	97	113.1	147
AUC + JSEQ	51.4	61	65.7	76.8	88.7	92.1	101	121.5	151

Table 6.4: The mean flow time (minutes) of various control approaches in different test scenarios for experiment 6C.

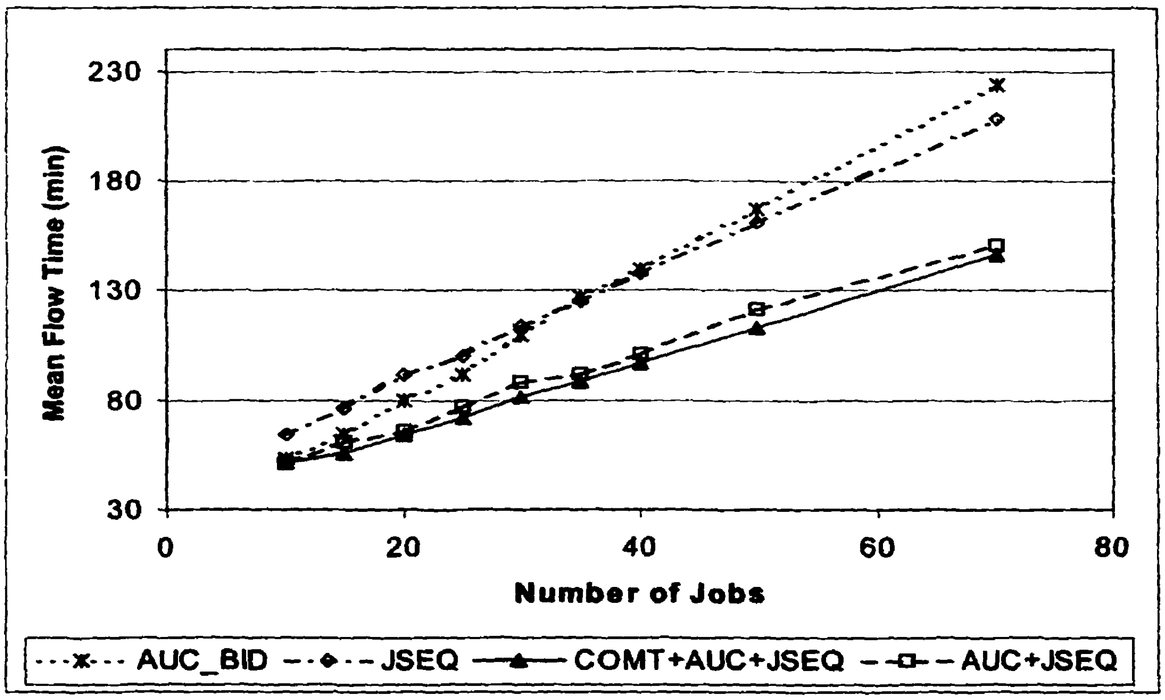


Figure 6.4: The graphical interpretation of the results shown in Table 6.4.

6.1.5 Experiment 6D – Four Machine Failure Disturbances

In this experiment, we will increase the machine failure disturbances by having both the machines in workstation WS1 down for 30 minutes after they have done their first operation, and both machines in workstation WS2 down for 30 minutes after the production has run for half an hour. It should be noted that since the operations of the jobs are non-preemptable, when the scheduled downtime is reached and a machine is operating on a job, the failure of the machine would be initiated after the job is finished. The results of the tests are shown in Table 6.5 and Figure 6.5 below.

Number of Jobs / Control Strategy	10 Jobs	15 Jobs	20 Jobs	25 Jobs	30 Jobs	35 Jobs	40 Jobs	50 Jobs	70 Jobs
AUC_BID	56.3	65.8	82.7	93.2	107.7	125.2	139.5	171	226
JSEQ	64.2	77.9	92.8	103.5	117.2	128.2	141.5	166.6	213
COMT + AUC + JSEQ	50	56.3	66.2	73.4	80.7	88.1	97.1	112.5	146
AUC + JSEQ	51.4	61.1	71.7	76.8	88.5	92.7	100.2	119.6	156

Table 6.5: The mean flow time (minutes) of various control approaches in different test scenarios for experiment 6D.

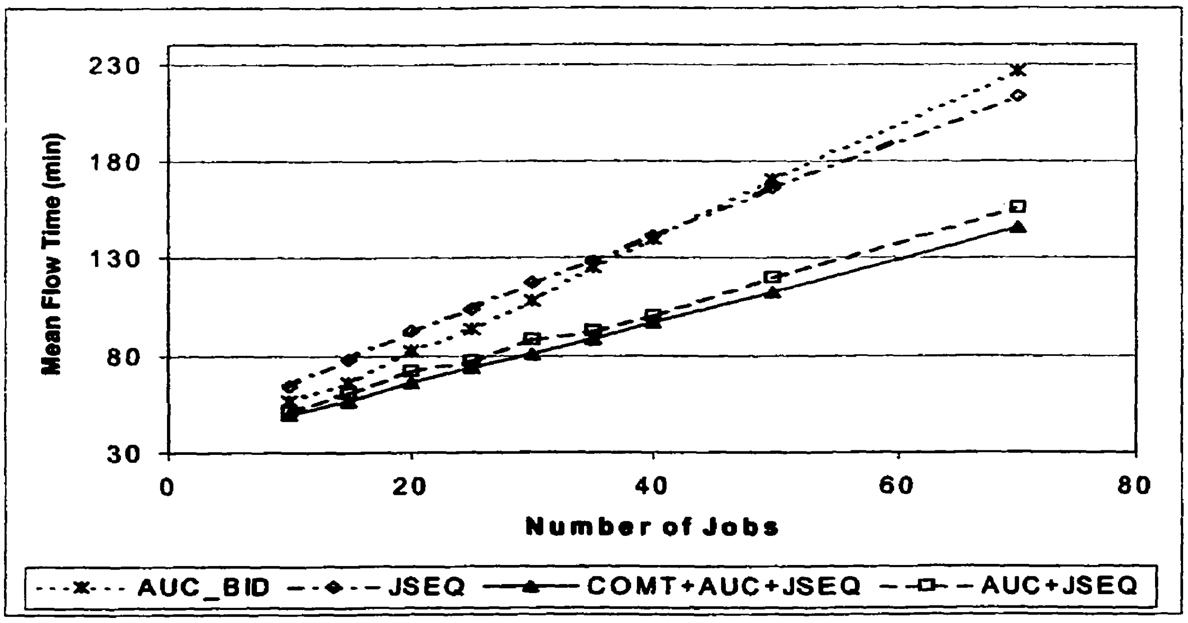
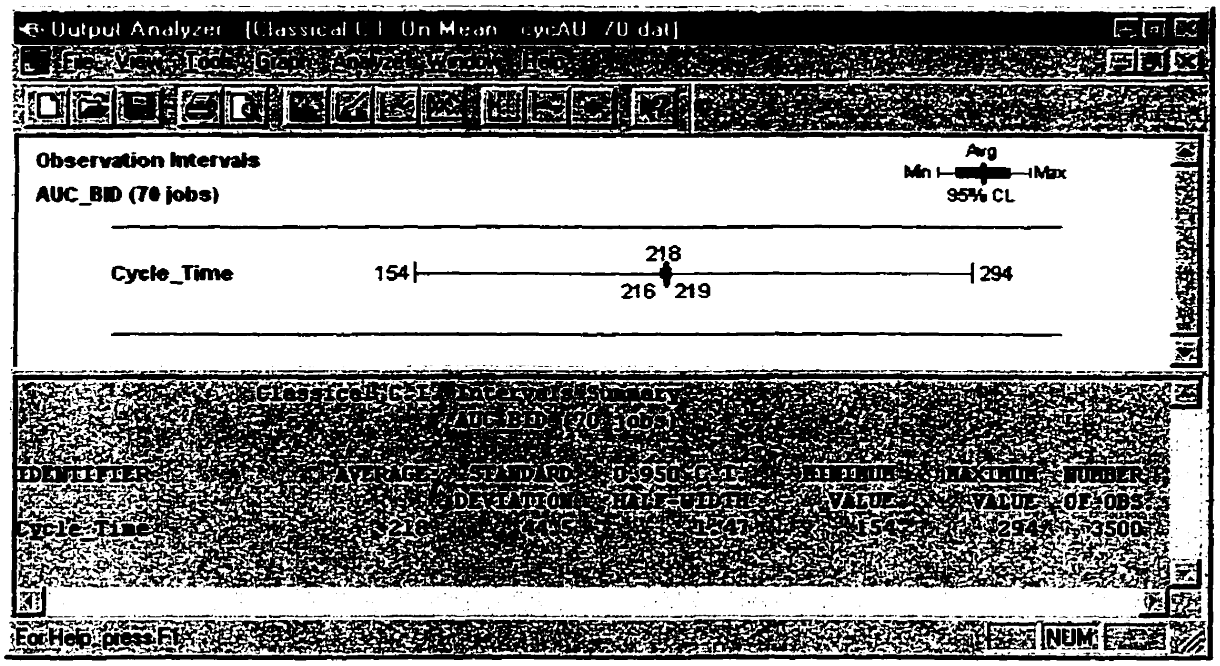


Figure 6.5: The graphical interpretation of the results shown in Table 6.5.

6.1.6 Results Discussion

For each of the test in the above experiments, 50 replications have been run for each simulation to ensure the consistency of the test results. With the 95% confidence interval, the variation of each result is about ±2% or less. For example, Figure 6.6 below shows the 95% confidence interval of the AUC_BID approach in Experiment 6B in the 70-job test case.



```
                    Classical C.I. Intervals Summary
                          AUC_BID (70 jobs)

IDENTIFIER          AVERAGE   STANDARD   0.950 C.I.   MINIMUM   MAXIMUM   NUMBER
                              DEVIATION  HALF-WIDTH   VALUE     VALUE     OF OBS
Cycle_Time          218       44.5       1.47         154       294       3500
```

Figure 6.6: 95% confidence interval result of the AUC_BID approach in Experiment 6B in the 70-job test case.

Referring to Figure 6.2, the experimental results with no disturbances show that, control systems that allow jobs to explore routing flexibility but do not implement job sequencing (AUC_BID) have worse performance than control systems that implemented job sequencing, but do not explore routing flexibility (JSEQ). This is because with the AUC_BID control approach, the workstations processed the jobs on a First-Come-First-Serve basis. But with the JSEQ control approach, the workstations enforce the co-operative behaviors of the jobs based on certain dispatching rules to ensure that certain desirable global objectives can be achieved.

Referring to Figures 6.3~6.5, the experimental results show that when there are machine failure disturbances, in control systems that have implemented the dispatching routing decision-making control algorithm, the opportunistic behaviors of the job agents can help them avoid the bottleneck workstation (workstation with down machines) while making the routing decisions. And this can help improve the system performance, but only in manufacturing environments with production volumes under certain limits. This is because as the manufacturing system's production volume increases, each workstation will be occupied by more jobs at any instant of time, and thus a job agent will have lesser chance to find an alternative workstation that can start processing its other operation sooner. As resulted, in control systems with high congestion, even when there are disturbances, job sequencing can better improve the control systems performance than the routing flexibility control mechanism.

To more clearly illustrate this point, we re-plot the experimental results in Tables 6.2~6.5 based on different production volume categories. Referring to Figures 6.7 (a) to (h), for instance, Figure 6.7 (a) shows the performance of the 4 control strategies in different machine failure test scenarios in control systems with production volume of 10 jobs, and Figure 6.7 (b) shows the results in control systems with production volume of 15 jobs, etc....

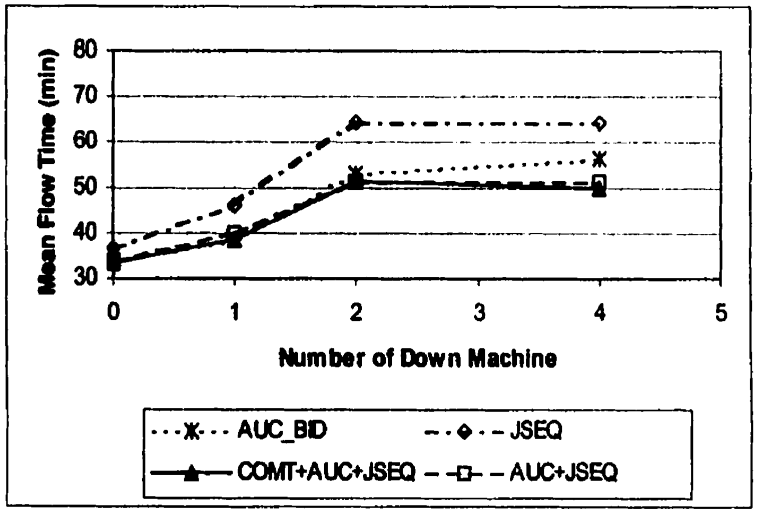


Figure 6.7 (a): Results for 10 jobs.

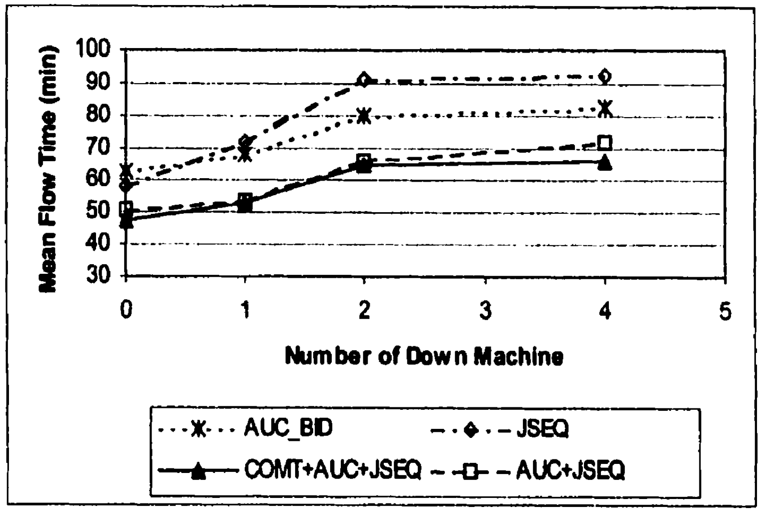


Figure 6.7 (b): Results for 20 jobs.

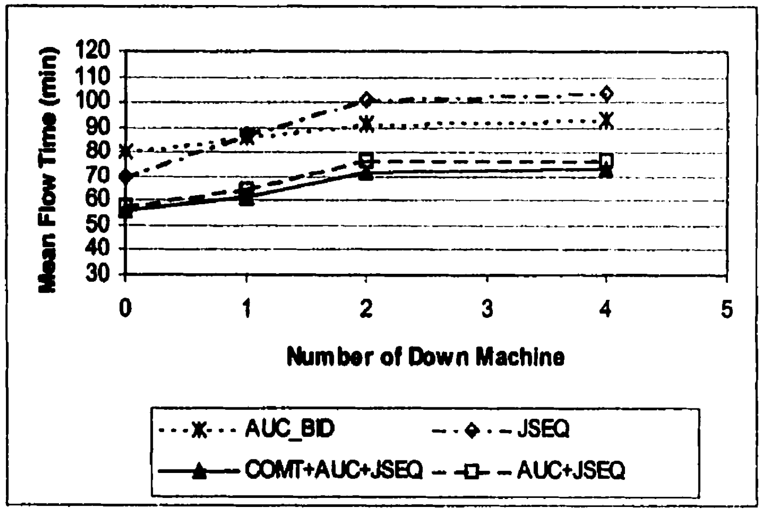


Figure 6.7 (c): Results for 25 jobs.

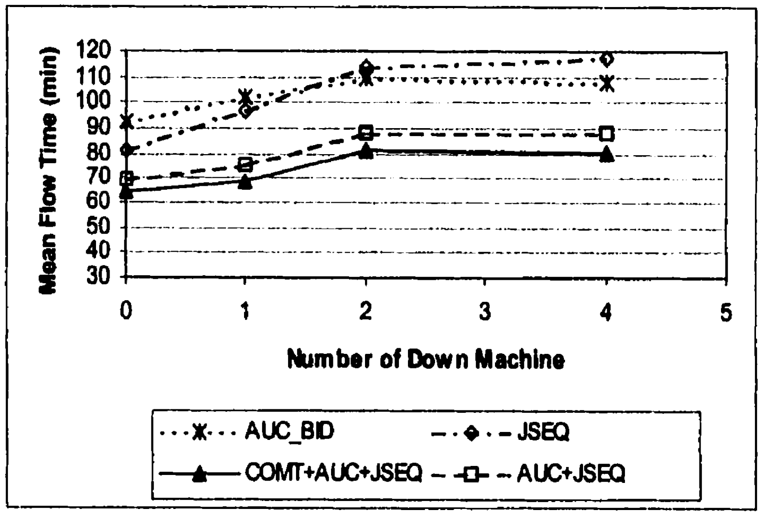


Figure 6.7 (d): Results for 30 jobs.

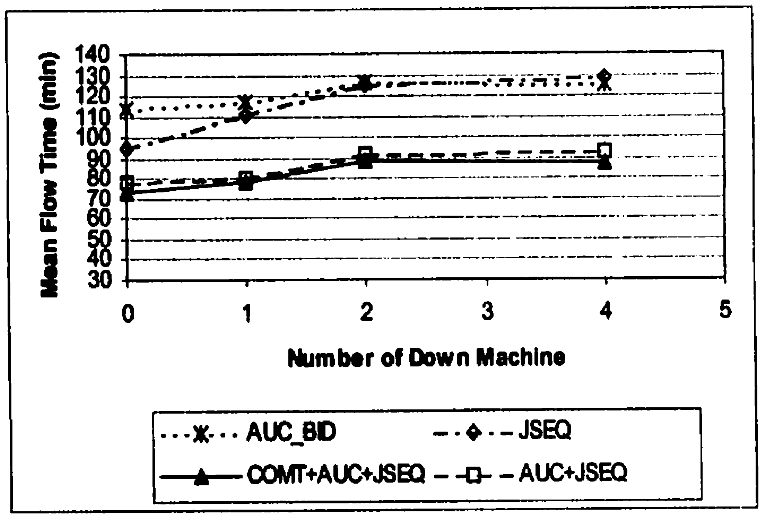


Figure 6.7 (e): Results for 35 jobs.

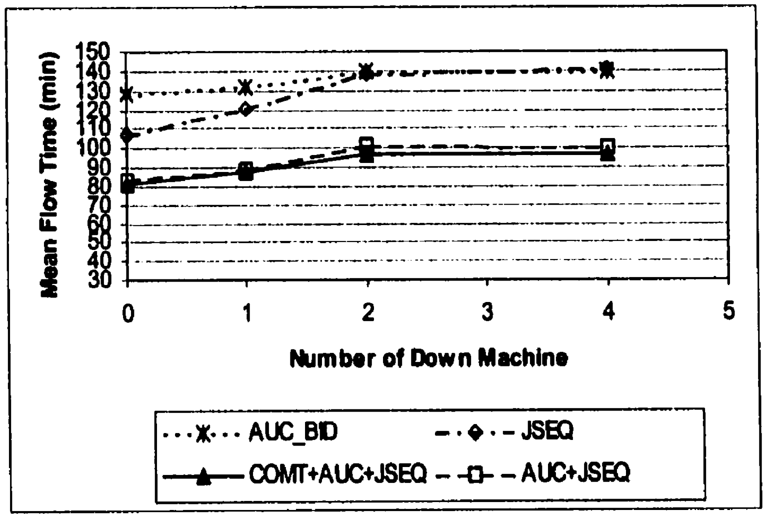


Figure 6.7 (f): Results for 40 jobs.

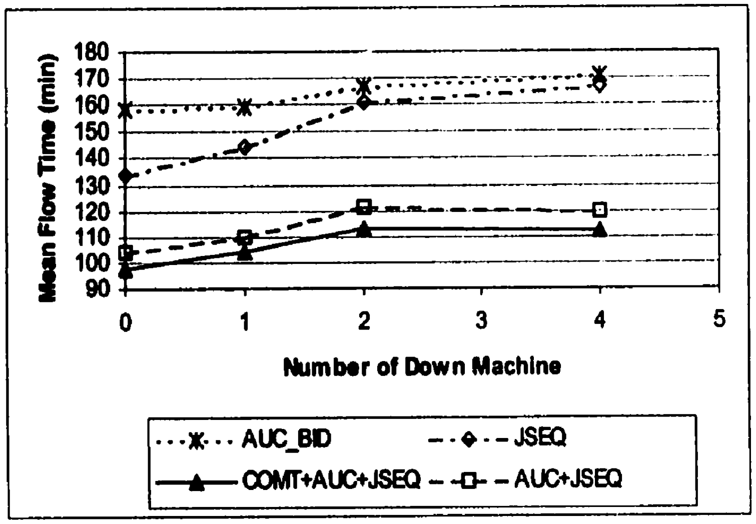


Figure 6.7 (g): Results for 50 jobs.

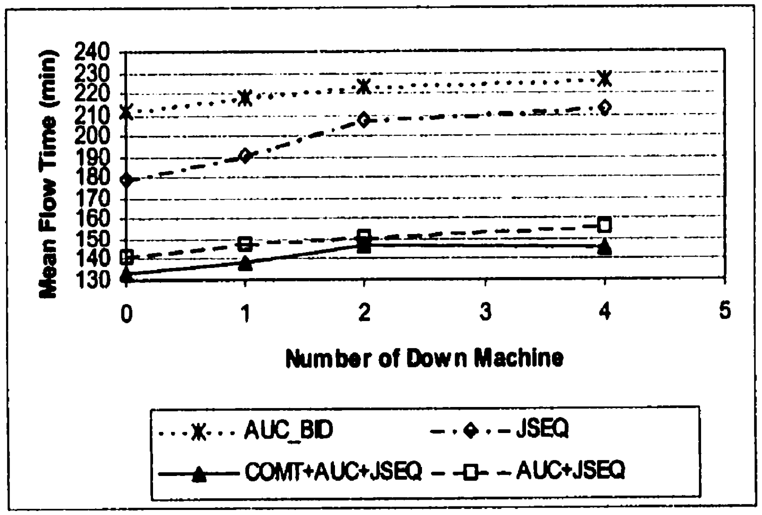


Figure 6.7 (h): Results for 70 jobs.

Referring to Figure 6.7 (a), we can see that when the production volume is small, the jobs can take advantage of the routing flexibility to find a shortest-time path through the production system. Thus in such systems, AUC_BID control approach outperform the JSEQ approach. But as the production volume increase, the control system will have less routing flexibility, and it is more important to coordinate the activities of the jobs to ensure that certain global performance objectives can be achieved. For instance, referring to Figures 6.7 (a) ~ (h), we can see that in control systems with 0 machines down, after the production volume exceeds 20 jobs (Figures 6.7 (b) ~ (h)), the JSEQ approach outperforms the AUC_BID approach.

When there are machine failure disturbances, the opportunistic behavior of the job agents can help solving the control system's bottleneck problem. Thus the AUC_BID approach will outperform the JSEQ control approach. But after the production volume exceeds a certain limit, the routing flexibility of the production system decreases, and the importance of the job sequencing control mechanism starts to kick in. As resulted, the JSEQ control approach will outperform the AUC_BID approach. Referring to Figures 6.7 (a) to (h), we can see that with more machine failure disturbances, the JSEQ control approach will outperform the AUC_BID approach after the production volume exceeds higher limits. For instance, in the 1-machine failure test scenarios, the AUC_BID approach outperforms the JSEQ approach in situations where the production volume is under 25 jobs (Figures 6.7 (a) ~ (c)). After the production volume exceeds 25 jobs, the JSEQ approach outperforms the AUC_BID approach (Figures 6.7 (d) ~ (h)).

But in the 4-machine failure disturbance test scenarios, the AUC_BID approach outperforms the JSEQ approach in situations where the production volume is under 40 jobs (Figures 6.7 (a) ~ (f)). After the production volume exceeds 40 jobs, the JSEQ approach outperforms the AUC_BID approach (Figures 6.7 (g) ~ (h)).

This is because with more machine failures, the opportunistic behavior of the job agents can help ease the production system's bottleneck problems. But after the production volume exceeds a certain limit, the importance of job sequencing start to outweigh the importance of the routing flexibility. As resulted, the JSEQ approach always outperforms the AUC_BID approach in high production volume systems, regardless of the disturbance situations (in respect to the experimental results described above).

Figures 6.8 to 6.11 show the performance of each of the control approaches, respectively, in various test scenarios. From the figures, we can see that in control systems wherein the jobs will explore the routing flexibility (the AUC_BID, COMT+AUC+JSEQ, AUC+JSEQ), the performance of the control systems are less sensitive to the machine failure disturbances (compared to the JSEQ control approach). For instance, we can see that in Figure 6.8, the gaps between the performance lines of the JSEQ control approach in the 4 difference machine-failure disturbance test scenarios are larger than those of the other control approaches as shown in Figures 6.9, 6.10 and 6.11, respectively.

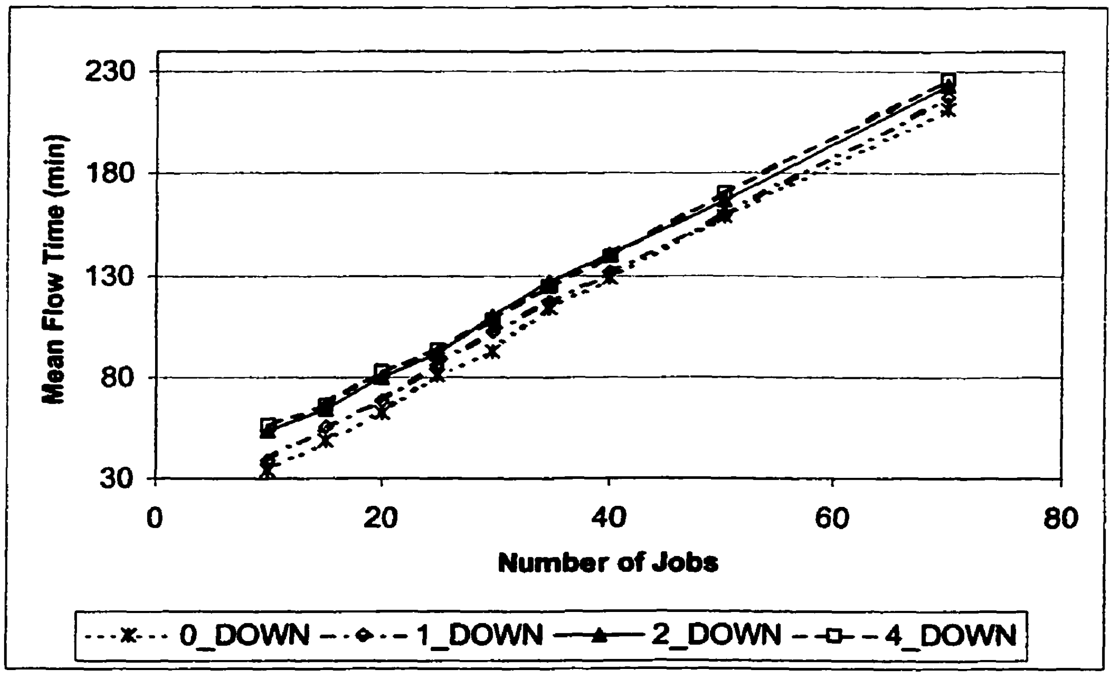


Figure 6.8: The results of the AUC_BID control approach in various test scenarios.

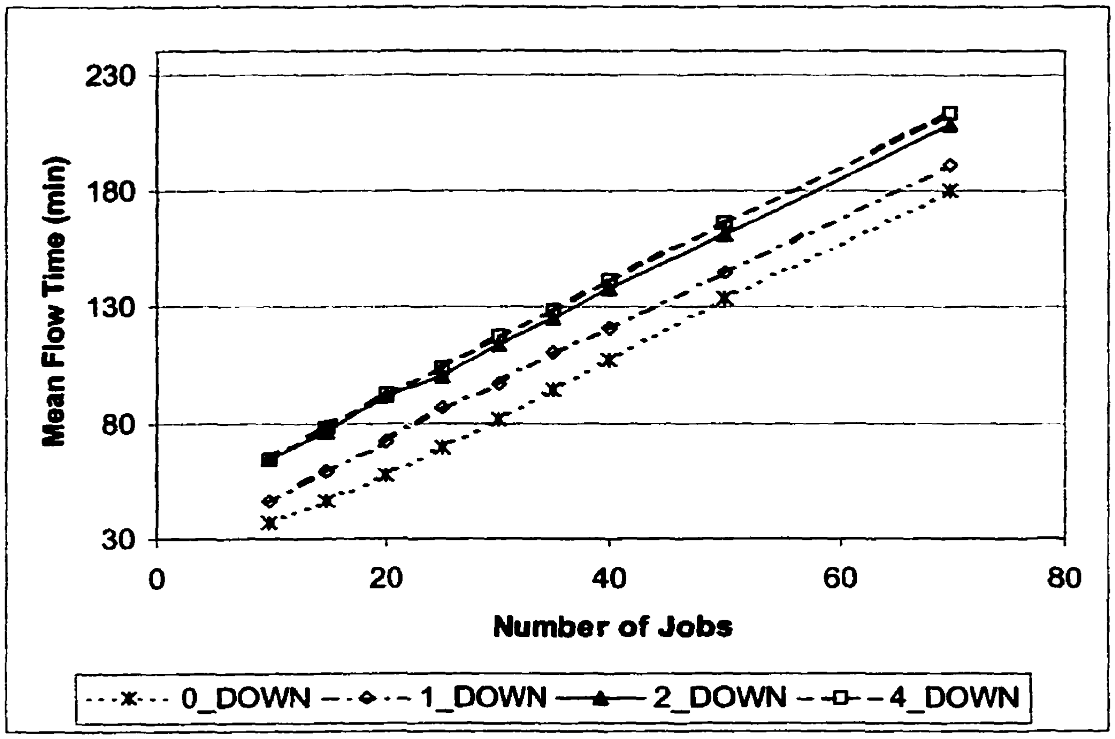


Figure 6.9: Results of the JSEQ control approach in various test scenarios.

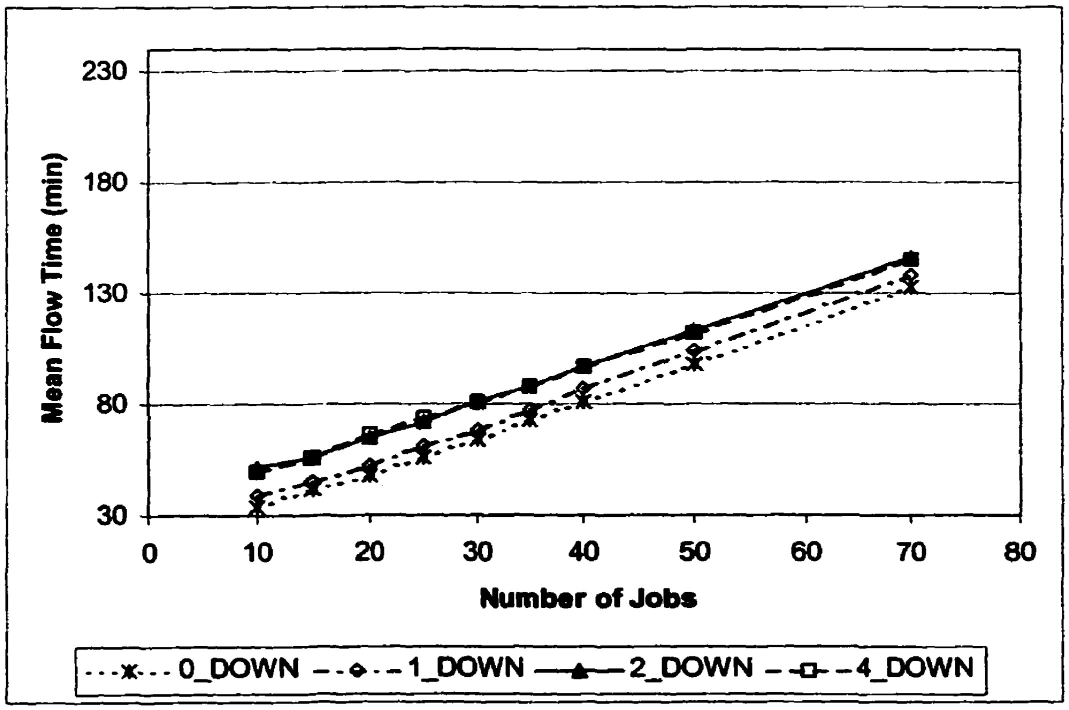


Figure 6.10: The results of the COMT+AUC+JSEQ control approach in various test scenarios.

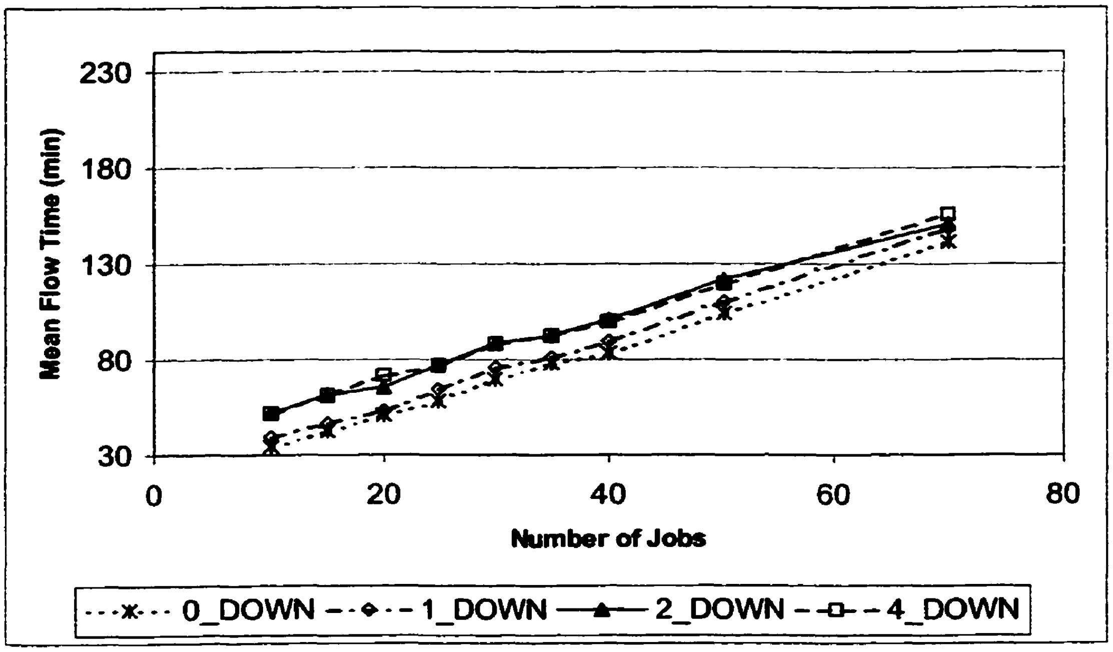


Figure 6.11: The results of the AUC+JSEQ control approach in various test scenarios.

Referring to Figures 6.2 ~ 6.5, we can see that as expected, the control systems that incorporate both the job sequencing and routing flexibility control mechanisms (COMT+AUC+JSEQ, AUC+JSEQ) always have superior performance over control systems that implement only the routing flexibility (AUC_BID) or the job sequencing (JSEQ) control mechanism. This is because in such control systems, while the job agents can explore the routing flexibility (and thus avoid the bottleneck stations when machine failure disturbances happen), the workstation agents will sequence the processing of the jobs to ensure that certain global performance objectives can be achieved.

In the AUC + JSEQ control approach, while deciding which operation to process next, the jobs agents use the contract net auction-bidding approach to make the routing decisions based on the bids (the quoted earliest possible start time) submitted by the workstations. But after a job contracts its operation to a workstation, when the workstation agent used the dispatching rule to sequence the arriving jobs, it might breach the contract that it had made with some of the previously contracted jobs. That is, some of the jobs' 'quoted start time' might be violated. In the COMT + AUC + JSEQ control approach, whenever a job's contract with a workstation is violated, the job will be notified about the situation, so that it can explore other routing opportunities. The following discussion is concerned with the impact of the enhanced opportunistic behavior of the job agents on the control system's performance and communication requirements.

Referring to Figures 6.2 to 6.5, we can see that while the performance of the AUC + JSEQ control approach always significantly outperforms the AUC_BID and the JSEQ control approaches (especially in the high production volume cases), the performances of the AUC + JSEQ and the COMT + AUC + JSEQ control approaches are always very close. To demonstrate these results, in Figures 6.12 and 6.13, we show the performance ratio of the other control approaches versus the AUC + JSEQ control approach in experiments 6A (zero machine failure) and 6D (4-machine failure), respectively.

Referring to Figures 6.12 and 6.13, we can see that in most cases, the results (mean cycle time) of the AUC_BID and JSEQ control approaches are higher than the results of the AUC + JSEQ approach by about 30~50%. But the results of the COMT + AUC + JSEQ control approach only outperform the results of the AUC + JSEQ control approach by about 6% or less in most cases (The experiments 6B and 6C also have similar performance ratio results).

In the COMT + AUC + JSEQ control approach, although providing the job agents with the updated information regarding their status in the workstations where they are residing (queuing), can slightly improve the control system's performance (compared to the AUC + JSEQ control approach); this performance improvement though, comes with a significant increase in the communications between the job and workstation control agents (as to be explained next).

In the COMT + AUC + JSEQ control approach, whenever a job's quoted start time in a workstation is violated, the workstation will inform the affected jobs about the situation. The affected jobs can then explore other routing opportunities to see if they want to stay in the original workstation, or if there are other workstations that can process their other operations sooner. In Table 6.6 below, we measured the total number of times (totChangeOffer) that the jobs have been notified about the change in their 'quoted start time' by the workstations, and the total number of times that the notified jobs actually changed workstations (totChangeQ) in the COMT + AUC + JSEQ approach in experiment 6A.

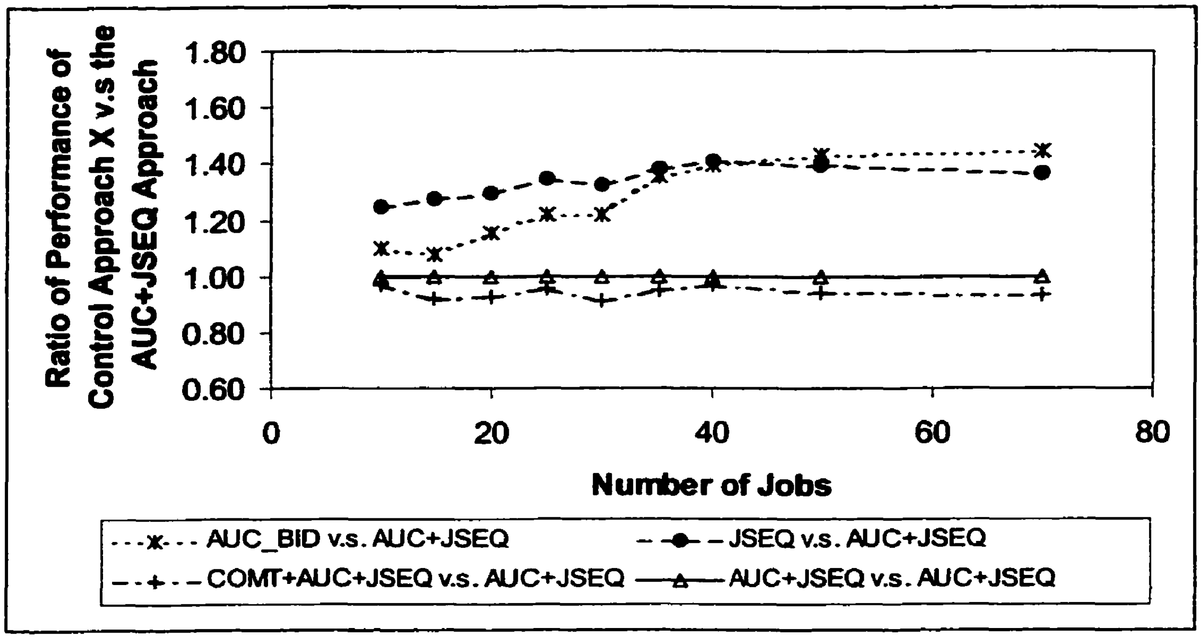


Figure 6.12: Performance ratio of the other control approaches versus the AUC+JSEQ control approach in experiment 6A.

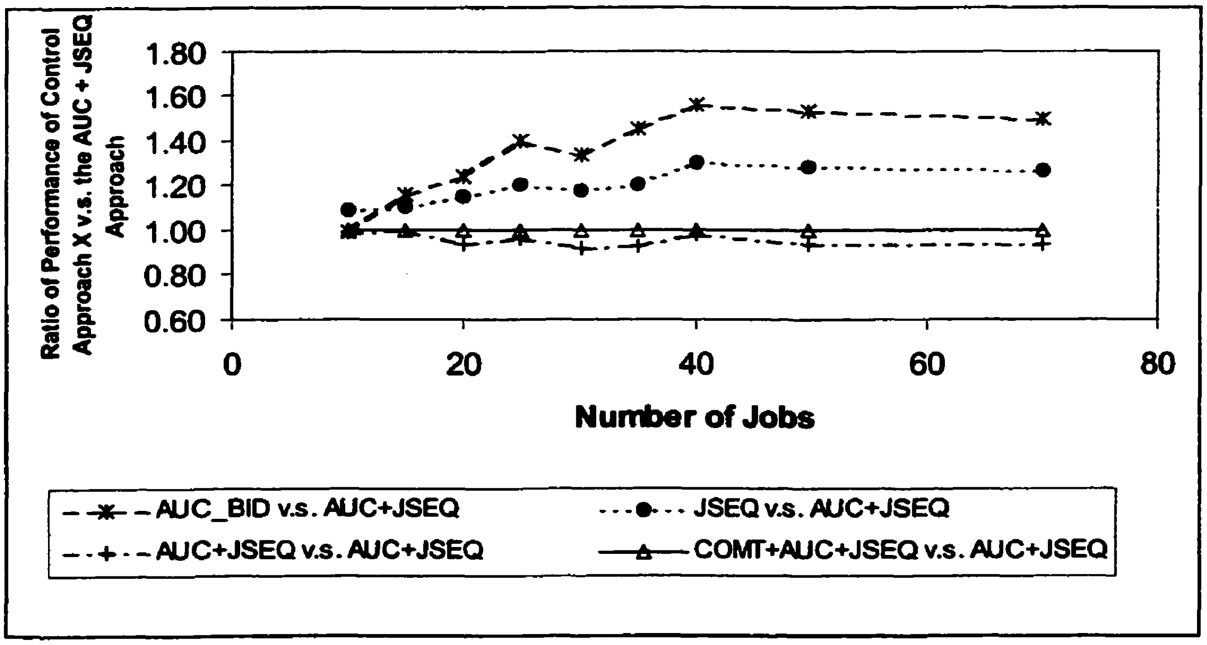


Figure 6.13: Performance ratio of the other control approaches versus the AUC+JSEQ control approach in experiment 6D.

	10 Jobs	15 Jobs	20 Jobs	25 Jobs	30 Jobs	35 Jobs	40 Jobs	50 Jobs	70 Jobs
totChangeOffer (A)	13	91	203	352	592	843	1055	2130	4817
totChangeQ (B)	11	69	146	196	281	329	330	581	933
Ratio of B / A	0.85	0.76	0.72	0.56	0.47	0.39	0.31	0.27	0.19

Table 6.6: The results of the total number of times that the job agents in the COMT + AUC +JSEQ control system had been notified by the workstations about changes in their 'quoted start time', and the total number of times that the jobs actually changed workstations.

Figure 6.14 shows the graphical interpretation of the results shown in Table 6.6 and Figure 6.15 shows the ratio of the totChangeQ versus the totChangeOffer in the various test scenarios. The totChangeOffer results represent the communication frequency between the job and workstation control agents in each test scenario. Every time a job is notified about the change in its 'quoted start time' by a workstation, the job will start contacting other workstations to explore other routing opportunities. Therefore, the higher the totChangeOffer frequency, the more the communications between the control agents in the system.

Referring to Figure 6.14, we can see that the number of the totChangeOffer increases exponentially as the total number of jobs increases. But in Figure 6.15, the results show that as the total number of jobs increases, the ratio of the totChangeQ versus the totChangeOffer decreases. This confirms our earlier explanation that as the production volume increases, there will be less routing flexibility in the control system. As a result, in control systems wherein the production volume exceeds a certain limit, as the totChangeOffer frequency increases, the job agents will spend more time in doing 'unproductive' communications. That is, even though the job agents are being notified about the changes in their 'quoted start time', after exploring other routing opportunities, most job agents ultimately decide to stay in their original workstation (Other experiments have similar totChangeOffer and totChangeQ results as described above).

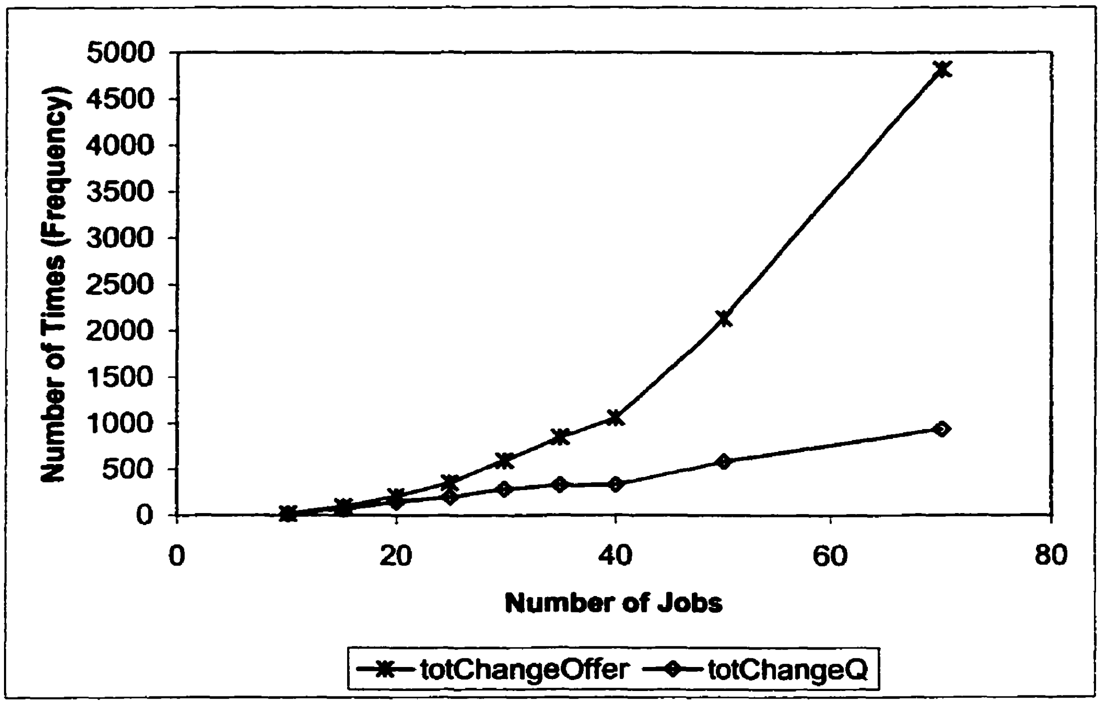


Figure 6.14: Results of the totChangeOffer and the totChangQ frequencies in the experiment 6A.

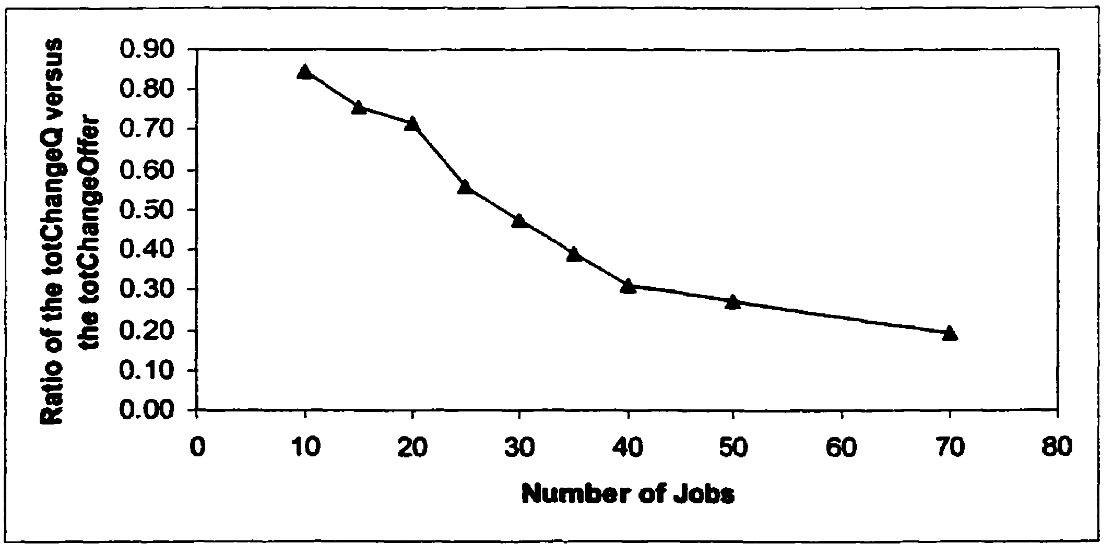


Figure 6.15: The results of the ratio of the totChangeQ versus the totChangeOffer in the experiment 6A.

6.2 Conclusion

In §6.1, we have conducted experiments to investigate and identify the role of the job sequencing and the routing flexibility control mechanisms in different manufacturing environments. The experimental results show that while the routing flexibility can help enhance a control system's flexibility and adaptability against disturbances such as machine failure, it is important to incorporate the job sequencing control mechanism in the control system to ensure that certain global performance objectives can be achieved (especially in manufacturing systems with high production volume). As expected, the control systems that implement both the job sequencing and routing flexibility control mechanisms always have superior performance over control systems that implement only the routing flexibility or the job sequencing control mechanism.

In control systems that implement job sequencing and job routing control mechanisms, sometimes it is justifiable to compromise some of the commitments between the workstation agents and the job agents in order to achieve certain global performance objectives and minimize the communications between the control agents. For instance, to strictly honor the commitments that they have made to the job agents, the workstation agents will either sequence the jobs based on the First-Come-First-Serve rule, or if they use another dispatching rule to sequence the jobs, the jobs has to be notified when their contracts ('quoted start time) with the workstations are violated, so that the job agents can explore other routing opportunities. The experimental results show that in the former case, the performance of the system will be compromised, and in the later case, the communications between the control agents will be significantly increased.

As resulted, we can see that with the AUC + JSEQ and COMT+AUC+JSEQ control approaches, the control systems can achieve certain desirable global objectives (compared to the AUC_BID and JSEQ control approaches). And by having the resource agents responsible for job sequencing (AUC+JSEQ), the communications between the resource and job control agents can be minimized (compared to the COMT+AUC+JSEQ control approach). As well, the AUC+JSEQ control approach also complies with the design principle of distributed control systems, wherein the system consists of a group of loosely-coupled, cooperative control agents: i.e., the control agents in the AUC+JSEQ approach will spend most of their time in computation rather than communication (Smith 1980).

CHAPTER 7

IMPLEMENTING CONTROL AGENTS AS COM/DCOM OBJECTS

In the previous chapters, although the experimental control systems are implemented with the distributed multi-agent control approach, the control agents are not actually distributed in nature (they are all resided in a single processor). But when considering a real-world system, one must face the fact that the agents described in the previous chapters will be distributed across multiple processors. Hence, an inter-operational approach is required. As well, since the benchmark framework (Cavalieri et al. 1999) is intended for different researchers to compare their control methodologies on a common testbed, it would be helpful if the control modules can be built into some platform independent software components that can be easily distributed across a network and/or be integrated into other researchers' logical control models for validation or testing.

Although it is possible to use a variety of programming languages to do the socket-layer programming to build the distributed object model, there are some available technologies that can help simplify the network programming and realize component-based software architecture. DCOM (Distributed Component Object Model) and CORBA (Common Object Request Broker Architecture) are two popular distributed object models that have emerged as standards (Chung et al., 1997).

> "DCOM is the distributed extension to COM (Component Object Model) that builds an object remote procedure call (ORPC) layer on top of DEC RPC to support to remote objects... CORBA is a distributed object framework proposed by a consortium of 700+ companies called the Object Management Group (OMG). The core of the CORBA architecture is the Object Request Broker

(ORB) that acts as the object bus over which objects transparently interact with other objects located locally or remotely." (Chung et al., 1997)

The motivation for the work that follows in this chapter is to explore how technologies such as the distributed object model could be used to create a framework to implement the control structures described in Chapters 5 and 6. And for this research, the COM/DCOM approach was chosen since it is fairly well used, its specifications are fairly well defined, and its software implementation is fairly well prescribed. It should be noted though that any of the other methods mentioned above are equally valid for this type of application. In the following sections, a distributed multi-agent control system will be built, and the control and production processes will be modeled by using the COM/DOCM technology and the discrete-event simulation software, Arena.

7.1 Brief Introduction to COM/DCOM

Component Object Model (COM) is a platform-independent, distributed, object-oriented system for creating binary objects that can interact. COM is not an object-oriented language, but a standard. It specifies the object model and programming requirements that enable COM objects to interact with each other. "By specifying the COM standard on a binary level, one can attempt to arrive at a standard that is independent of the operation system, the transmission medium, and the computer language used for implementation. Extending this with a binary protocol standard, object inter-operation can be made hardware platform and location independent" (Sing et al. 1998). The essence of COM is an agreed binary interface that is based on Remote Procedure Call (RPC) technology with some wrappers that form the concept of objects and interfaces between the objects (Bates 1999).

As is defined in the Microsoft Developer Network CD (1998), "A critical part of COM is how clients and servers interact. A COM server is any object that provides services to clients. These services are in the form of implementations of COM interfaces that can be called by any client who is able to get a pointer to one of the interfaces on the server object. A COM client is whatever code or object gets a pointer to a COM server, and uses its services by calling the methods of its interfaces. There are two main types of servers, in-process and out-of-process. In-process servers are implemented in a dynamic linked library (DLL), and out-of-process servers are implemented in an EXE file. Out-of-process servers can reside either on the local machine or on a remote machine." A COM object can play the role of a server, a client or both. All clients must interact with a COM server through its interfaces.

Figure 7.1 represents a COM object. The object is represented by a box and its interfaces are represented by plugs. Each COM object can have several interfaces. An interface is a table of function pointers, and it represents a well-defined binary contract between the COM object and its client.

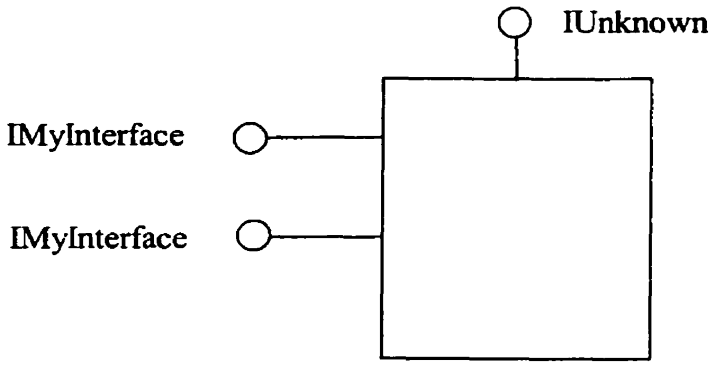


Figure 7.1: A COM object diagram.

Conventionally, the interface on the top represents the IUnknown interface, which is the base interface inherited by all other COM interfaces. The IUnknown interface provides three functions (methods), namely AddRef(), Release() and QueryInterface(). AddRef() and Release() are reference counting mechanisms for COM objects to manage their lifetimes. Each COM object has an internal counter that holds the number of users referencing the component. As suggested by its name, QueryInterface() is used by a client to query if a COM server supports a particular interface. If it does, a pointer to the required interface will be returned to the client. Since all COM interfaces are based on IUnknown, they must also implement the AddRef(), Release() and QueryInterface()

methods. Therefore, given any interface pointer to an COM object, a client should also be able to obtain any other interface supported by the object by calling QueryInterface() on the existing interface pointer.

Distributed COM (DCOM) extends COM so that COM clients and servers can all run on a single machine or be distributed across a wide area network.

7.2 Building a Distributed Manufacturing Control System with COM/DCOM

7.2.1 Design Background

In order to enhance a control system's adaptability and flexibility against disturbances such as machine failure or uncertain processing times, researchers have proposed the real-time distributed scheduling and control approach for shop floor manufacturing system (Duffie et al. 1994, Saad et al. 1997, Zhang et al. 1999). The most commonly used distributed scheduling and control approach is to use the contract-net (Smith 1980) auction-bidding protocol to allocate manufacturing resources to jobs. In such an approach, when a job arrives, it will request machines in the system to submit bids for its first operation. Upon receiving the job's request, machines that can perform the operation will evaluate their task agenda, then reply to the job with a message containing information such as the earliest time they can start/finish the operation, and/or the number of jobs that have already reserved the usage of the machines. The job will then evaluate all the responses based on some criteria and choose a machine to reward the operation to it. The job will confirm with the selected machine about the reservation, so that the machine can allocate a time slot in its task agenda for the job. The job will repeat the afore-mentioned procedures to find a machine for its remaining operations.

Due to the fact that in a heterarchical control system, entities use purely localized information and all forms of hierarchy are eliminated, heterarchical control result in problems with global optimization and predictability of system behavior. In an attempt to combine the best features of hierarchical ("top down") and heterarchical ("bottom up", "cooperative") control structures, some researchers (Van Brussel et al. 1998, Bongaerts et al. 1998, Zhang et al. 1999) have proposed the Holonic Manufacturing concept to preserve the stability of hierarchy while providing the dynamic flexibility of heterarchies. Valckenaers et al. (1997a) have defined the Holonic Manufacturing System (HMS) as "system components of autonomous modules and their distributed control. A holonic manufacturing architecture shall enable easy (self-)configuration, easy extension and modification of the system, and allow more flexibility and a larger decision space for higher control level".

The following list of definitions are developed by the HMS consortium to help understand and guide the translation of holonic concepts into a manufacturing setting (Van Brussel et al. 1998):

- Holon: An autonomous and co-operative building block of a manufacturing system for transformation, transporting, storing and/or validating information and physical objects. The holon consists of an information processing part and often a physical processing part. A holon can be of another holon.

- Autonomy: The capability of an entity to create and control the execution of its own plans and/or strategies.

- Co-operation: A process whereby a set of entities develops mutually acceptable plans and executes these plans.

- Holarchy: A system of holons that can co-operate to achieve a goal or objective. The holarchy defines the basic rules for co-operation of the holons and thereby limits their autonomy.

A holonic control architecture also captures the concepts of aggregation and specification. "Aggregated holons are defined as a set of related holons that are clustered together and form in their turn a bigger holon with its own identity. As such, an aggregation hierarchy is formed, which is open-ended at the top and at the bottom." and "specification separates the holons with respect to their characteristics " (Van Brussel et al. 1998). Although there is a rich literature on distributed (multi-agent) or holonic control systems, most research is based on the architectural discussion, and few have disclosed how modular control entities (agents or holons) can be built, distributed (across a network) and integrated into a production control system. In this chapter, we will use the above-mentioned distributed control approach and holonic concepts to build a shop floor control system, and simulate the (distributed) control and production processes by using the COM/DCOM technology and the discrete-event simulation software, Arena.

7.2.2 Experimental Model Design and Implementation

The characteristics of the production and control model are listed as follows:

1. The production system contains a number of manufacturing resources, which include workstations and machines.

2. Each workstation or machine can offer a single type of operation.

3. Set-up time for each operation and transportation times for moving jobs between manufacturing resources are ignored.

4. The processing order of a job's operations is not important.

The roles and responsibilities of different holons presented in our model are described as follows:

Job holon – Each job is represented by a job holon, which is responsible for initiating the auction-based bidding process to find the resources for the job's operations, and monitor the job's production progress.

Station holon - A workstation can contain a number of homogeneous machines. Therefore, a station holon's responsibilities are to assign tasks to the machines it manages, to monitor the production progress of the machines and to response to the job holon's bidding request.

Machine holon – It was pointed out in (Dilts et al. 1991) that the functional limitations of some commercially available low-level controllers can prevent the application of intelligent subordinate controllers. Therefore in our experimental model, we define two types of machine holons, namely machSimp (the simple machine) holon and machIntel (the intelligent machine) holon. As was disclosed in the previous sections, in order to carry out the resource bidding process, each resource must have the capability to respond to a job holon's bidding request. MachIntel holon represents the machine with the controller that has the information processing and communication capability to participate in a bidding process, and bears similar responsibilities as a station holon. MachSimp holon represents the machine with a controller that can only perform simple operation recording duties. As we will see in the later, the machSimp holons are usually aggregated with the station holon to form a workstation. Figure 7.2 shows the specialization of machine holon in the UML (Unified Modeling Language) notation. The arrow with the hollow triangular end indicates that both the machSimp and machIntel holons 'is-a' machine holon.

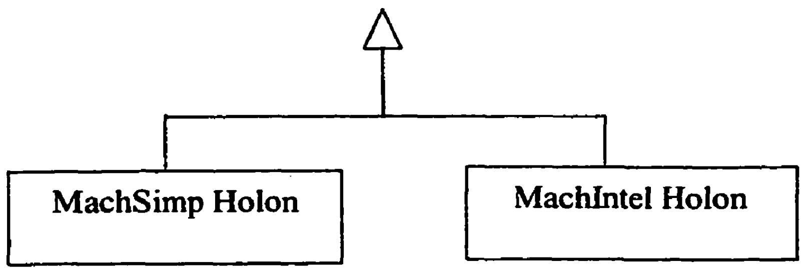


Figure 7.2: Specialization of machine holon.

Mediator holon – The mediator holon is similar to the Yellow Page agent defined in (Shen et al. 1999). It is responsible for registering the manufacturing resources in the system, and responding to the job holon's query regarding which resource in the system can perform a particular type of opreation.

Referring to the holon definition stated above, a holon consists of an information processing part and often a physical processing part. In our experimental model, the information processing part of a holon is represented by a COM object, and the physical part is represented by the corresponding entity in the simulated production system in Arena. The COM diagram for the 5 holons mentioned above are shown in Figures 7.3 – 7.7.

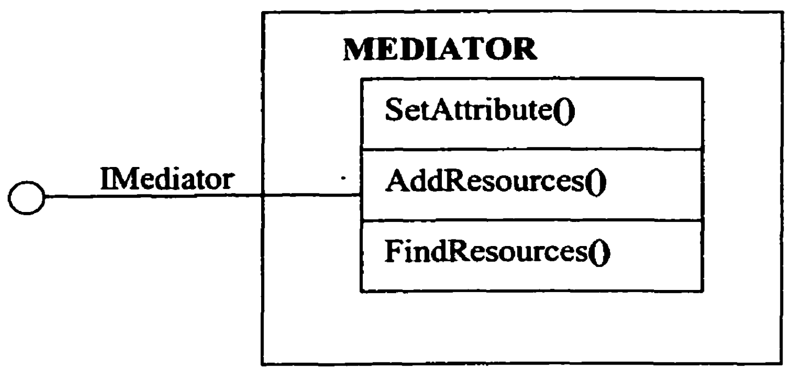


Figure 7.3: The mediator COM diagram.

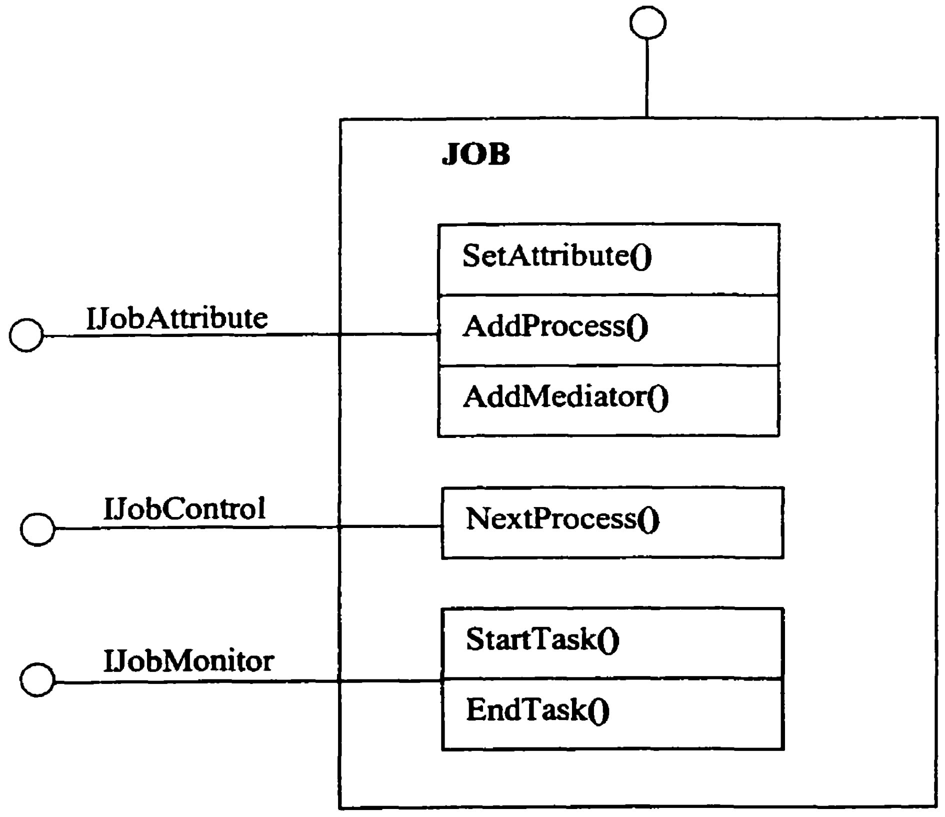


Figure 7.4: The job COM diagram.

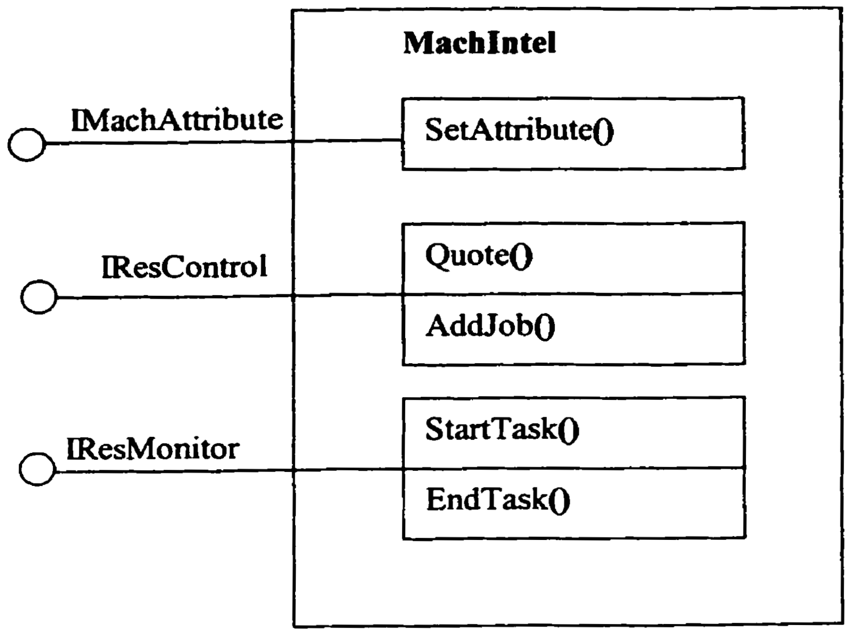


Figure 7.5: The machIntel COM diagram.

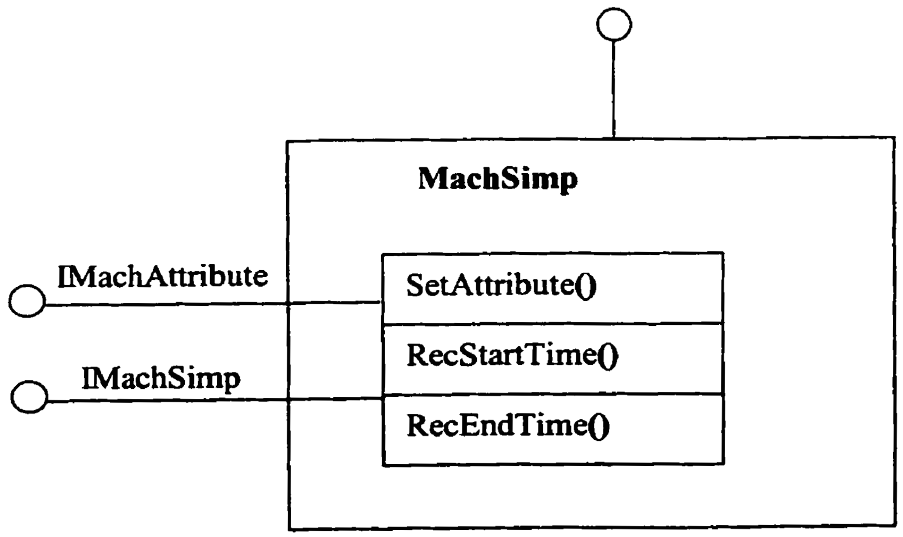


Figure 7.6: The machSimp COM diagram.

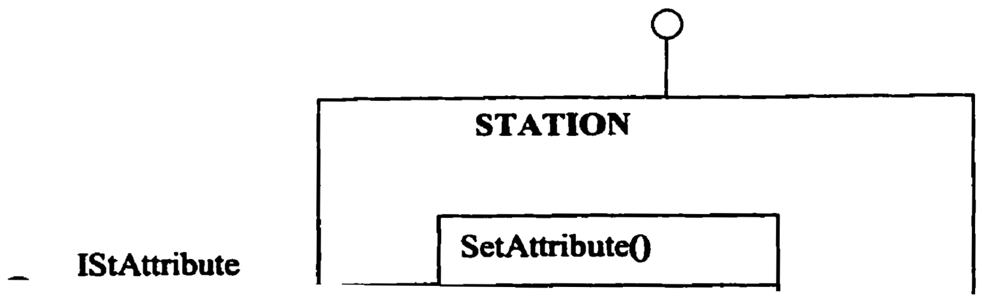


Figure 7.7: The station COM diagram.

As we have mentioned earlier, each holon (except the meidator holon) represents the controller of a corresponding manufacturing entity in the Arena model. In the following, we will present an example to demonstrate the interaction model of the holons and the production processes. In our example, the production system will contain the following resources and job types:

1) A workstation (Station 100) contains 2 machines (Mach 10 and Mach 20 of MachSimp type) and can provide the drilling operation.

2) A single machine (Mach 200 of MachIntel type) that can perform the milling operation.

3) A single machine (Mach 300 of MachIntel type) that can perform the cutting operation.

4) There are three job types. Each job has 2 operations and the processing order of the operations is not important. Table 7.1 lists the operations for each of the job type.

Job	Operation 1	Operation 2
Type A	Drilling	Milling
Type B	Milling	Cutting
Type C	Drilling	Cutting

Table 7.1: Operation list for the 3 job types.

The production plant layout is shown in Figure 7.8. At the beginning of the simulation,

1) A mediator COM object is created.

2) A station and 2 machIntel COM objects are created.
   - The attributes of the station and the machIntel objects (such as resource number, function type) are set via the SetAttribute method.
   - As one can see for the station object, there is an AddMach method in its IStAttribute interface, this is for creating and initializing the (MachSimp) machines that it contains.

3) The instantiated station and machIntel objects register with the mediator via the AddResources method of its IMediator interface, so that the mediator will know what resources are available in the system, and what function each resource can offer.

4) A job COM object is created for each of the jobs introduced into the system.

- Since a job has to contact the mediator to query about the resource that can perform its operations, a job is informed about the existence of the mediator via the AddMediator method of its IJobAttribute interface.

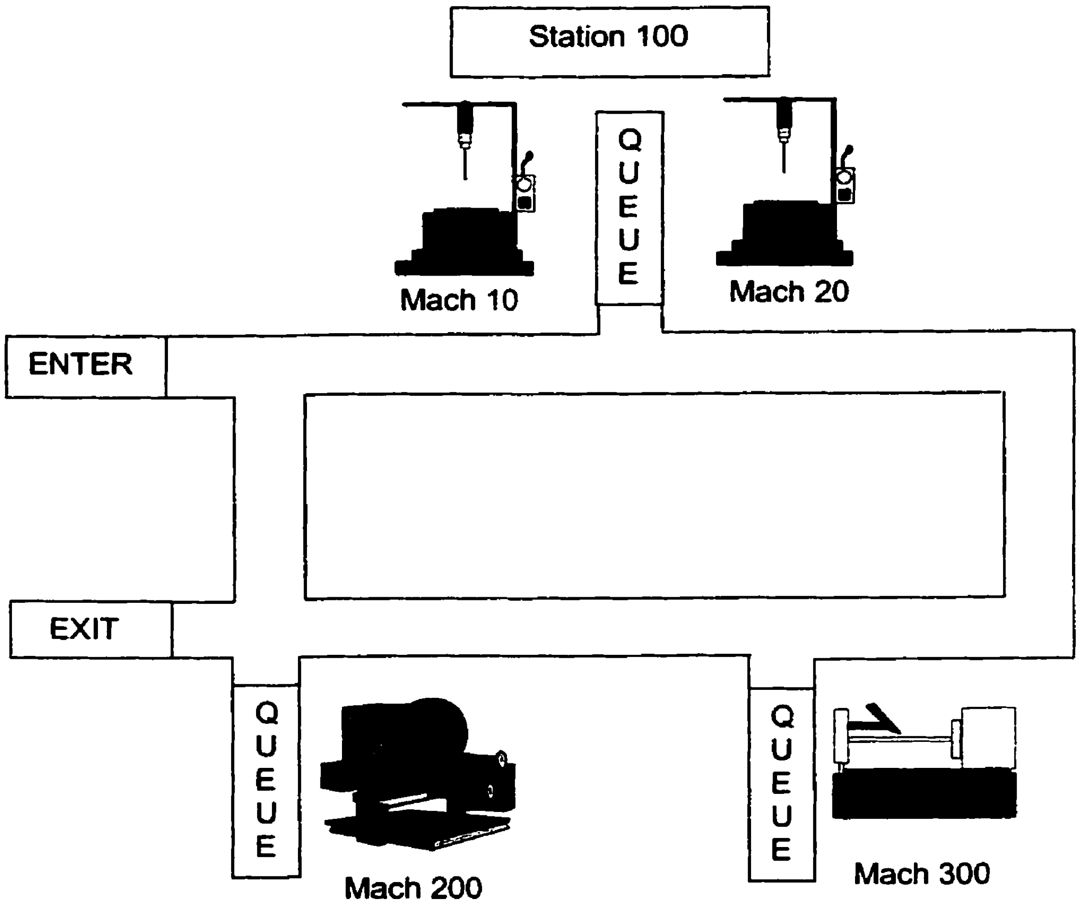


Figure 7.8: The production plant layout.

After the jobs and the manufacturing resources are instantiated, each of the jobs will start finding the resources for their operations. From hereon, we will regard the above-mentioned COM objects as holons. The resource reservation bidding processes are as follows:

1) To find a resource for its next operation, the job holon will ask the mediator holon (via the FindResource method of its IMediator interface) which resources can do the selected operation type.

2) The mediator holon answers the job holon with the corresponding resource address. The job holon then contacts the resources (station or machIntel holon) for a quote (when can it start the operation, how many queuing jobs are there now).

3) Since the processing sequence is not important for a job, a job holon will try to do an operation that can start on a resource earliest. Therefore, the job holon repeats steps 1 and 2 for all of its remaining operations, and then select an operation with the resource that has the best quote (can process the job earliest, or if there's a tie, the second criterion will be the one with the least jobs in queue).

4) The job holon contacts the selected resource to add itself to the resource's reservation list.

5) The job moves to the selected resource's location.

Referring to Figures 7.7 and 7.5, we can see that each station and machIntel COM object has to support an IResControl interface which provides the 'Quote' and 'AddJob' methods for a job COM object to request for a quote and confirm the resource reservation, respectively. When the mediator holon answers the job holon with the address of the resource, the job holon doesn't need to know what the exact type of the resource is. It will contact the resource through the same method (with the same parameters) of the same interface (IResControl). This provides the robustness for using different types of resource controllers. As long as the controllers support the IResControl interface, how they implement the 'Quote' and 'AddJob' methods is irrelevant.

When it is time for a machine to start processing a job in the simulated production system, the corresponding station/machIntel holon will be notified. The station/machIntel holon will then notify the job holon via its IJobMonitor interface about the start of the operation (so that a job holon can keep track of its production progress). For a machIntel holon, it will then record the start time of the operation (for some statistical study purpose). For a station holon, after contacting the job holon, it will delegate the operation recording duty to its selected, contained machine (machSimp) holon. Figure 7.9 shows the containment diagram of a station object. Since the (machSimp) machine holon (or controller) has the capability to record the operation time (refers to Figure 7.6), therefore, it will be reasonable for the station holon to delegate this task to its contained machSimp holon (each machine contained in a workstation is represented by a machSimp holon) via the RecStartTime method of its IMachSimp interface. The same procedures are carried out when a machine finishes an operation in the production system. Once again, one can see that both the station and machIntel objects have to support the IResMonitor interface, so that when the Arena application notifies the station/machIntel holon about the start/end operation event, it doesn't need to know what exact type of resource it is communicating with, even though the station and machIntel holons implement the StartTask/EndTask methods in different ways.

In the above, we have seen that how the different holons can interact with each other to carry out the control of the production processes. After we have developed the COM objects, we can actually distributed them over the network, and have them interact with each other as described above to simulate the communication and co-operation of the actually controllers distributed in a production plant. Figure 7.10 shows the layout of our networking model. In our model, the Arena application was run on the same computer as the mediator, job and mach 200 holons. The mach 300 and station 100 holons were distributed to another computer that was connected to the Arena computer.

The production simulation worked in the same way as described previously. To monitor the status of the holons, we can have each holon log all its activities in a local database. Since the job, machIntel, and machSimp holons all keep records of the operation start/end times, we have each of the holon record the times in a local file (local database). This local data can provide a channel for someone (such as centralized staff controller) to check on the status of these holons at any instant of time at any location by viewing the data through a browser. For example, to view the status of the mach 300 and station 100 holons from the Arena computer, we launch an internet browser to view what resources are running on the ‘other computer’ (as shown in Figure 7.11). Then to view the status of the station 100, we just choose the WStation item. Figure 7.12 shows the status of the station 100 at time 0. As we can see, at time 0, job 4 first joined the station, and the machines of the station were idle at that time (JX indicates no job is loaded on the machine). Then job 4 was loaded to mach 10 and job 3 arrived. Then job 3 was loaded to mach 20 and job 1 joined the station. Since no machine was available then, job 1 stayed in the queue (In our example, a number of jobs with job types A, B and C were created).

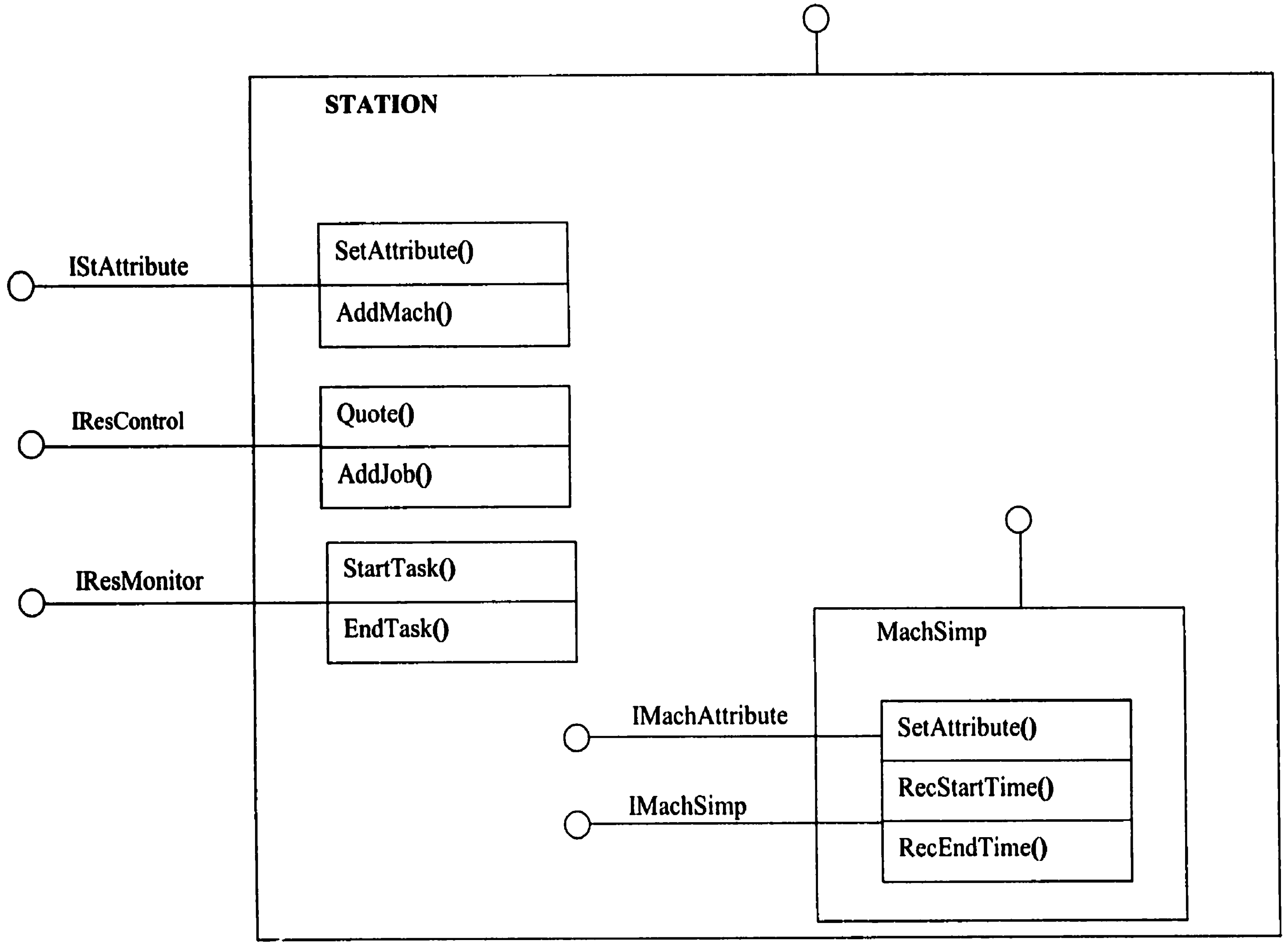


Figure 7.9: The containment diagram of the station object.

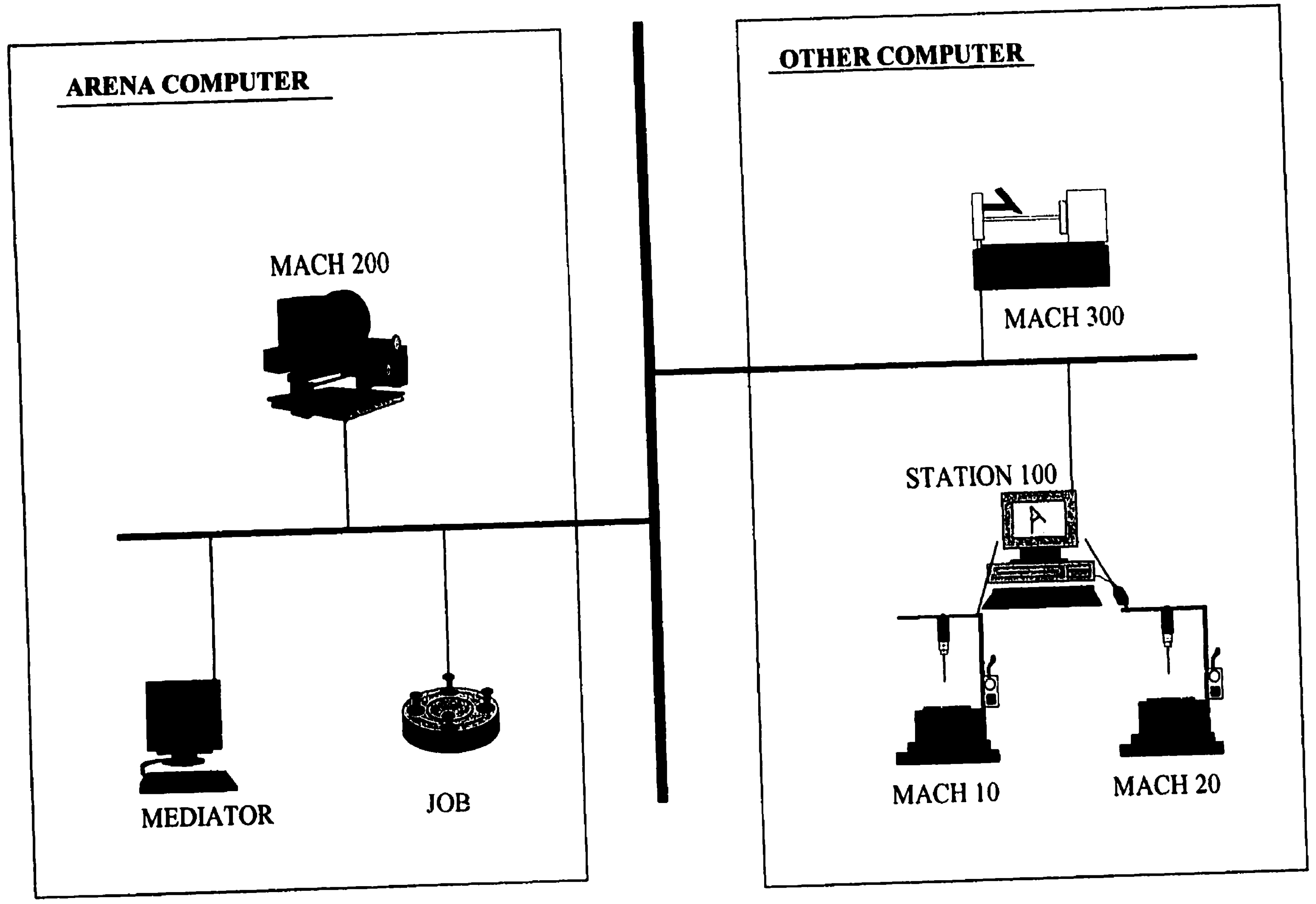
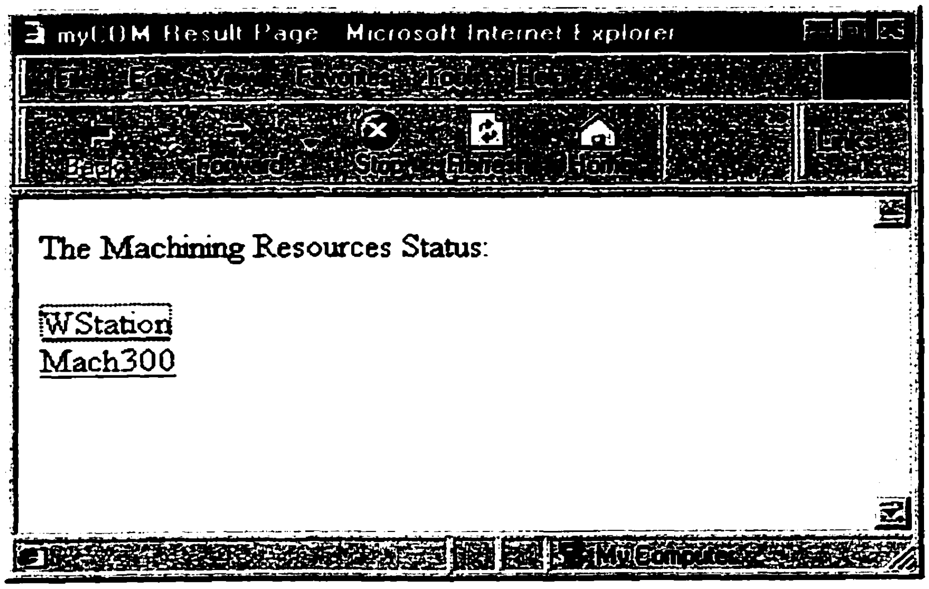


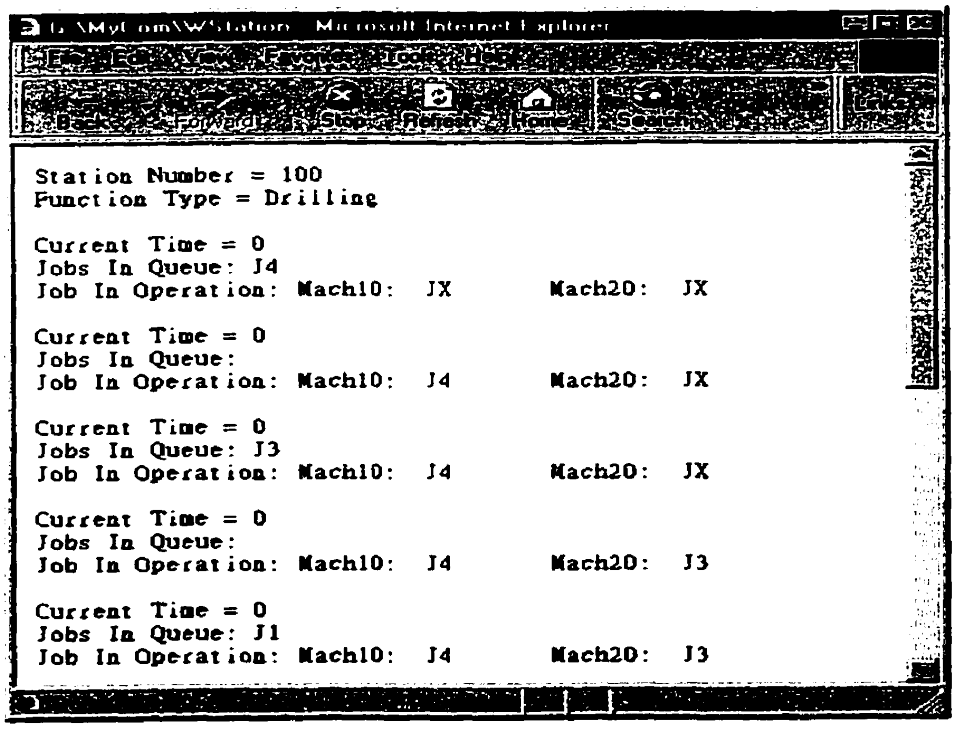
Figure 7.10: The layout for the networking model.



The Machining Resources Status:

WStation
Mach300

Figure 7.11: The manufacturing resources on a network computer.



```
Station Number = 100
Function Type = Drilling

Current Time = 0
Jobs In Queue: J4
Job In Operation: Mach10:   JX        Mach20:   JX

Current Time = 0
Jobs In Queue:
Job In Operation: Mach10:   J4        Mach20:   JX

Current Time = 0
Jobs In Queue: J3
Job In Operation: Mach10:   J4        Mach20:   JX

Current Time = 0
Jobs In Queue:
Job In Operation: Mach10:   J4        Mach20:   J3

Current Time = 0
Jobs In Queue: J1
Job In Operation: Mach10:   J4        Mach20:   J3
```

Figure 7.12: The status of station 100 at time 0.

7.3 Conclusion

In this chapter, we have discussed how to develop the different control roles (or holons) into the COM modules (objects) that can be easily distributed over a network of computers. As one can see in the previous sections, it doesn't matter who takes what role. But for a controller (holon) to take a particular role, the controller must have the capability to fulfill the responsibilities of that role. This provides the flexibility and robustness that for various controllers (servers) that support the same interface, other controllers (clients) can communicate with these controllers via the same interface, without having to differentiate their types, and different controllers can implement the responsibilities in different ways. Also, it's easy to modify an entity's (controller's) responsibilities by having it support/not support a certain interfaces. Referring to the station and machSimp objects, it demonstrates the software reusable advantage wherein, we can create an object that uses some of the functionality of an existing object without duplicating that functionality in the new object.

With the help of object-oriented analysis and design technique, we can identify the roles and responsibilities in a manufacturing control system, and then assign the roles to the entities that have the capabilities to fulfill the corresponding responsibilities. These (control) entities can be easily developed into COM objects, which can then be distributed to work with whatever applications that need them (or distributed to other researchers that might need to use or test the objects). "Rather than write large monolithic object-oriented applications, you can write applications as small independent components that can slot together to make a complete application. With a little extra work, your C++ objects can become COM objects. As COM objects, they are not as tightly tied to one running process or computer as a conventional C++ object would be" (Bates 1999).

The other advantage of developing the manufacturing (control) entities as COM objects is that some large industrial vendors such as GE Industrial System and Sisco, Inc. already have the automation and control products that support the COM/DCOM technology. Therefore, by using the COM/DCOM approach, we can close the gap between the academic field and the manufacturing industry, and can also minimize the logical (software) control model's development lead times, by facilitating the process of shifting from the design phase to the implementation phase.

CHAPTER 8

CONCLUSION AND FURTHER RESEARCH

In this chapter, we will first provide a summary of the work carried out in this research, as well as a discussion of the results in the context of the general research objectives. Next we will discuss the contributions of this study and finally, provide a brief discussion of further research possibilities.

8.1 Summary

In Chapter 4, to design a distributed scheduling and control system, instead of using the 'top-down' approach to determine the control agents first and then structure the scheduling and control algorithms around these agents, we used the object-oriented analysis and design approach to struct‌ure the scheduling algorithm first. After identifying the roles (objects) that are involved in the scheduling processes, we identified the possible control agent candidates from these roles, and control responsibilities were then added to some of these control agents accordingly.

The use of the object-oriented methodology to decompose the control algorithms can help decouple the software control model from any preconceived control structure. For instance, after we have developed the logical scheduling model, we can implement the scheduling software solution in a single (control) processor to implement the centralized scheduling scheme. Or as we have done in Chapter 4, we can identify some possible control agent candidates from the logical scheduling model, and implement the scheduling solution in a distributed scheduling and control structure. Moreover, using the object-oriented approach to decompose the control algorithms can allow us to explore a broader set of possible control agent candidates. "While some entities may prove unnecessary, it's easier to cast the net broadly and leave some as stubs than to build an architecture into which omitted entities cannot easily be added later" (Parunak et al. 1998a).

In Chapter 5, experiments were conducted to clarify the confusing concepts regarding distributed scheduling and real-time distributed (dispatching) control, and to identify the role of the 'control' mechanism in the control systems that use the distributed scheduling approach to perform pre-production scheduling. The experimental results have shown that while implementing the distributed scheduling algorithm for real-time production control can enhance the control system's flexibility against disturbances, sometimes it is hard to predict the behavior and performance of such control systems. On the other hand, in control systems that use the distributed scheduling algorithm to generate (predictive) pre-production schedules, although the distribute scheduling approach can help enhance the scheduling performance through parallel computing (Dilts et al. 1991), the experimental results have shown that in such control systems, it is still important that the control agents should have the proper local control mechanisms to react to disturbances in real time, so that the control system's adaptability against disturbances can be enhanced.

Control issues are usually ignored in the current research that discusses the distributed scheduling and control approaches. But referring to our experimental results, we can see that different control algorithms can have various impacts on the control system's performance and communication requirements. For instance, in control systems wherein the control agents do not have the proper local control mechanisms to react to disturbances in real time, when disturbances happen, either the control agents will just wait for the disturbances to pass (the WAIT approach), or a rescheduling process will be invoked (the NO_WAIT approach). In the former case, the system performance will be significantly degraded. And in the latter case, the communications between the control agents will be increased due to the high rescheduling frequencies (especially in the stochastic manufacturing environments).

Alternatively, in control systems wherein the control agents have the proper local control mechanisms to react to disturbances in real time (the WAIT_INTEL approach), not only can the desirable system performance be achieved, but the communications between the control agents can also be minimized.

Most of the research on multi-agent heterarchical control systems only deals with dispatching the routing decisions, and ignores the job sequencing issues. As a result, most of these control systems performed scheduling by jobs on a First-Come-First-Serve basis. In Chapter 6, we have conducted experiments to investigate and test the impact of the job routing and job sequencing control mechanisms on the control system's performance. The experimental results have shown that while dispatching the routing decisions can help enhance the control system's flexibility and adaptability against disturbances, it is important to incorporate the job sequencing control mechanism in the control system so that certain global performance objectives can be achieved.

The experimental results show that, in the control system wherein the job agents will exploit the routing flexibility, when the production volume is low, the opportunistic behavior of the jobs agents can help improve the system performance (by helping the jobs find a 'shortest-time path' through the production system), and enhance the system's adaptability against disturbances (by helping the jobs avoid the bottleneck 'down' resources). But as the production volume increases, the production system has less routing flexibility, and the job sequencing control mechanism becomes more important (especially in production systems with high production volume). This is because the job sequencing control mechanism can help enforce the cooperative behavior of the job agents and ensure that certain global performance objectives can be achieved. That is, while in contention for the use of certain resources, based on the global performance objectives and the dispatching priority rules adopted, jobs with lower priority 'have to' let jobs with higher priority to use the resources first.

As expected, the control system that implements both the job routing and job sequencing control mechanisms (the AUC + JSEQ control approach) can best improve the control system's performance and adaptability against disturbances. Although in the AUC + JSEQ control approach, having resource agents use a dispatching rule to sequence the jobs can violate the commitments between the resource agents and some of the jobs.

Our experimental results have shown that this did not affect the system performance much (compared to the COMT + AUC + JSEQ control approach, wherein the job agents will be notified whenever their 'quoted start time' is changed so that they can explore other routing opportunities). Therefore, in control system that implements the job sequencing and job routing control mechanisms, it is sometimes justifiable to compromise some of the commitments between the workstation agents and the job agents in order to achieve certain global performance objectives and minimize the communications between the control agents (compared to the COMT + AUC + JSEQ control approach).

As some researchers have proposed that control algorithms should be compared on a common testbed, it will be helpful if researchers can built their control modules as some platform-independent software components that can be easily distributed across the network and integrated into some other control systems. In Chapter 7, we have shown that by using the COM/DCOM technology to build some control modules, we can easily distributed them across the network to implement the simulated distributed shop floor control system. By using the COM/DCOM technology to implement the control algorithms, this provides an opportunity for researchers to explore the possibility of developing some control modules with standardized interface, so that these control modules can easily be distributed across the network. This allows researchers to easily integrate some of their peers' work into their own control model to test or validate some of the proposed control modules.

8.2 Contributions

Through the work in this thesis, it is believed that this study can contribute to the research in manufacturing system control in the following areas:

- To provide some insights regarding the decomposition approaches for various control methodologies. The work in Chapters 2 and 4 has shown that the use of the object-oriented analysis and design approach to structure control algorithms can help decouple the software control model from any preconceived control structure. Moreover, this approach can allow us to explore a broader set of possible control agent candidates when designing a distributed control system.

- In Chapter 5, experiments were conducted to clarify the confusing concepts in current research regarding the 'distributed scheduling' and 'real-time distributed control'. Due to the confusion of these concepts, many researchers have ignored the control issues while discussing the distributed scheduling and control systems. In this study, we have identified the role and importance of the 'control' algorithm in the control systems that using the distributed scheduling approach to perform pre-production scheduling.

- Most work in the multi-agent heterarchical control system only deals with dispatching the routing decisions, and ignores the job sequencing control. As a result, most of the proposed control approaches performed scheduling by jobs on a First-Come-First-Serve basis. In this study, we have given some insights regarding how the job routing and job sequencing control mechanisms can affect the control system's performance in various manufacturing environments.

- In Chapter 7, we have shown that by using the COM/DCOM technology to build control modules, we can easily distributed them across the network to implement the simulated distributed shop floor control system. This provides researchers an opportunity to explore the possibilities of enhancing the collaborations between each other by building some platform independent software that can be easily distributed to and evaluated by other researchers.

8.3 Further Research Directions

In this study, we have shown that using the object-oriented methodology to decompose the control algorithms can help decouple the software control model from any preconceived control structures. This allows researchers to implement certain control algorithms in various control forms, and an objective comparison of the alternative control methodologies can then be made. In Chapters 5 and 6, experiments were conducted to evaluate the performance of various control methodologies in different manufacturing environments. Although the experiments conducted here do not represent an exhaustive evaluation of the alternative control forms or algorithms for a multi-agent control system, experimental results show that the performance of a control approach can be affected by the characteristics of a manufacturing environment. This shares similar views with the work of some other researchers in manufacturing control. For instance, in (Brennan 1996), it has mentioned that:

> "The hypothesis concerning the best choice of control architecture is that the 'best' choice of control architecture is a function of the controlled system's characteristics".

Therefore in future research, while proposing or evaluating alternative control approaches, researchers not only should identify the characteristics of the manufacturing environment wherein a control approach is implemented, but efforts should also be made to identify the parameters that might affect the validity or the performance of the alternative control approaches. This can help open the opportunity for the development of some intelligent control systems, wherein control agents will be able to select the appropriate control algorithms to use in real time based on their knowledge of the current status of the manufacturing environment.

The use of the object-oriented methodology can help facilitate the development and evaluation of alternative control approaches. For instance, as discussed in this study, the logical software objects can be organized into different control modules to implement/evaluate various control forms. The abstraction and encapsulation properties of the object models allow us to modify the implementations of a control agent's control logic/method easily, and the modifications will be transparent to the other control agents that interact with it (as long as the communication interface is not changed). And as discussed in Chapter 7, by using the COM/DCOM technology to build control modules, we can easily distribute them across the network. With such an approach, the collaborations between researchers can be enhanced in a way that researchers can easily integrate each other's proposed/developed COM control modules into their own control models to test or validate the alternative control algorithms. But if the COM/DCOM approach is to be adopted by researchers to develop the software control modules, further research needs to be done in the area of developing some standards (or design patterns) for designing the COM object interfaces in the context of manufacturing control systems.

Finally, in future research, when modeling different control approaches, efforts should also be made to investigate what control resources are currently available in industry, and what are the computation limitations, reliability and cost of these resources. As by incorporating these factors into the modeling and evaluation of the alternative control approaches can make the research results be more realistic. As well, it can facilitate the shifting from the academic practice into the industrial practice, as the analysis of the justification of the expense and risk of installing the alternative control systems can be incorporated into the research results.

REFERENCES:

1) Baker, A. D., "A Survey of Factory Control Algorithms which Can be Implemented in a Multi-Agent Heterarchy", Journal of Manufacturing Systems, April 9, 1997.

2) Bates, Jonathan, "Creating Lightweight Components with ATL", SAMS, 1999.

3) Bauer, Bowden, Browne, Duggan and Lyons, "Shop Floor Control Systems- From design to implementation", Chapman & Hall, 1994,

4) Bongaerts, L. Monostori, D. McFarlane, B. Kadar, "Hierarchy in distributed shop floor control", accepted for IMS-EUROPE 1998, the First Open Workshop of the Esprit Working group on IMS, Lausanne 15-17 April 1998.

5) Bonagerts, L., "Integration of Scheduling and Control in Holonic Manufacturing Systems", (98D11), http://www. mech.kuleuven.ac.be/~lbongaer/doc/abstr.html, doctoral dessertation, 1998.

6) Booch, Grady, "Object-Oriented Analysis and Design with Applications", 2nd Edition, Addison-Wesley, 1994.

7) Brennan, R. W., "Appropriate Control Architecture for Automated Manufacturing Systems", the doctoral dissertation, Department of Mechanical Engineering of the University of Calgary, December 1996.

8) Chung, P. E., Huang, Y., Yajnik S., Liang, D., Shih, J., Wang, C. Y., Wang, Y. M., "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer", http://www.cs.wustl.edu/~schmidt/submit/Paper.html, 1997.

9) Cavalieri, S., Luc Bongaerts, Marco Macchi, Marco Taisch, Jo Wyns, "A Benchmark Framework for Manufacturing Control", Proc. of the Second International Workshop on Intelligent Manufacturing Systems, Leuven, September 22-24, 1999.

10) Conway, R. W., Maxwell, W. L., and Miller, L. W., "Theory of Scheduling", Addison-Wesley, 1967.

11) Dilts, D. M., N. P. Boyd, H. H. Whorms, "The evolution of control architectures for automated manufacturing systems", Journal of Manufacturing Systems, Vol. 10, No. 1, pp. 79-93, 1991.

12) Duffie, N. A., R.S. Piper, B. J. Humphrey and J. P. Hartwick Jr., "Hierarchical and non-hierarchical manufacturing cell control with dynamic part-oriented scheduling", Proceedings of NAMRC-XIV, 1-4, 1986.

13) Duffie, N. A., V. V. Prabhu, "Real-time distributed scheduling of heterarchical manufacturing systems", Journal of Manufacturing Systems, Vol. 13, No. 2, pp. 94-107, 1994.

14) French, S., "Sequencing and Scheduling: An introduction to the Mathematics of the Job-Shop", Ellis Horwood, 1990.

15) Larman, G., "Applying UML And Patterns, An Introduction To Object-Oriented Analysis and Design", Prentice Hall, 1997.

16) Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G., 1992, "Object-oriented Software Engineering", Workingham, England, Addison-Wesley, p. viii.

17) Microsoft Developer Network (MSDN) Library, April 1998.

18) Maturana, F., and Norrie, D. H., "Multi-Agent Mediator Architecture for Distributed Manufacturing", Journal of Intelligent Manufacturing, v.7, pp. 257-270, 1996.

19) Parunak, H. V. D., "Manufacturing Experience with the Contract Net", In M. N. Huhns, ed., Distributed Artificially Intelligence, Pitman, 285-310, 1987.

20) Parunak, H. V. D., "Autonomous Agent Architectures: A Non-Technical Introduction", http://www.erim.org/~van/nontech.pdf, 1993.

21) Parunak, H. V. D., "Applications of distributed artificially intelligence in industry", Industrial Technology Institute, 1994.

22) Parunak, H. V. D., "Case Grammar: A Linguistic Tool for Engineering Agnet-Based Systems", ITI Technical Memorandum, http://www.iti.org/~van/casegram.ps, Industrial Technology Institute, Ann Arbor, 1995.

23) Parunak, H. V. D., "Case Grammar: A Linguistic Principles from Natural Agent Systems", Annals of Operations Research, 1998.

24) Parunak, H. V. D., John, S., and Steve, C., "Toward the Specification and Design of Industrial Synthesis Ecosystems", the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL 1997).

25) Parunak, H. V. D., "What can agents do in industry, and why? An overview of industrially-oriented R&D at CEC", CIA, 1998A.

26) Parunak, H. V. D., Baker, A., Clark, S., "The AARIA Agent Architecture: From Manufacturing Requirements to Agent-Based System Design", Workshop on Agent-Based Manufacturing, ICAA 1998b.

27) Ramaswamy, S. E., Joshi, S. B., "Distributed Control of Automated Manufacturing Systems", in Proceedings of 27th CIRP International Seminar on Manufacturing Systems Proceedings, Ann Arbor, MI, May 21-23, 1995.

28) Rubin, k., and Goldberg, A., "Object Behavior Analysis", Communications of the ACM, vol. 35(9), p. 48, September 1992.

29) Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., "Object-oriented Modeling and Design", Englewood Cliffs: Prentice Hall, 1991.

30) Saad, A., G. Biswas, K. Kawamura, E. M. Johnson, "Effectiveness of dynamic rescheduling in agent-based flexible manufacturing systems", SPIE Vol. 3203, pp. 88-99,1997.

31) Shen, W., Norrie, D.H., and Kremer, R., (1999) " Developing Intelligent Manufacturing Systems Using Collaborative Agents", Proc. of the 2nd International Workshop on Intelligent Manufacturing Systems, Leuven, Belgium, pp. 157-166, September 22-24, 1999.

32) Shen L., Brennan, R. W., Norrie, D. H., "Agent classification in manufacturing systems", the IASTED International Conference on Artificial Intelligence and Soft Computing (ASC'2000), Banff, Canada, July 24-26, 2000.

33) Simpson, J. A., R. J. Hocken, J. S. Albus, "The automated manufacturing research facility of the National Bureau of Standards", Journal of Manufacturing Systems, Vol. 1, No. 1, 1982, pp. 18-31.

34) Sing Li, Panos Economopoulos, "Professional COM Applications with ATL", Wrox Press Ltd., 1998.

35) Sipper, D., R. L. Bulfin, Jr., "Production planning, control, and integration", McGraw-Hill, 1997.

36) Smith, R. G., "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", IEEE Transactions on Computers, vol. c-29, No. 12, December 1980.

37) Sousa, P., Carlos Ramos, "Proposal of a Scheduling Holon for Manufacturing", Proceedings of the second International Conference and Exhibition on The Practical Application of Agents and Multi-Agent Technology (PAAM'97), London, pp. 255-268, UK, 21st to 23rd April 1997.

38) Valckenaers, P., H. V. Brussel, L. Bongaerts, J. Wyns, "Holonic Manufacturing Systems", Integrated Computer Aided Engineering, Vol. 4: 191-201, 1997a.

39) Valckenaers, P., H. V. Brussel, L. Bongaerts, P. Peeters, "Reactive Scheduling in Holonic Manufacturing Systems: Architecture, Dynamic Model and Co-operation Strategy", http://www.mech.kuleuven.ac.be/~lbongaer/ref/97p60.html, 1997b.

40) Van Brussel, H., Jo Wyns, Paul Valckenaers, Luc Bongaerts, Patrick Peeters, "Reference Architecture for Holonic Manufacturing Systems: PROSA", Computers in Industry 37, pp. 255-274, 1998.

41) Veeramani, D. and Wang, K. J., "A Flexible Auction-based Shopfloor Control Paradigm for Highly-Distributed Manufacturing Systems",1998.

42) Voris, W., "Production Control Text and Cases", Richard D. Irwin Inc., third edition, 1966.

43) Wooldridge, M. J., and Jennings, N. R., "Software Engineering with Agents: Pitfalls & Pratfalls", IEEE Internet Computing, May/June 1999, pp. 20-27.

44) Zhang, X., Norrie, D. H., "Holonic Control at the Production and Controller Level", Proc. of the 2nd International Workshop on Intelligent Manufacturing Systems, Leuven, Belgium, pp. 215-224, September 22-24, 1999.