Component-based Intelligent Control Architecture for Reconfigurable Manufacturing Systems

Jiancheng Su

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Industrial and Systems Engineering

(Alphabetically)
Dr. F. Frank Chen, Chair
Dr. Philip Y. Huang
Dr. Subhash C. Sarin
Dr. Robert H. Sturges, Jr.

November 13, 2007
Blacksburg, Virginia

Keywords: Reconfigurable Manufacturing Systems, System-level Control, Component-based Intelligent Control Architecture

Component-based Intelligent Control Architecture for Reconfigurable Manufacturing Systems (RMS)

Jiancheng Su

ABSTRACT

The present dynamic manufacturing environment has been characterized by a greater variety of products, shorter life-cycles of products and rapid introduction of new technologies, etc. Recently, a new manufacturing paradigm, i.e. Reconfigurable Manufacturing Systems (RMS), has emerged to address such challenging issues.

RMSs are able to adapt themselves to new business conditions timely and economically with a modular design of hardware/software system. Although a lot of research has been conducted in areas related to RMS, very few studies on system-level control for RMS have been reported in literature. However, the rigidity of current manufacturing systems is mainly from their monolithic design of control systems. Some new developments in Information Technology (IT) bring new opportunities to overcome the inflexibility that shadowed control systems for years.

Component-based software development gains its popularity in 1990's. However, some well-known drawbacks, such as complexity and poor real-time features counteract its advantages in developing reconfigurable control system. New emerging Extensible Markup Language (XML) and Web Services, which are based on non-proprietary format, can eliminate the interoperability problems that traditional software technologies are incompetent to accomplish. Another new development in IT that affects the manufacturing sector is the advent of agent technology. The characteristics of agent-based systems include autonomous, cooperative, extendible nature that can be advantageous in different shop floor activities.

This dissertation presents an innovative control architecture, entitled Component-based Intelligent Control Architecture (CICA), designed for system-level control of RMS. Software components and open-standard integration technologies together are able to provide a reconfigurable software structure, whereas agent-based paradigm can add the

reconfigurability into the control logic of CICA. Since an agent-based system cannot guarantee the best global performance, agents in the reference architecture are used to be exception handlers. Some widely neglected problems associated with agent-based system such as communication load and local interest conflicts are also studied. The experimental results reveal the advantage of new agent-based decision making system over the existing methodologies. The proposed control system provides the reconfigurability that lacks in current manufacturing control systems. The CICA control architecture is promising to bring the flexibility in manufacturing systems based on experimental tests performed.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Prof. F. Frank Chen, who helped me all the way and made this research possible. During the period of time I have been working under his guidance, I learned so much from his independent and critical way of thinking in academic research. Professor Chen has been always supportive and generous to help me overcome stressful situations. All the things I learned during the journey in pursuing the doctoral degree will definitely benefit me in the rest of my life.

I also want to show my great sincere appreciation to my committee members: Prof. Sarin Sahbash, Prof. Robert Sturges and Prof. Philip Huang. They helped me shape this research into it final appearance. They all give me valuable suggestions on different aspects of this research. Professor Sarin is an expert in manufacturing systems, especially, scheduling aspects. He has been always very kind, and willing to help. The mathematics I learned from his class form the fundamentals of the analysis work. Prof. Sturges is always full of original ideas. The discussions we had are always stimulating. The comments he gave me in my research are very enlightening and constructive. Prof. Huang is a very knowledgeable professor in both engineering and management areas. His expertise on management side provides creative views on the problems I am working on.

There are also several other professors that I took classes from or had discussions with. Without all these training at Virginia Tech, I can't finish the research work. Although I cannot make a full list of all the professors, I definitely will never forget them and their help. My friends who work in the FMS lab at Virginia Tech with me also gave me great help. The discussions I had with Hungda Wan, Radu Babiceanu, Leonardo Rivera, and Guorong Huang are always helpful and informative.

Finally, I also want to thank my parents who are always supportive of me in my life, and my girlfriend, Ying, who gave me a lot of encouragement when I encountered difficulties.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1 Introduction to Reconfigurable Manufacturing Systems (RMS)

1.1.1 Background information

Changing manufacturing environment is now characterized by aggressive competition on a global scale and rapid changes in process technologies. In order to stay competitive in the market, new manufacturing systems need to be fully and rapidly responsive to different variations by improving its flexibility and agility while maintaining the level of productivity and quality.

A new manufacturing paradigm called reconfigurable manufacturing systems (RMS) has been emerged to address the needs raised by the rapidly changing markets and introduction of new technologies (Koren *et al.* 1999). An RMS is designed for rapid adjustment of production capacity and functionality, in response to new circumstances, by rearrangement or change of its components. These new systems can provide "exact functionality that is needed exactly when it is needed" (Mehrabi *et al.* 2000b). The reconfigurable manufacturing system is increasingly recognized today as a requirement for industrial growth in a global economy.

The National Research Council, in a study entitled "Visionary Manufacturing Challenges for 2020", identified Reconfigurable Manufacturing Systems as a high priority technology in future manufacturing (NRC 1998). The study also mentions the Reconfigurable Manufacturing Enterprise as one of the Six Grand Challenges for the future of manufacturing.

1.1.2 Evolution of manufacturing paradigms

A manufacturing paradigm, according to Merriam-Webster dictionary, can be defined as a philosophical and theoretical manufacturing framework of a scientific school or discipline within which theories, principles, laws, generalizations and experiments are formulated.

From the first industrial revolution, the manufacturing environment is continuously changing in order to adapt the customer demands, advances in automation technologies and economical trends. During the last century, several manufacturing paradigms have been introduced that aims to bring more competitiveness to manufacturing enterprises.

- Dedicated manufacturing systems

Historically, the first manufacturing paradigm introduced at the beginning of the last century is mass production developed by Henry Ford. The revolutionary concept of mass production is characterized by the production of the same product that consists of identical interchangeable parts in large scale using a rigid assembly line. Dedicated Manufacturing Systems (DMS) are used to produced large amounts of parts at high level of quality and low cost.

The mass production model requires stability and control in the input variables and markets. When these parameters became less stable, the mass production is incapable to treat variations. This rigid structure of DMS is the main obstacle in using mass production models. The mass manufacturing became viable only for some products in a global market characterized by uncertainty.

- Cellular manufacturing systems

Cellular manufacturing system (CMS) is another manufacturing paradigm aimed to improve productivity. A cell is a group of workstations, machines or equipment arranged such that a product can be processed progressively from one workstation to another without waiting for a batch to be completed. Cells may be dedicated to a process, a subcomponent, or an entire product. Normally, parts that belong to the same family advance from raw material to finished parts within a single cell. Group technology developed in the 1970's is often used in cellular design. Group technology is the process of studying a large population of different parts, and then grouping them into logical families with similar characteristics so that they can be produced by the same group of machines, tooling, and people with only minor changes on procedure or set-up.

The CMS is designed for a fixed set of part families and its orientation towards stability of demand and long life cycles. Therefore, increased product variety is difficult

to be achieved, since CMS is structurally inflexible and the redesign of the cells or changing the shop floor layout is too costly. Subsequently, CMS is not suitable for production systems with fluctuations in volume.

- Flexible manufacturing systems

Flexible manufacturing systems (FMS) are widely accepted and developed in industry in the last two decades. FMS is defined as "a computer-controlled production system capable of processing a variety of part types." The main components of FMS are computer numerically controlled (CNC) manufacturing machines, tools to operate CNC machines, robots, and automated material handling equipments. Although FMS was introduced with great expectations, a study conducted by Mehrabi *et al.* (2002) claims that two-thirds of the practitioners did not believe that FMS is living up to its promise.

Although FMS focuses on flexibility, their hardware and software are predetermined and fixed. This implies that FMS is not adequately responsive to change, as its capabilities in terms of upgrading, add-ons, and customization are limited. Moreover, FMS was built for low or medium volume productivity, so they are not suitable for large market fluctuations. The new paradigm required by today's manufacturing should incorporate the advantages of FMS but need to be simpler, responsive, and less expensive. The Reconfigurable Manufacturing Systems (RMS) paradigm attempts to satisfy these requirements and avoid the shortcomings of the conventional manufacturing philosophies.

- The need for reconfigurable manufacturing systems.

Today's manufacturing systems should enable flexibility in capacity, responsiveness in market changes, product variety, adoption and utilization of new technologies, and general scalability, in a cost effective manner. The emergence of RMS concept was largely due to the limited capabilities of FMS (Ayres 1992; Ito 1988; Mansfield 1993). FMS addresses changes in work orders, production schedules, part-programs, and tooling for production of a family of parts. But in terms of design, an FMS uses a fixed set of hardware/software, which hardly allows changes to be made. Owing to the inflexibility nature of FMS system structure, manufacturers normally purchase FMS

with excess capacity and excess features (some of those have never been used), which lead to waste of capital investment. The RMS concept that promotes "exact functionality and capacity needed, exactly when needed" is introduced to overcome the shortcomings of FMS. A survey conducted by Mehrabi *et al.* (2002) concluded that experts also have the same opinion that RMS is a desirable next step in the evolution of production systems.

The aim of RMS is to design systems, machines, and controls for cost-effective, rapid responsive to changes in market demand and products. Methodologies for the systematic design and rapid ramp-up of RMS are the cornerstones of this new manufacturing paradigm. The barrier in developing reconfigurable systems is the lack of a systematic approach in designing optimal, scalable configurations of systems that include reconfigurable hardware and reconfigurable software.

1.1.3 Concept of Reconfigurable manufacturing systems

Dedicated manufacturing lines (DML) typically have high capacity but limited functionality. They are cost effective to produce a few part types, but when market demand changes, the dedicated lines cannot operate at their full capacity and may even become totally obsolete. Flexible manufacturing systems (FMS), on the other hand, are built with full flexibility, however, usually with functionalities that may not be needed at installation time. Such kind of wastes can be eliminated with RMS technology. The key feature of RMS is that, unlike a DML and an FMS, its capacity and functionality are not fixed. A given RMS configuration can be dedicated or flexible, or in between, and can be changed as required. With different granularity of the RMS modules, RMS can evolve from either DML or FMS by adding or removing capacity/functionality.

The RMS is designed through the use of reconfigurable hardware and software, such that its capacity and/or functionality can be changed over time. An RMS goes beyond the economic objectives of FMS by permitting: (1) reduction of lead time for launching new systems and reconfiguring existing systems, and (2) the rapid manufacturing modification and quick integration of new technology and/or new functions into existing systems.

As reported by Garro and Martin (1993) and Lee (1997), underlying components and structure of a manufacturing system significantly affect its ability to be reconfigured

for rapid and cost effective production of new products. In order to be flexible in producing a variety of products and changing the system itself, RMS should be open-ended, using modular hardware/software structure such that they can be improved and upgraded rather than replaced. RMS must be designed at the outset to be reconfigurable, and must be created from basic hardware and software modules that can be arranged quickly and reliably.

An RMS must have certain key characteristics such as modularity, integrability, convertibility, diagnosability, and customization (Mehrabi *et al.* 2000).

1.2 Control software for RMS

Control software plays an important role in modern manufacturing systems for its well-known effects on cost and implementation time. In the reported survey of experts in manufacturing conducted by NSF Engineering Research Center (ERC) for Reconfigurable Machining Systems at the University of Michigan (Mehrabi *et al.* 2002), software issues represented the greatest concern for the successful development of RMS technology. The control software of RMS needs to be adapted quickly to new requirements without affecting and interrupting the performance of manufacturing systems. Conventional control methods are unable to fulfill the requirement due to its rigid structure.

Manufacturing control is required at different levels for an automated production system: low and high level (Figure 1.1). At the low-level, the automation devices, such as industrial robots and NC machines, require control techniques that regulate its behavior according to a specific objective. At this level, time-based control techniques such as Proportional, Derivative, Integral and On/Off techniques, which are commonly used either alone or in some combination like the well known PID (Proportional, Integrative and Derivative), and even intelligent control techniques, such as fuzzy logic, are often used to design and implement appropriate control algorithms on digital or analog devices. Since, control functions present at this level are not related to our research, hence are not discussed further in this dissertation.

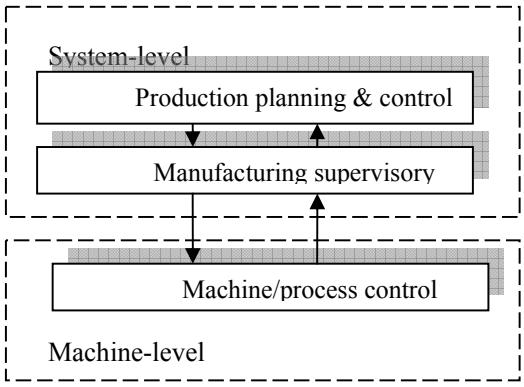


Figure 1.1 System-level control vs. Machine-level control

The high-level control is concerned to coordinate the manufacturing resource activities that aim to produce the desired products, such as in the case of flexible manufacturing control systems. The application of different stochastic and nature-inspired algorithms at this level are used to decide what to produce, how much to produce, when production is to be finished, how and when to use the resources or make them available, when to release jobs into the factory, which jobs to release, job routing, and job/operation sequencing.

Research to date on control for RMS mainly focuses on design of open-architecture control of machines. Traditional machines in manufacturing systems were built as heterogeneous, device-oriented systems with proprietary software components. The tight coupling of software and hardware leads to very complex and inflexible systems. Open-architecture control (OAC) is considered as the key enabler for the realization of modular and re-configurable machining systems. OAC is aimed to provide a vendor-neutral, standardized environment based on defined open interface and software platform. Since early 1990's, several international initiatives have worked on concepts of OAC (Katz *et al.* 2000). Compared with the low-level physical control of machines, the RMS system-level control mainly considers the higher-level planning and scheduling aspects of an integrated manufacturing system. Currently, quite limited efforts were made on open architecture design for manufacturing system level control for RMS.

1.3 Research motivations, objectives and contributions

1.3.1 Research motivations

For RMS, it is highly desirable that the control software should have a modular structure and be "open" such that upgrading and customization of the system is practical when system is reconfigured. Today's industrial control systems are dominated by proprietary solutions. Control software is usually custom written for the given system configuration and such designs are often inflexible due to their rigid structures. The resulting problems are high development costs and long innovation cycles. The manufacturing control systems are required to adopt the latest development in information technology to increase its effectiveness.

Object-oriented programming (OOP) is the most commonly used software technology in design of manufacturing control (Chan *et al.* 2000; Grabot and Huguet 1996; Howard *et al.* 1998). OOP is also considered as a desirable choice for RMS control software (Mehrabi *et al.* 2002). In reality, OOP had promised much greater code reuse, but largely unfulfilled. Compared with OOP, component-based software technology is initially designed to build modular software (with components). Earlier component-based software platforms suffered from complexity, large software size, and usually they do not support real-time features well. These drawbacks stalled their applications in industry. However, the growing maturity of component-based software technology has gained its popularity and has made easy development and integration of modular software possible.

The current software technologies such as CORBA, J2EE and .NET provide a modular framework for software development. Nonetheless, the modularity is highly vendor-dependent, i.e. control software can only integrate with components from the same platform. In recent years, emerging open standards-based integration technologies such as Web services and XML allow pieces of software to communicate without the interoperability problems that traditional software technologies have. Extensible Markup Language (XML) is a cross-platform, extensible, and text-based standard for representing data. Since XML data is stored in plain text format, it provides a software and hardware independent way of sharing data. Web Services is a new integration technology that allows interactions between applications on different systems. The web service protocols

and XML-based are not tied to any particular operating system or programming language. As such, these new technologies can be used as backbone for developing open, standards-based, and vendor-neutral reconfigurable system-level control framework.

Based on open XML data format and open protocols, the control architecture proposed in this research has a higher-level of abstraction than current majority of component-based control software studies. The architecture does not specify how software components are implemented and imposes no restriction on how components might be combined. Each component is a black box and exposes only services to outside. Components will be loosely coupled and they are connected by open data format. The services only communicate with each of them by platform-neutral, standardized XML format messages.

In a component-based software system, the specification of the components is more important, from the assembly perspective, as compared to the realization of specification is realized (Gorton and Liu 2004). In contrast to traditional development, implementation has given way to integration as the focus of system construction (Zhu and Wang 2003). Right component specifications have been considered as a critical factor in major activities of CBSD, such as component qualification, and component adaptation. The robustness and flexibility of architectures can be produced only when components are clearly identified and specified.

The component-based software architecture provides reconfigurable software architectures for RMS control systems, which is able to adapt to physical changes in manufacturing systems. The design of control logic needs to be reconfigurable too when it faces a dynamic manufacturing environment. Agent technologies are famous for dealing with dynamic variations or in general to unexpected events. Moreover, agent-based systems are flexible and adaptive, since they may be added or replaced easily. Therefore, the system configuration can be continuously changed to accommodate changing requirements.

The agent paradigm allows solving distributed problems quite elegantly (Ferber 1999; Kendall 2000). Unlike static programs that rely on pre-assumptions on the environment or their cooperation partners, agents are considered individuals who are able to react appropriately to change. The characteristics of an agent include autonomous,

reactive, cooperative, modular, and a multi-agent system is known for its extensibility, flexibility and reconfigurability. All these have proven great advantages for their use in a shop floor control system (Berbers *et al.* 2002a).

Although agent-based systems have many desirable features, there are a few shortcomings, such as, it is usually impossible for agent-based systems to seek global optimization point and system performance is usually unpredictable. Another potential problem is the effects of large number of agents and excessive communications. They can significantly deteriorate system performances. Because of these issues, in the control architecture proposed in this research, agents are activated only when exceptions occur. Compared to a pure agent-based scheduling system, such exception handling mechanism avoids the risk of missing the global optimal point and lessen system burden when system is under normal condition.

1.3.2 Research objectives

To deal with the increasing challenges faced by nowadays' manufacturing enterprises, development of reconfigurable control systems is considered as important as the development of reconfigurable hardware in manufacturing systems. However, compared with the studies in the modular design of machine-level hardware/control, system-level manufacturing control with reconfigurable features can hardly be found in the literature. The proposed research aims to study reference control architecture that has a modular architecture using newly emerging information technologies. Control systems based on this reference architecture should be robust and flexible and it is able to adapt to changes inside/outside of manufacturing systems.

The key features of control systems based on this reference architecture should include:

- Reconfigurability: Able to adapt the manufacturing system to new circumstances through rearrangement/change of the components.
- Reusability: Software components can be reused over different manufacturing demand and configurations.
- Robustness: able to maintain system operability in the face of large and small malfunctions.

- Disturbance handling: Able to provide faster recognition and response to disturbances such as machine malfunctions, rush orders, unpredictable process yield, human errors, etc.
- Extensibility: Able to permit "plug and play" of software component and it can be extended to intra/inter enterprise level by including or integrating with more manufacturing functions.

1.3.3 Contributions

A new reference control framework for system-level control of RMS is studied in this dissertation research. It includes the necessary background information and a comprehensive state-of-the-art literature review of studies on control systems for RMS. Based on the review, very limited research on system-level reconfigurable control design has been observed. This research is conducted to explore an innovative yet practical way to design reconfigurable control software for RMS utilizing new information technologies.

As compared to other existing control architectures, the proposed control architecture is reconfigurable in both software architecture and software logic. The architecture is built with loosely coupled components, which are connected by open data format. The control architecture has a higher-level of abstraction than current majority of component-based control software studies, in which it does not specify how software components are implemented and imposes no restriction on how components are combined. Components can be obtained from different vendors using different programming languages as long as the specifications match the requirements. The specifications that clearly identify and specify components are defined. The benefits of a component-based framework can only be achieved when components are properly specified. This framework can also be easily extended for use in inter/intra enterprise level (Su and Chen 2005).

Agent technologies are used to handle exceptions in system. Since decisions made by agent-based systems rely on their interaction among agents, they are very flexible and reconfigurable, and more suitable to deal with a dynamic manufacturing environment as compared to traditional centralized methodologies. A new stepwise decision process is

designed for the scheduling agents. A few shortcomings of previous studies are communication burden and local interest conflict among agents. These issues are analyzed and resolved in Chapter 5 of this dissertation. The results of simulation show the proposed agent-based methodology is very efficient in handling different exceptions and has better output performance than currently popular algorithms.

This research is an original study in design of reconfigurable control systems. The result of this research is expected to increase the capability of manufacturing control systems in adapting to dynamic environment. The reference architecture developed and tested in this research is considered as a part of the critical effort in developing the foundations of the next generation manufacturing systems.

1.4 Research scope

Manufacturing systems are continuously facing rapid changes in the environment. RMS is able to handle changes and disturbances much better than current systems. This research contributes a new paradigm by developing reference architecture for RMS system-level computer control. A reference architecture is defined by Wyns (1999) as "the specification of a generic solution for a kind of problems in a domain." It can be seen as a style or method being used to tackle a certain type of problems.

Since the topic of the study is generally very broad, some aspects that are not closely related to this research are not included. Although new information technologies are used in developing the new control architecture, this research is not intended to provide a development manual of these new information technologies. Instead, this research focuses on the completeness of the concept of the new framework. Some technical details, such as, the communication between manufacturing equipments (machine-level control) and the host computer (system-level control) are not within the scope of this research. When using agent systems to handle exceptions, details related to exceptions are not studied but rely on results of literature from other researchers.

1.5 Definition of terminology

This section defines key terminology used in the context of this dissertation. The existing literature fails to offer a clear set of definitions, and different authors may use the same terminology for different meanings. For clarification, terms related to this research are defined as follows:

Manufacturing control: Manufacturing control is an extremely broad topic, ranging from long-range strategic management of the manufacturing facility to real-time management of sensors/actuators. It may consider activities across several shops, or even several plants. It includes purchasing, material requirements planning, design, process planning, etc. Alternative terminology used for manufacturing control system is manufacturing planning and control system (MPCS).

Shop floor control: Shop floor control is the group of activities responsible for managing the transformation of orders into outputs in one shop. It mainly contains the short-term scheduling, execution, and monitoring and resource allocation activities, etc. Different terms for Shop floor control include manufacturing activity planning (MAP), production activity control (PAC), and manufacturing Execution System (MES).

System-level manufacturing control: The manufacturing control problem can be considered at two levels: machine (low) and system (high) level. At the machine level, the individual manufacturing resource is controlled to process product units according to the system level control decisions. System level manufacturing control can be viewed as a generalization of shop floor control but in a broader way. With a modular architecture, the system level control system can also incorporate mid-term or long-term objectives of the manufacturing system. It can consist of manufacturing information system (MIS) and manufacturing execution system (MES).

As machine level control is the main focus in current studies of RMS control, this research proposes a reference architecture for system level control for RMS.

System architecture: system architecture refers to the architecture of a specific construct or system. A system architecture is the result of a design process, and specifies a solution for a specific problem. It specifies the structure of the pieces that make up the

complete solution, including the responsibilities of these pieces, their interconnections, and possibly even the appropriate technology.

Reference architecture: reference architecture is the "architecture as a style or method" as defined in the dictionary. It refers to design principles used for solutions in a specific domain. A reference architecture may specify similar aspects as a system architecture but in a more generic way.

Component: Software components are building blocks of software. A software component is a piece of code, with defined interfaces, that can be called to provide the functionality that the component encapsulates. In this proposal, "module" and "component" are considered the same and used interchangeable.

Agent: An autonomous component, that represents physical or logical objects in the system, capable to act in order to achieve its goal, and being able to interact with other agents, when it cannot reach alone its objectives.

Exception: an event happened in manufacturing system, which can affect the performance of the manufacturing system and is not considered in original scheduling process.

Chapter 2. Literature Review

This chapter reviews the literature in support of the development of this research and it is presented in four sections as follows:

- Existing manufacturing control structures
- Research on RMS control systems: state-of-the-art
- Component-based software technologies and new trends
- Agent-based systems and exception handling

2.1 Existing manufacturing control structures

2.1.1 Traditional control structures

The control structure has crucial importance in the final performance of the manufacturing control system. Even before the time of RMS, control structures have been widely studied. According to researchers (Diltis *et al.* 1991, Overmars and Toncich 1996), the control architectures can be classified as centralized, hierarchical and heterarchical. In this section, different traditional manufacturing control approaches are summarized.

- Centralized control architecture

The centralized architecture is characterized by a single decision node, where all the planning and processing information functions are concentrated (Diltis *et al.*, 1991). The advantages of the centralized system include a simple architecture and the possibility of global optimization. However, this architecture has a few drawbacks such as slow speed of response when the system has a large number of resources, difficulty in making any changes to the control software, and no fault-tolerance feature.

The centralized architecture has been used extensively for manufacturing systems. An example of a manufacturing system with a centralized control architecture is the flexible manufacturing cell (FMC) installed in Virginia Tech's Flexible Manufacturing Systems (FMS) Lab. A single computer is controlling all the equipment and devices of the cell formed by four processing machines, two MH robots and other devices, which

include conveyors, measuring devices, powered rotational buffers, and a quality checking system

- Hierarchical control architecture

The hierarchical architecture is a top-down approach, assuming a deterministic behavior of the system. The flow of commands is coming from top to down whereas information is flowing in the opposite direction. The main advantages of this architecture are the robustness, the predictability and the efficiency. The architecture is also easy to understand, and has shorter response time than in the case of centralized architecture.

Even though the hierarchical system has the possibility to obtain global optimization, it also has a few disadvantages. The inherent uncertainty of the real-world systems is not taken into consideration. The modified hierarchical (hybrid) control architecture allows horizontal flow of information among the lower level controllers, contrary to the proper hierarchical form where the information flow in vertical direction. Since the involvement of the lower level controllers in decision-making is limited, the hybrid architecture inherits the main features of the hierarchical architecture. The main difference is that modified hierarchical architecture allows a relatively local autonomy at the lower levels of the hierarchy and decreases the computational load on the master controllers, but at the same time, it adds more complexity to design and implement.

Nowadays, hierarchical architectures are most commonly used in the industries. The National Institute of Standards and Technology (NIST) of the United States, and the International Standards Organization (ISO) developed general hierarchical control models based on the hierarchical control approach. NIST developed the general hierarchical model of a manufacturing facility comprising of five levels: facility, shop, cell, workstation, and equipment. The ISO developed a standard for a hierarchical architecture, called Factory Automation Model, formed by six level of hierarchy, starting from the enterprise level, going through the facility/plant, section/area, cell, and station levels, and then, finally to the equipment level (Bauer *et al.*, 1994). Each of these levels is responsible for specific tasks and operates at decreasing planning time-horizons from months at the enterprise level to real-time at the equipment level.

Some other well-known hierarchical control structures include the COSIMA (Control Systems for Integrated Manufacturing), which has developed functional software architecture for cell and shop floor level. It consists of some function modules that are grouped into the Production Activity Control (PAC), located at the cell level, and Factory Coordination, located at the factory level (Bauer *et al.*, 1994); RapidCIM, a joint venture project between Texas A&M University and Penn State University, aiming to facilitate the process of developing full-automated computer controllers for FMS, used hierarchical control structure. FACT (Factory Activity Control Model), developed at University of Twente, contains six basic modules, distributed over two hierarchical levels (Arentsen, 1995).

- Heterarchical control architecture

Centralized and hierarchical control architectures are most suitable for dealing with pre-established situations. Any change in manufacturing conditions could result in large delays and even closing down the whole system. The introduction of the heterarchical control architectures, which allows only horizontal flow of information, meets the need for some degree of autonomy to enable components to respond dynamically to changes.

The heterarchical architecture is a totally decentralized architecture, formed by a group of independent entities called agents that have their own internal meta-heuristics embedded in a control unit. The tasks in the heterarchical architecture are performed by exchanging information among the agents. Heterarchical systems have the advantage of reduced complexity, high flexibility, and robustness against disturbance. However, without any global view entity, these systems are likely to exhibit unpredicted behavior, and worse can lead to chaos.

Some famous heterarchical control structures include MetaMorph I, II projects (Maturana and Norrie, 1996, Shen *et al.*, 1998), PABADIS (Sauter and Massotte, 2001) using the concepts of CMUs (co-operative Manufacturing Units), and YAMS (Yet Another Manufacturing System), etc.

2.1.2 Holonic manufacturing control systems

To face the requirements of new manufacturing environment, an international collaborative research program in manufacturing, called Intelligent Manufacturing Systems (IMS), was started in the beginning of nineties. Within the IMS program, several paradigms for future of the factory were developed, such as holonic, bionic and fractal manufacturing systems. Among these, holonic manufacturing systems (HMS) has been most extensively studied.

In middle of sixties, Arthur Koestler (Koestler, 1969) introduced the word holon to describe the basic unit of organization in living organisms and social organizations. Koestler concluded that parts and wholes do not exist in domain of life, and proposed the word holon to represent this hybrid nature, being a combination of the Greek word holos, which means whole, and the suffix on, which means particle.

HMS is a paradigm that translates to the manufacturing world the concepts developed by Koestler to living organisms and social organizations, mainly those that are complex hierarchical systems formed by intermediate stable forms. A holon in HMS can represent a physical or logical activity, such as a robot, a machine, an order, a flexible manufacturing system, or even an human operator. A holarchy is defined as a system of holons, organized in a hierarchical structure, cooperating to achieve the system goals, by combining their individual skills and knowledge. A holon can dynamically belong to multiple holarchies at the same time, which is an important difference to the traditional concept of hierarchies. The holons can integrate themselves into a holarchy and, at the same time, to preserve their autonomy and individuality. A HMS is a holarchy that integrates the entire range of manufacturing elements, such as machines, products, parts and automated guided-vehicles. In HMS, the holon's behaviors and activities are determined through cooperation with other holons, as opposed to being determined by a centralized mechanism.

Holonic control structures systems combine advantages of hierarchical and heterarchical approaches (Bongaerts *et al.*, 1998). Many HMS application systems have been developed already (Marik et.al. 2002, Deen 2003). However, current HMS applications are limited to small or simple ones. The reasons include the complexity of

implementing holons and holarchy, unpredictability of cooperative behavior of holons and also there is often no associated strategy to migrate the conceptual holon model into real factories with their heterogeneous automation and business information systems. All these problems are major hurdles that stop HMS concept being implemented in industrial applications.

From the review of existing control structures, it can be easily concluded that none of these control architecture can be directly used to fulfill RMS's challenges that are discussed in chapter 1.

2.2 Research on RMS control systems: state-of-the-art

Reconfigurable Manufacturing System (RMS) concept is formally introduced by Cakmakci *et al.* (1998) and Koren *et al.* (1999). A number of studies are conducted to justify the application of technology that leads to the development of RMS (Bertok *et al.* 1988; Cakmakci *et al.* 1998; DeGaspari 2002; Koren 2002; Wills et al 2001; Mehrabi *et al.* 2000a; Struebing 1996). Different levels of issues related to design and operation of RMS are distinguished in these papers. Current research of RMS mainly focuses on evaluating and choosing optimal configuration, modular machine design and open architecture control of machines. NSF Engineering Research Center (ERC) for Reconfigurable Machining Systems at the University of Michigan (Ann Arbor. Michigan) organizes its research into three categories: (1) System-Level, (2) Machine/Control, and (3) Calibration and Ramp-Up. The goal of system-level design is to reduce in reconfiguration lead-time by developing mathematical tools for cost effective systems. Machine/Control design studies the design of a new generation of reconfigurable machine tools and their reconfigurable controllers. Since early 1990's, several international initiatives (OMAC, OSACA, JOP) had worked on Open Architecture Control (OAC) for control of machines (Katz *et al.* 2000; West 2003). Compared with international initiatives and other activities on OAC for machine-level control, much less efforts were made on open architecture design for manufacturing system level control.

The following sections provide a review of available literature to date on RMS control. The literature is categorized in two general categories: machine-level and system-level. Although some of the reported studies may not explicitly indicate their

research efforts are related to RMS, we consider all manufacturing control attempting to realize reconfigurability as the control design for RMS. Reconfigurability (NSF 1996) is defined as the ability to adjust the production capacity and functionality of a manufacturing system to new circumstances through rearrangement or change of the system's components.

2.2.1 Machine-level control for RMS

Traditional machines in manufacturing systems were built as heterogeneous, device-oriented systems with proprietary software components. The tight coupling of software and hardware leads to very complex and inflexible systems. Open-architecture control (OAC) is considered as the key enabler for the realization of modular and re-configurable machining systems. OAC is aimed to provide a vendor-neutral, standardized environment based on defined open interface and software platform.

Since 1990's, several international initiatives have worked on concepts of OAC. They are OMAC in U.S.A, OSACA in Europe, JOP in Japan and also activities in other organizations and universities. Comparison of these OAC approaches shows that although they are all based on similar concepts and common elements, they are not compatible with each other (Katz *et al.* 2000). Since 1996, the main groups from Europe, U.S, and Japan started to work together towards a global human machine interface (HMI) standard with a shared and unified Application Programming Interface (API). More information about OAC initiatives can be found in Katz *et al.* (2000) and West (2003).

In the area of design and implementation of machine-level control, some fo the research area work reported in the literature are described in this section. Balasubramanian and Norrie (1996) addressed seamless integration of all the manufacturing control functions, and also reported design of a distributed intelligent controller using combined agent-holon approach without implementation information; Ambra *et al.* (2002) reviewed various design options available for open architecture CNC systems, and proposed a different approach integrating the real time activities, including interpolation and position measurement, within an Field Programmable Gate Array (FPGA); Shah *et al.* (2002) described the design and implementation of logic controllers on small-scale machining line testbed. The logic control was written using modular finite

state machines; Park *et al*. (1998) introduced and formalized a modular logic controller for high volume transfer lines. The modular logic controller was implemented using sequential function chart (SFC). Logic control modules in overall system was developed using petri net; Huang *et al.* (2001) applied UML to a software development process and presented that UML can modularize and share software under generic real-time control system architecture. Also, UML can provide a standard representation that is critical to open systems and be easy to understand. Some researchers built control for facilities other than CNC machines: Chen *et al.* (1998) used Cross-Coupling Control with Friction Compensation to provide Modular Control for Machine Tools. Furness *et al.* (1996) studied Feed, Speed, and Torque Controllers for Drilling in their research. Zhang *et al.* (2000) presented system architecture for holonic PLC and detailed multi-level reconfiguration control mechanism for function block based PLC.

There was also research undertaken to study proper software tool to build machine-level control. Wang and Shin (2002) presented a modular architecture for constructing reconfigurable software for machine tools control systems using OOP. Kolla *et al.* (2002) reviewed software programming issues and the use of Microsoft COM in the development of machine control components.

Other researchers conducted performance analysis and validations in their work. Lian *et al.* (2000) parameterized and analyzed system performance and reconfiguration issues using a machine tool. They addressed issues of evaluating system performance and developing control software solution in their design of the machine tool. Yook *et al.* (1998) considered several different architectures for a control system of a reconfigurable machining system in terms of their effects on system performance. Kalita and Khargonekar (2002) presented a hierarchical structure and framework for the modeling, specification, analysis and design of logic controllers for a RMS.

2.2.2 System-level control for RMS

A number of studies have been reported in the area of shop floor control or system-level manufacturing control before RMS concept has been introduced. Detailed overview of these works can be found in Basnet and Mize (1994), Daeyoung *et al.* (1996) and Dilts *et al.* (1991). Some well-known control architecture includes centralized control

architecture, hierarchical architecture, and heterarchical architecture. A centralized system has a single control unit that is responsible to make all the decisions. Such systems have the advantage of having a simple architecture and the possibility of global optimization. However, the architecture suffers from slow response and limited fault-tolerance. In a hierarchical architecture, higher levels control the lower level via a master-slave relationship. They typically have a top-down flow of commands and a bottom-up flow of information. In some modified hierarchical control architectures, peer-to-peer communications are allowed among entities. All the hierarchical control architectures assume deterministic behavior of the system and require a fixed structure. Hierarchical architecture yields a simple and fault-tolerant system, which is able to overcome the difficulties deliver a rapid response to disturbances in system presented by both centralized and hierarchical architecture. The main drawback of such system is that it is impossible to seek the global optimization and therefore the performance of the system is unpredictable.

Recently, significant research efforts have been made in holonic control (Brennan 2000; Dumitrache *et al.* 2000; Heikkila *et al.* 2001; Lim and Zhang 2003; Monostori 1999). The concept is created based on an observation made by Koestler (1967) that a complex system's constituent entities are both wholes and parts at the same time. Typically, in holonic manufacturing systems, manufacturing facilities are modeled as resource holons, which enables system-level reconfiguration by adding, removing, or replacing resource holons. Holonic control provides autonomy, reliability, fault-tolerant, and other real-time functionalities. Some researchers considered reconfigurability specifically in their holonic control design. Another research area that is closely related to honlonic control approach is agent-based control system. Zhang *et al.* (2002) defined a multi-agent based architecture that supports the design and implementation of reconfigurable control systems for agile manufacturing cells. Different agents in the cell control system can be organized dynamically, communicate with each other through messages, and cooperate with each other to perform tasks flexibly in cell control system. Brennan *et al.* (2002) described a general approach for dynamic and intelligent reconfiguration of real-time distributed control systems that utilizes the IEC 61499 function block model. The approach was based on object-oriented and agent-based

methods. Dumitrache *et al.* (2000) presented a supervised control architecture designed for flexible manufacturing systems. The architecture was intended to combine the advantages of hierarchical and heterarchical agent-based control structures. Zhou *et al.* (1999) described agent-based metamorphic control architecture and function-block-based design of control agents for distributed manufacturing environments. The approach provided a viable alternative to current static hierarchical systems, by enabling the control structure with reactive and dynamic reconfigurable capabilities. Cheung *et al.* (2000) carried out their research with the aim of devising a holonic control framework. Simulation results showed the framework was extensible and reconfigurable. More detailed review of holonic/agent-based control can be found in (Babiceanu and Chen 2006; Dumitrache *et al.* 2000; Kotak *et al.* 2003; Shen and Norrie 1999).

Nowadays, object-oriented programming (OOP) is most commonly used software technology in the design of manufacturing control (Chan *et al.* 2000; Grabot and Huguet 1996; Howard *et al.* 1998). OOP can also be considered as a desirable choice for RMS control (Mehrabi *et al.* 2002). For example, Chan (2000) defined an object-oriented architecture that supports design and implementation of reconfigurable control systems for agile manufacturing cells. The architecture is composed of database objects, control objects, and resource objects. Pasek (2000) described the development of OpenFront, a high-level control architecture that can be used by system configurators/integrators. They reported to use platform-independent software (e.g. object-oriented software, java computing, etc). In holonic manufacturing systems, holons are usually programmed as objects using OOP.

Recently, component-based software technology is reported in design of manufacturing control. Chirn and McFarlane (2000) presented an a conceptual architecture of holonic component-based architecture (HCBA) that is implemented in a robot assembly cell. Authors reported plug-and-play capacity via an internet-based infrastructure but without details. Morton *et al*. (2002) presented a state-based approach to model and design software components that can be used as building blocks for flexible manufacturing control software. The experiment, however, is limited to one product and buffers with unlimited sizes in the production line.

Other research topics on system-level control for RMS include selection of configuration, control logic design, and updating of legacy systems. Rao and Gu (1995) presented an entropic measure to indicate the need of reconfiguration ability required by manufacturing systems. The author concluded that the reliability of the entropic measure was not determined. Yook *et al.* (1998) presented a theoretical framework to build models, which is used to determine the optimal network architecture for a given control system. The given control system was assumed to be designed without taking into account the network architecture. Time delays were the main factor in considering optimal network architecture. Su and Chen (2003) presented a method of developing device controls from legacy system. These device controls can be used as building blocks for RMS control design. Ramirez-Serrano and Benhabib (2003) used a nominal/complementary supervisor pair to control workcells. The supervisors are synthesized using Extended Moore Automata (EMA). The methodology allows the control of such workcells without changing the structure of its nominal supervisor when adding/removing machines. Storoshchuk *et al.* (2001) reported their work of transferring software technology from the theoretical domain into industrial applications. Their approach to model manufacturing control systems is based on a framework for subsystem composition and a Nested Finite State Machine (NFSM) model for system behavior. They reported development and debug time is reduced to half, and modification time is reduced to one-third.

2.2.3 Section summary

Subsystems and components (hardware/software) in RMS must be designed as open-ended, so that it can be improved, upgraded, and reconfigured, instead of totally replacing the complete system. As compared with international initiatives and other activities on OAC for machine-level control, much less efforts were made on open architecture design for system level control. To make a manufacturing system reconfigurable, system-level control software must be designed with reconfigurability at the outset (Mehrabi *et al.* 2002).

Agent technologies have rapidly gained popularity in manufacturing area in past few year. Some of the system-level control using agent technology claimed

reconfigurability because agents may be added or replaced freely in agent-based systems. Agent-based systems enable reconfigurable control logic for a manufacturing system. However, the rigidity of software architecture, which makes control software difficult to be changed to accommodate changing requirements, remains unresolved.

Most current industrial and academic manufacturing control systems are built using OOP. Now, development in component-based software technology can change this situation. Component-based software technology is initially designed to build modular software (with components). A simple example explaining why component-based software technology is better than OOP in building modular control software: it is not easy to add or remove an object from object-orient software compared with adding/removing a component in component-based software technology. More discussion about comparison of OOP with component-based software technology can be found in Hoagland (2005) and Pasek *et al.* (2000). The next section provides reviews on the component-based software technologies and emerging trends.

2.3 Component-based software technologies and new trends

As concluded by the Nobel prize winner Herbert Simon (1982), "the complex systems will evolve from simple systems much more rapidly if there are stable intermediate forms than if there are not". Simon (1982) illustrated this feature of complex systems by a parable of two watchmakers. Both watchmakers made very fine watches. In the end, one lost his shop while the other prospered. The reason for the one who succeeded is that he designed his watches that can be put together by stable subassemblies. Then his work did not fall into pieces when phone rang like the failed watchmaker. The software engineering is inherently complex (Kozaczynski and Booch 1998). Software components can help cope with system complexity. In the last few years, components and component-based software development have gained substantial interest in the software community. The new methodology helps organize large-scale development and, most importantly, makes system building less expensive.

2.3.1 Component-based software technology

OOP was a major advancement in software technology. One of the most frequently mentioned feature of OOP is its modularity and reusability. However, objects in OOP cannot be reused easily because they are tightly coupled and may have complicated interactions with other objects in the object-oriented software.

The high cost and complexity of software creation have driven both researchers and practitioners toward design methodologies that will decompose design problems into manageable pieces, which then can be assembled into complete software artifacts. Component-based development (CBD) is increasingly gaining popularity in software development. A component can be defined as "a coherent package of software that can be independently developed and delivered as a unit, and that offers interfaces by which it can be connected, unchanged, with other components to compose a larger system" (D'Souza and Wills 1998a). By enhancing the flexibility and maintainability of systems, this approach can potentially be used to reduce software development costs, assemble systems rapidly, improve quality and reduce the huge maintenance burden associated with the support and upgrade of large systems.

In CBD, the notion of building a system by writing code has been replaced with building a system by assembling and integrating software components. In contrast to traditional development, implementation has given way to integration as the focus of system construction (Capretz 2001). As such, the specification of the components is more important, from the assembly perspective, than the way that specification is realized (Gorton and Liu 2004). Right component specifications have been viewed as a critical factor in major activities of CBSD, such as component qualification, and component adaptation. Only when components are clearly identified and specified, robust and flexible application architectures can be produced. Sometimes, abstract specifications or even design documents are considered to be components themselves. There is an increased interest in business-level components, which are independent of any specific implementation or middleware technology.

When comparing with OOP, component based programming is a packaging and distribution technology while OOP is an implementation technology. Internal code of components is usually implemented using object-oriented methods.

2.3.2 New trends in CBSD

Current component-based software platforms from Microsoft, Sun Microsystems etc overcome drawbacks of earlier component-based software platforms in terms of complexity, large software size and poor real-time features. However, it is still difficult for applications based on these different platforms to interact with each other. The approaches to solve the problem normally involve developing middleware applications to communicate the non-interoperable applications using the message broker technology. For example, developing and deploying a compatible object request broker (ORB) with the Common Object Request Broker Architecture (CORBA) or Distributed Component Object Model (DCOM) is a complex issue requiring special expertise.

In recent years, emerging standards-based integration technologies such as Web services and Extensible Markup Language (XML) allow pieces of software to communicate without the interoperability problems that traditional software technologies have difficulties to solve. The XML is a W3C-recommended general-purpose markup language for creating special-purpose markup languages, capable of describing many different kinds of data. XML is a cross-platform, extensible, and text-based standard for representing data. Since XML data is stored in plain text format, XML provides a software and hardware independent way of sharing data.

Web services have emerged as the next generation of integration technology. Based on open standards, the web services technology allows any piece of software to communicate with each other in a standardized XML messaging system. It eliminates many of the interoperability issues that the traditional integration solutions have difficulty in resolving. Web services technology depends mainly on the XML data format. Both messages and all web services protocols are based on XML.

The Web-services framework contains three main elements: communication protocols, service descriptions, and service discovery, each specified by an open standard. Web services are invoked using Simple Object Access Protocol (SOAP), a standard

XML-based protocol for messaging on the Internet. Encoded in XML, SOAP provides a way to communicate between applications developed with different programming languages and running on different operating systems. The interface exposed to the external Web services is described in the XML-based Web Services Description Language (WSDL). WSDL is used to describe the Web service, specifying its location and publishing its operations (i.e., methods). Universal Description, Discovery, and Integration (UDDI) is a standard-based specification for service description and discovery. The UDDI specifications offer Web-service users or consumers a unified and systematic way to find service providers through a centralized registry of service. The registry of service is an automated online "phone directory," through which the registered Web services advertise their business services. Registry access is accomplished using a standard SOAP API for both querying and updating. Combining these open standards, the Web-services technology provides a standardized way of publishing, locating, and invoking the business services.

The use of standard XML protocols makes Web services platform, language, and vendor-independent. Most current modular design of control systems is vendor-dependant, thus they can only integrate with components from the same vendor. Advances in information technology can help overcome the shortcomings. Based on open XML data format and open protocols, it is possible to develop open, standard-based, and vendor-neutral, reconfigurable, system-level control architecture. Due to this reason there is an increased interest in component specifications, which are independent of any specific implementation or middleware technology.

2.3.3 Component specification in CBSD

An analogy of software component specifications is Very High Speed Integrated Circuits Hardware Description Language (VHDL), which defines the characteristics and behavior of a chip. VHDL defines specifications for a circuit designer to choose chips and chipsets for a new printed circuit board design. In CBSD, precise component specifications allow designers to locate available components that match their criteria and evaluate their performance through simulation, based on their specifications (Messina *et al.* 1999). Component specifications are of increasing importance to the manufacturing

industries as the present time is in the midst of a major shift in technology from closed, proprietary systems to the era of open and plug-and-play systems.

The component specification has been studied in different ways in literature. The component models such as COM or CORBA uses interface definition language (IDL) to define the specification of its interface as component specification. Architecture Description Language (ADL) is a formal language for representing and reasoning about software architecture. The different ADL provides support for different kinds of properties, such as protocol specifications or connector specifications. In manufacturing industries, some researchers use IEC61499 for their modular design of control software. IEC61499 is a standard for distributed industrial-process measurement and control systems based on programming language standard IEC61331-3. It uses the function block model and finite state machine to represent the structure and behavior of a system. For example, Xia *et al.* (2004) uses IEC61499 to specify components in control system engineering. Brennan *et al*. (2002) described a general approach for dynamic and intelligent reconfiguration of real-time distributed control systems that utilizes the IEC61499 function block model.

However, IDLs and ADLs are usually proprietarily developed and hard to understand. IEC61499 is originally designed for process control and it is not suitable to specify generic software components. As compared with above approaches, the Unified Modeling Language (UML) is much easier to understand as a graphic language and it can specify software components precisely. UML is the de facto standard for nearly all application modeling and it has been used in modeling CBSD recently (Cheesman and Daniels 2000; D'Souza and Wills 1998a). UML is originally designed for object-oriented modeling. Cheesman and Daniels (2000) described how UML could be extended for component specification in their book. The work provides a complete process of UML modeling of software components from requirements definition, to component identification, component interaction and component specification.

2.4 Agent-based scheduling and exception handling

Due to inherently centralized and hierarchical structure, conventional control logic are not flexible enough to address the new constraints such as openness, inter-operability

and fault-tolerance. Multi-agent systems have been recently proposed as a promising technology and an appropriate way to overcome these problems and to offer a convenient environment for designing and implementing distributed manufacturing solutions. While component-based software technology enables modular reconfigurable software architecture for control systems, agent technologies brings reconfigurability to logic of control systems since agents can be added or replaced freely in agent-based systems.

An agent is a computer program that is capable of autonomous action in order to meet its design objectives (Wooldridge 1995). A Multi-Agent System (MAS) is a system composed of a population of autonomous agents, which cooperate with each other to reach common objectives, while simultaneously each agent pursues individual objectives (Ferber 1999; Wooldridge 1995). A large number of applications and research has been conducted on agent-based systems in both academia and industries.

Although there are many desirable features contained in agent-based systems, some researchers studied weaknesses of agent systems and recommended to use them with cautions (Deshmukh 1999; Wooldridge and Jennings 1998). A centralized control system based on global information and optimization techniques are highly likely to yield better decisions than completely agent-based system. It is usually impossible for agent systems to seek global optimization point and therefore system performance is therefore unpredictable (Brussel *et al.* 1999; Duffie and Piper 1987; Roy *et al.* 2001). Global properties of such systems emerge from all individual behaviors and one-to-one interactions. Global performance is therefore hardly predictable, and generally simulation is used to prove or establish stable, good performing agent-based solutions. Another potential problem is the effects of large number of agents and excessive communications (Shen 2002; Wang *et al.* 2003). They can significantly deteriorate the system performances. Due to these issues, agents can be designed to be activated only when exceptions occur. Compared to a pure agent-based scheduling system, such exception handling mechanism avoids the risk of missing the global optimal point and reduce system burden when system is under normal condition.

Uncertainty and the disruptions in production associated with the resulting perturbations have been discussed as early as the beginning of 1900s (Aytug *et al.* 2005). Beek (1993) presented a comprehensive review and defined concepts of failure, error,

faults, and exception in his doctoral thesis. Bruccoleri *et al.* (2003) defined Exceptions as differences between the actual and the expected state of the production system and Exception handling as the policy meant for countering unwanted effects of exceptions and for recovering from errors caused by exception occurrences. Exceptions can be machine breakdowns, changes in job priorities, dynamic introduction of new jobs, order cancellations, increases in job arrival rates, changes in the mix of parts, and reworks due to quality problems (Bruccoleri *et al.* 2003). Prabhu *et al.* (1992) classified nine categories referring to temporal errors in manufacturing systems. Three main ways of exception handling that can be found in the literature are summarized in Aytug *et al.* (2005) and Bruccoleri *et al.* (2006): predictive schedule; reactive scheduling; predictive reactive scheduling. Some of the most utilized methods for exception handling include: knowledge-based systems, artificial intelligence systems, agent-based systems, holonic systems, simulation, beam search, heuristic algorithms, dynamic selection of dispatching rules, etc. A complete literature reviews on exception handling methods can be found in Prabhu *et al.* (1992) and Sabuncuoglu and Bayiz (2000).

2.5 Summary

In this chapter, relevant topics and subject areas are reviewed. After comparing existing literature on system-level and machine-level control systems for RMS, research conducted on system-level control systems is quite limited and needs further exploration. There are no current existing control structures that can be directly used to fulfill the promises of RMS control systems. At the same time, most current industrial and academic manufacturing control systems are developed in OOP. Some researchers started to use emerging component-based software technology, which enables true modularity compared to OOP, in designing control systems. However, most of these works are based on specific component-based framework such as .Net from Microsoft or CORBA from OMG. With the new developments of XML-based open data format and open standards, it is possible to have new software framework that is able to be vendor-neutral and independent of programming languages and operation systems. Under such circumstances, component specifications will be more important than how a component is implemented.

While component-based software technology enables modular reconfigurable software architecture for control systems, another promising technology that can bring reconfigurability into control logic of control systems is agent-based technologies. Agent-based technologies have been widely studied in different areas in the manufacturing sector. The multi-agent paradigm has proven its ability as in non-deterministic environment when compared to traditional methodologies. However, it also has a few shortcomings, such as unpredictable global performance, high computation burden, etc. One possible way to avoid risks caused by such drawbacks is to use agent-based systems to handle only unexpected events only.

In the following chapters, a new generic system-level control architecture for RMS is presented. Component-based technology and agent-based technology comprise the backbone of the control architecture. The details of the architecture and experimental analysis are presented in later chapters.

Chapter 3. Component-based Intelligent Control Architecture

Today, the common practice of developing manufacturing control software is the use of proprietary design, which is completely compiled and linked into one executable program before run-time. Such a solution is inflexible and requires tremendous efforts if modifications are necessary after installation. In a RMS, control system needs to be adapted quickly to new requirements without affecting and interrupting the performance of manufacturing systems. The reconfigurable nature of RMS requires its software/hardware to be in a modular form (Mehrabi *et al.* 2002). A component-based intelligent control architecture, which is built with software modules and intelligent agents, can realize the simple and cost effective configuration and extension of control systems and therefore permitting flexible adaptation to varying production requirements.

The idea of the proposed research is mainly derived from new advances in component-based software technology as well as outcomes from the agent-based manufacturing control studies. The result of this study is expected to provide a promising way of designing future reconfigurable manufacturing control system. In the following sections of this chapter, an overview of the proposed control architecture and its component-based feature are presented.

3.1 Overview of proposed control architecture

The proposed control architecture entitled Component-based Intelligent Control Architecture (CICA) consists of two main features: component-based software architecture and agent-based intelligence. Control architecture is composed of software components. The application of component-based software technologies facilitate the software architecture of CICA to be easily reconfigured while using agent technology to handle exceptions. The inherent flexible and reconfigurable feature of agent technologies brings reconfigurability to the control architecture in control logic.

The physical equipments in a manufacturing system with CICA architecture have their corresponding software control modules and intelligent agents in software layer and intelligence layer respectively. The reconfiguration ability can easily be added in the software layer and intelligence layer of CICA along with physical changes in the

hardware layer when reconfiguration is needed. Such design makes the CICA architecture reconfigurable both in software structure and control logic.

In such a design, a shop floor control system can be viewed as a system consisting of three layers (Figure 3.1): 1) physical layer: this includes all the physical equipments on shop floor with all machines and their individual controllers; 2) software layer: this includes control software built with software components including machine control modules, decision making module and other auxiliary modules; and 3) intelligence layer, which includes the intelligent agents that can be used to handle exceptions when disturbances are detected.

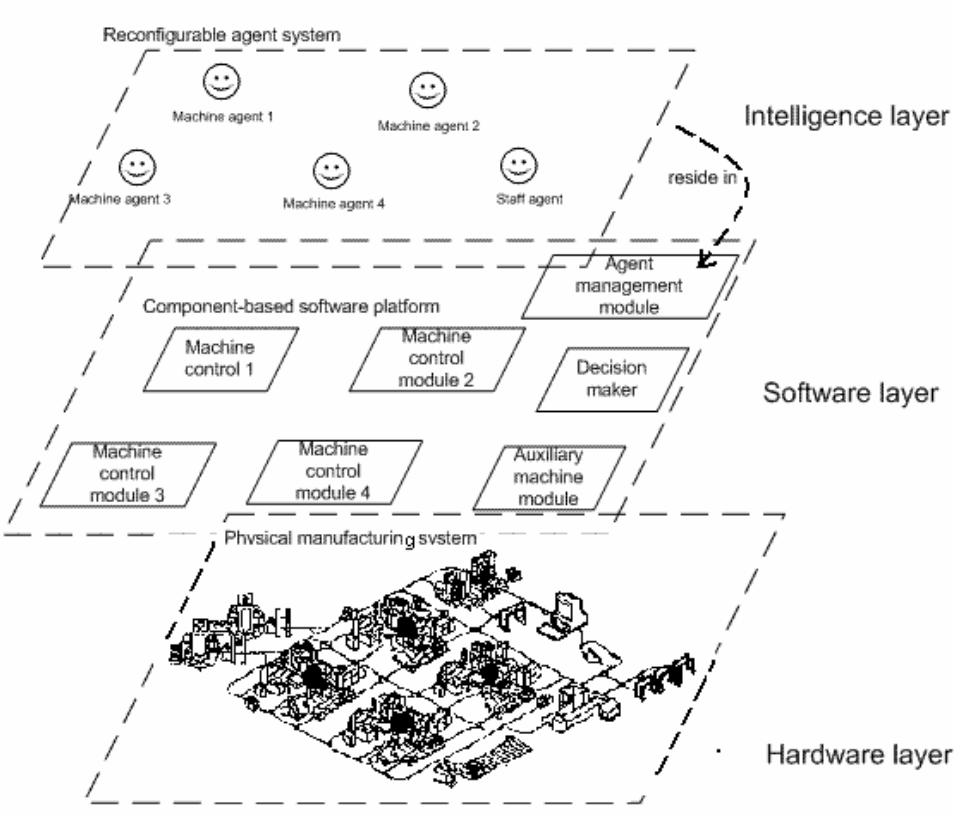


Figure 3.1 A 3-layer manufacturing system with CICA control architecture

Decoupling is considered as one of the important issues in designing complex systems (Wyns 1999). In CICA, software structure is decoupled from control algorithm. Based on the modular design of CICA, control software can be changed with minimal cost according to changes made in manufacturing systems. A centralized decision module

is implemented as plug-in module that contains the control logic of the shop floor system. When exceptions in the system are detected, the intelligence layer players - agents that reside in agent management module will take over the control of shop floor systems. The control logic is able to be changed without affecting the architecture.

3.2 Component-based control architecture

The key to provide reconfigurable systems is to have a modular structure. The Component-based Software Development (CBSD) is an emerging discipline enabling software that can be able to be assembled from components in the same way that most hardware systems (machinery and equipment) are constructed from standard parts nowadays. In hardware production industries, the component-based development has been widely used in practice over a long period of time. The development and maintenance of software components is quite similar to hardware components. By enhancing the flexibility and maintainability of systems, CBSD has a great potential to reduce software development costs and to dramatically reduce system assemble/configure time. Due to its inherent modularity and plug-and-play nature, CBSD is a powerful tool for designing system-level control for RMS.

One of the major motivations for using a component approach in this work is to manage changes more efficiently. The module architecture simplifies the design of a system by decomposing the system into functional “building blocks” and capturing their interrelationships. The encapsulation of components implies that changes can be done inside a component with only minimal impact on the whole system.

Since a component is intended to be used as a “black box” entity, external aspects of software components are more important than their internals, e.g. how the components are implemented. The programming language, or even the underlying operating system of a component, should be transparent to other software components. Because of the transparency, it is essential to have clear specification that is separated from component implementation.

A major part of a component specification is the definition of component interfaces, which defines the services that the component provides and their constraints. An interface is an essential element of a component, since it is often the only information that is

exposed to outsiders. The component interfaces define the question *what* services are provided by a particular component to the context of its existence, but not *how* these services are actually realized. The inter-component dependencies can be restricted to individual interfaces, rather than encompass the whole component specification. These characteristics allow a component to be upgraded or replaced with minimal impact on the system.

The component composition is the process of assembling software components. Components are 'wired' together to form the complete system based on the functionality and dependencies they have. The meta-model of main component concepts in CBSD is presented in Figure 3.2

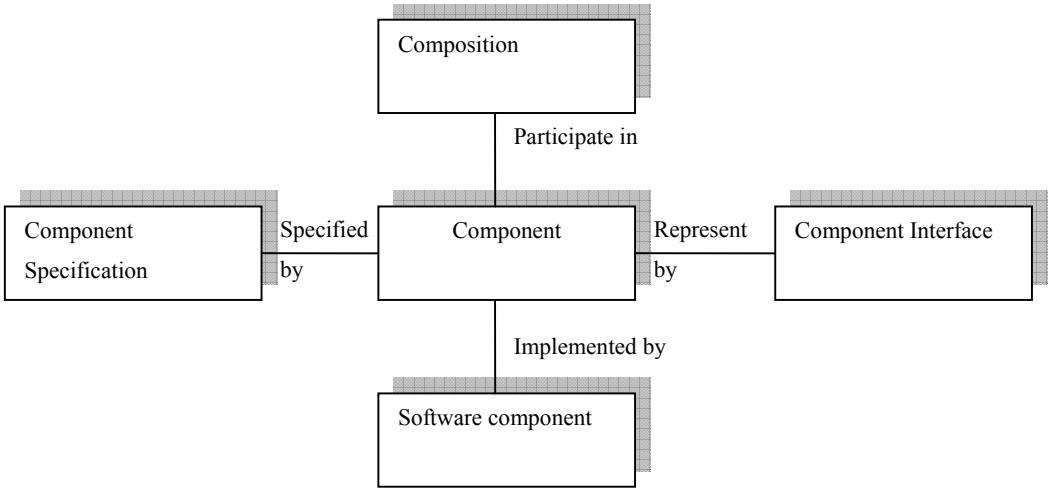


Figure 3.2 Component forms in CBSD

Most control software claims modularity in its design nowadays. Nonetheless, the modularity is highly vendor-dependent, which implies control software can only integrate with components from the same vendor. The emerging standards-based integration technologies such as Web services and XML overcome the interoperability problems that traditional software technologies are facing. The strengths of XML and the Web services, namely simplicity of access and ubiquity, are important in resolving the fragmented middleware world where interoperability is hard to achieve.

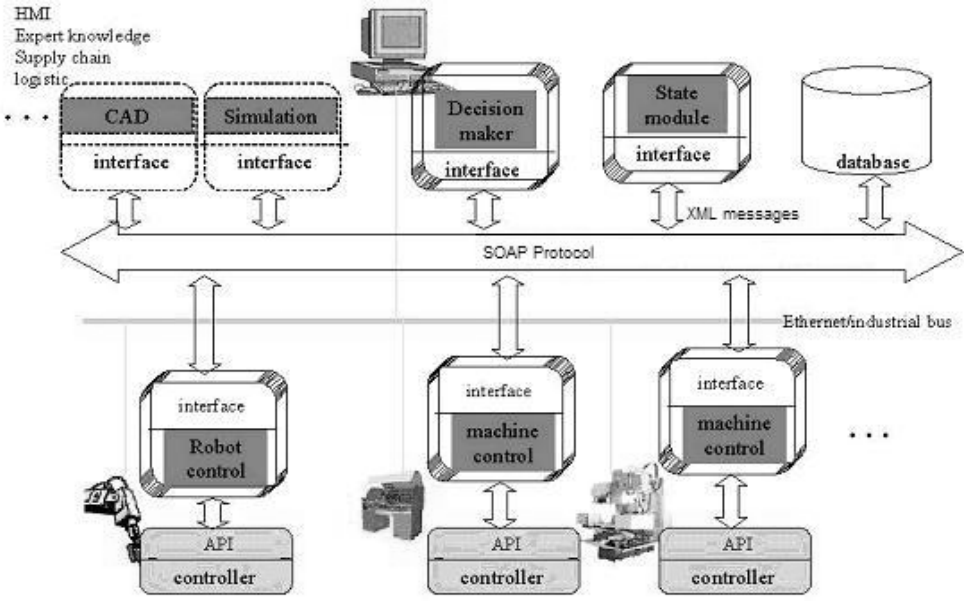


Figure 3.3 Component-based system-level control architecture

In the component-based intelligent control architecture, components are loosely coupled and they are connected by open data format. With well-defined interfaces, these components can be easily added/removed/updated in flexible software using component based software technology (Figure 3.3). All these components may be implemented by different vendors as long as their interfaces fit together. The performance improvement and system tuning is made much easier. The software system can easily be reconfigured when configurations of system hardware are changed.

A simulation environment can be designed with a system of existing software/hardware components and simulated modules. All the changes to the system can be validated and tested in the simulation environment. Once the changes are tested, integration of the new code with the manufacturing system will proceed quickly. The architecture also provides significant advantages during maintenance and upgrades. Both hardware and software related errors can be identified faster and fixed more conveniently.

3.3 Component specifications using UML

Component specification has been studied in many different ways. As compared with other approaches, the Unified Modeling language (UML) is much easier to understand and can specify software components precisely.

UML is a general-purpose modeling language defined at Object Management Group (OMG) that includes a standardized graphical notation used to create an abstract model of a system, referred to as a UML model. The advantage of UML diagrams is that they are simple and standardized. UML used different means of diagram types to specify software. Each UML diagram is designed to view a system from a different perspective and in varying degrees of abstraction. The total thirteen UML diagrams belong to three classifications of UML diagrams: 1). Behavior diagrams: a type of diagram that depicts behavioral features of a system or business process. This includes activity, state machine, and use case diagrams as well as the four interaction diagrams. 2). Interaction diagrams: a subset of behavior diagrams which emphasize object interactions. This includes communication, interaction overview, sequence, and timing diagrams. 3). Structure diagrams: a type of diagram that depicts the elements of a specification that are irrespective of time. This includes class, composite structure, component, deployment, object, and package diagrams.

UML was originally designed as a language for objective-oriented analysis and design. Further it is based on the assumption of an objective-oriented implementation. The currently standardized version of UML does not provide the necessary modeling and specification constructs for representing various component and service concepts, mainly as a result of its OO focus and of it introducing component concepts as afterthought. As UML components are implemented packaging and deployment constructs, whereas specification construct is the real concern in CBSD. The component diagram or deployment diagram are also not used because specification but implementation is their focus. However, the standard UML offers a number of extension mechanisms that enable the user to extend the UML foundation with new modeling constructs according to different purposes. Stereotype is one of UML extension mechanisms. Stereotypes allow

users create new UML modeling elements. A stereotype's name normally appears on the element enclosed by << >>.

Cheesman and Daniels (2000) developed a process by which components can be precisely identified and specified. The method called UML components is strongly influenced by Catalysis, RUP, and Syntropy (Cook and Daniels 1994; Fowler 2003). It focuses on the specification of component using the UML, while the internal design and implementation of components are not concerned. A detailed explanation of the basic principals of software components and component-based development is given in their book, in a manner that establishes a precise set of foundational definitions. Although their work is originally aimed at using a model-based approach to design and construct generic enterprise-scale business software, their methodology can be easily adapted into manufacturing context.

A number of UML extensions have been made to suit needs of the UML to model components. As UML component diagrams focus on the implementation and deployment, component specification is defined as another stereotype of class called <<comp spec>>. A UML component realizes a <<comp spec>> class (Figure 3.4). UML interfaces also have implementation focus they do not have attributes or associations. A stereotyped class <<interface type>> is used to represent interface at specification level.

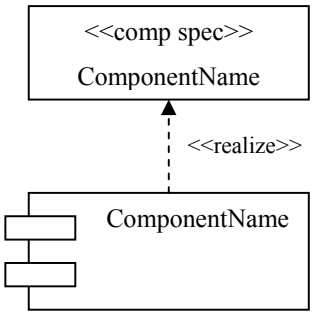


Figure 3.4 UML component realizes component specification diagram

Following the UML components methodology, a component architecture and a set of component specifications is generated in the end. The component architecture is a set of application level software components, their structural relationships, and their behavioral dependencies. The structural relationships represent associations and inheritance between component specifications and component interfaces, and

composition relationships between components. The behavioral dependencies imply that dependency relationships between components and other components, between components and interfaces, and between interfaces.

The component architecture describes how the component specifications fit together in a given configuration. It binds the interface dependencies of the individual component specifications into component dependencies. The architecture depicts how the building blocks fit together to form a system that meets the requirements.

A component specification (Figure 3.5) is the building block of a component architecture. In addition to the interfaces it offers, it defines the set of interfaces it must utilize. A provided interface is modeled using the lollipop and a required interface is modeled using a dashed arrow line towards the interface. A component specification also defines any constraints that it requires between the interfaces, and any constraints on the implementation of its operations.

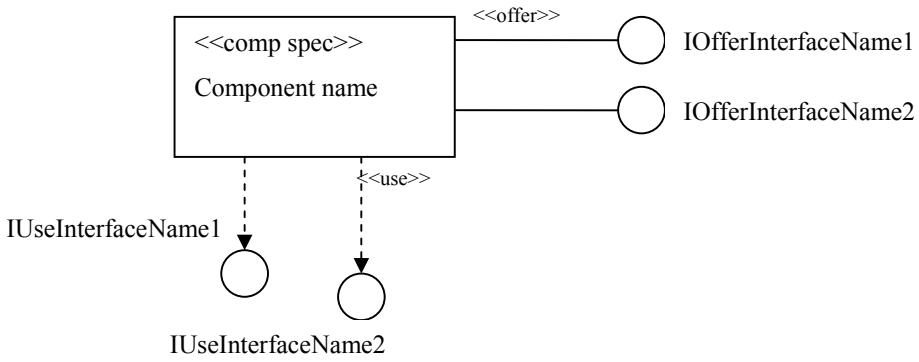


Figure 3.5 Provided/Required interfaces in Component Specification

Interfaces are access points of components through which other components can request services declared in the interface. An interface (Figure 3.6) describes a group of operations used or created by components. Each operation defines some services or function that the component object will perform. An operation signature is defined in the following format:

InterfaceName::OperationName(input parameters info): Output parameter info [ ]

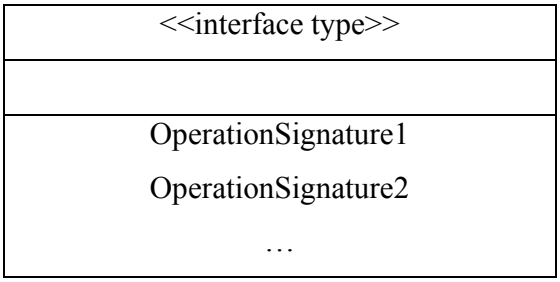


Figure 3.6 Interface specification diagram

Each operation has pre and post conditions. These conditions specify the effects of the operation without prescribing an implementation. The pre-conditions are the conditions under which the operation guarantees that the post-conditions will come true. Pre- and Post- conditions can be specified by Object Constraint Language (OCL) precisely. OCL is a formal language that can be used to describe expressions on UML models. It is used for writing invariants and operation pre and post conditions (Figure 3.7). It is hard to use and understand, but offers great precision. A full analysis of OCL writing for pre- and post-condition can be found in (D'Souza and Wills, 1999).

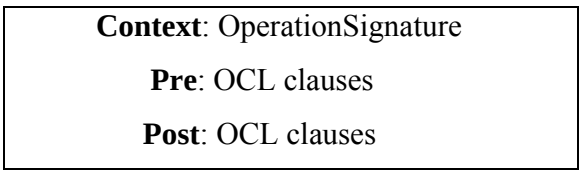


Figure 3.7 structure of OCL description of an operation

3.3.1 Component specification workflow

Figure 3.8 shows the overall development process in CBSD. The process is composed of six workflows: requirements, specification, provision, assembly, test and deployment. The specification workflow is further divided into three sections: component identification, component interaction and component specification.

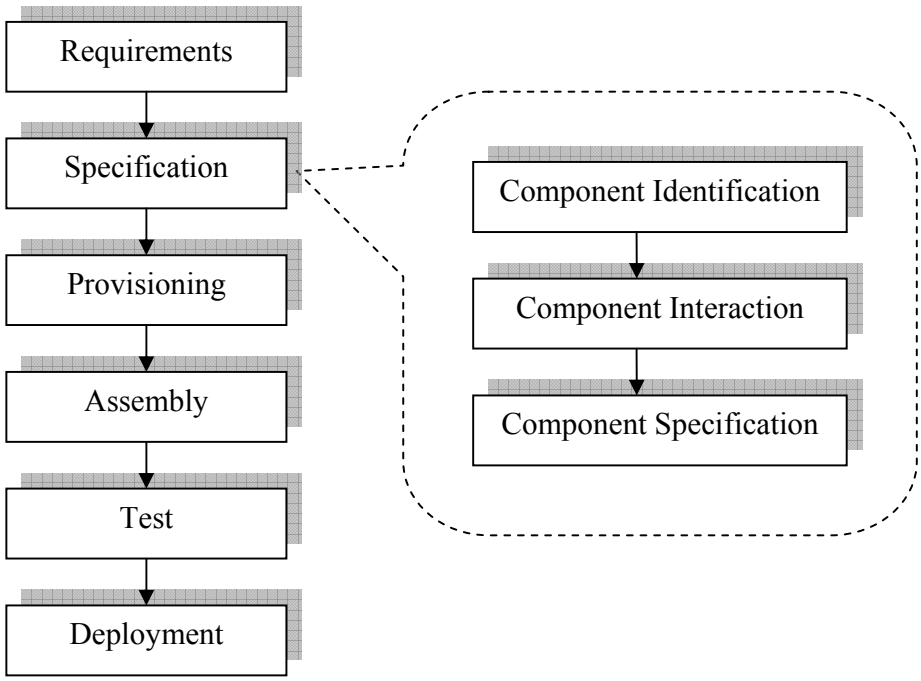


Figure 3.8 The workflow in the overall CBSD development process

1. Requirements definition workflow

The requirement identification stage lists the important concepts in the problem domain and establishs the relationship between them, and clarifies the software boundary by identifying actors who interact with the system, and describe those interactions.

In the manufacturing context, gathering the requirements for a system involves obtaining information from the customer about the behavior of the system and constraints or standards which must be followed. A requirements document is produced that lists constraints, process needs, production capacity, interfaces to other equipment, relevant standards, mechanical layout, and general strategies for its implementation.

2. Specification workflow

The specification workflow is the stage that a set of component specifications and component architecture is generated. There are three sections in the stage: component identification, component interaction and component specification.

The goal of component identification phase is to create draft application architecture, composed of a set of interfaces and preliminary component specifications. The activity identifies system interfaces and operation.

Based on the outputs from requirements workflow, business components of the system can be identified and specified. Components are identified through identifying system interfaces that are related to use cases. A use case is a way of specifying certain aspects of the functional requirements of the system. It describes interactions between an actor and the system, and therefore helps to define the system boundary. The interfaces correspond to use cases, and their operations are derived from use case steps. Use cases are broken down into steps and the steps are used to identify the system operations needed to fulfill the system's responsibilities (Figure 3.9).

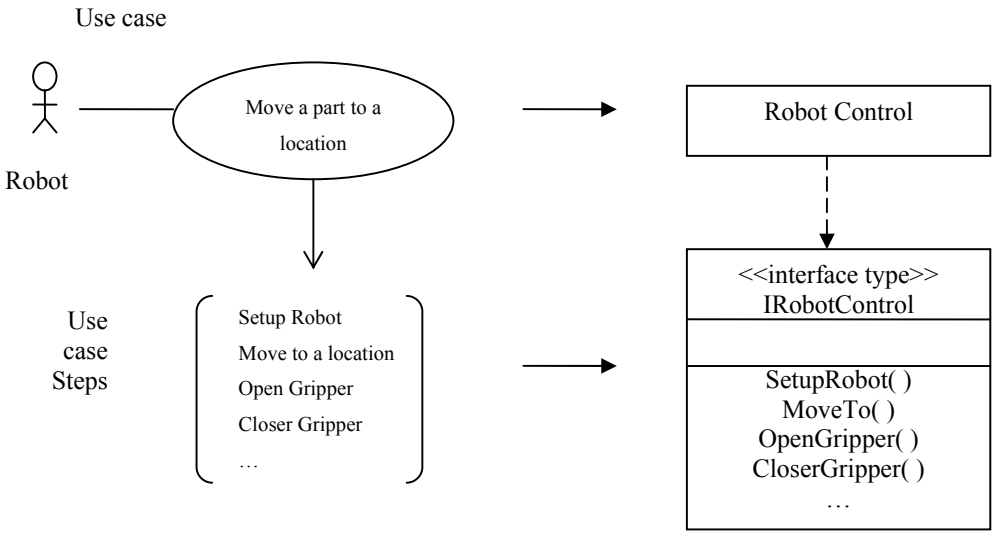


Figure 3.9 Use cases map to system interfaces

The component interaction activities decide how the components work together to deliver the required functionalities. The component interaction stage examines how each of the system operations will be achieved using the component architecture. It uses interaction models to discover operations on the business interface. The management of references between component objects needs to be carefully designed so that dependencies are minimized and referential integrity policies are accommodated.

The final component specification stage is to be precise about the usage and realization contracts. Look at interfaces and component specifications individually and add more details. Component specification is the stage where the detailed specification of operations and constraints takes place.

The outputs from specification workflow are used in the provisioning workflow to determine what components to build or buy, in the assembly workflow to guide the correct integration of components, and in the test workflow as an input to test scripts.

3. Provision

The provisioning workflow ensures that the necessary components are made available, either by building them from scratch, buying them from a third party, or reusing, integrating, mining, or otherwise modifying an existing component or other software. The provisioning workflow also includes unit testing the component prior to assembly.

4. Assemble, test and deployment

Once the element models have been developed they can be tested individually or in groups. Once the designs have been thoroughly tested, the executable design model is the detailed specification.

The assembly workflow takes all the components and puts them together with existing software assets and a suitable user interface to form an application that meets the business need. The component specification can then be mapped to a platform specific model such as CORBA components, EJB, or COM+/.NET.

3.3.2 Types of Main components

A major effort is required to identify appropriate partition of functional modules in a modular design. The component identification is considered as the first stage of the component specification workflow. In Cheesman and Daniels' methodology, components are identified by analyzing business requirements with business concept model and the use case model. Generally, the identification of main software components in manufacturing control systems is easier and straightforward than most business processes.

The main software components can be identified according to major physical equipments and main system functionalities in the proposed control architecture. Some of the common software components include:

1). Decision Maker Module

Decision Maker Module acts as the "brain" of the whole control system. The control algorithms are mainly implemented in this module. Based on the information (order info, part type info, etc) from outside (other software modules), decision maker module provides decisions on scheduling and control of the whole production system. The decision component listens for input events. Based on its observation, it conducts the control logic operation, and fires control action. Optimal strategies can be implemented into this module before processing a production order. Once an exception occurs, agent mechanism in Exception Handling Module is then activated, and starts to make decisions according to new environment. The system control logic architecture is then changed from hierarchical to heterarchical.

2). Machine Control Module

Machine Control Modules have one-to-one links with associated physical machines. The main functions of a machine control module are to take decisions from decision maker module or Exception Handling Module and send instructions to corresponding machine controllers. The physical equipments can be observed in any typical FMS/RMS include material processors, material handlers, and material transporters, etc.

Although the functions within different manufacturing system may change drastically, the equipments and their basic operations do not. By partitioning the system along those lines we are more likely to develop components which can be reused on future designs. The generalized equipment module also allows the standardization of driver interfaces. For example: a standard CNC machine is capable of downloading machine files from computer, receiving commands and providing status output. This component offers its users the capabilities of executing a set of part programs and querying for the status of the machine center. The center will be busy when executing a part program and idle otherwise. The details associated with the execution of a part

program are not of interest to the use of the component, and, are hidden in the component's internals.

Other simple equipments such as sensors and buffer location etc that only perform simple functions can have their corresponding software parts implemented in an auxiliary equipment control module.

3). Component Directory Module

The Component Directory Module offers a standard mechanism to classify, catalog, and manage components and their services. The software components can search for "services" from this module by its yellow page function.

A library of components is developed which can be searched for those components matching the requirements of a system. A class hierarchy assists in guiding the designer in finding and choosing appropriate components from the library for a new system. If no appropriate components exist then a new component class is created and placed in the class hierarchy.

4). Configurator Module

RMS can be reconfigured reliably according to the changes without incurring high costs. When a system is to be changed, there are usually numerous configurations to choose. How to select the "right" configuration system is an important issue. The duty of the configurator module is to choose a suitable configuration for a system during its reconfiguration.

5). Other Software Modules

Due to the modular nature of the component-based framework, different software components can be easily added. For different scenarios, other modules such as database module, state module, simulation module, and HMI module can be included.

Different software components can be easily added to component-based control system. The proposed control architecture can be extended to be different enterprise-level software architecture by adding other specific software modules (e.g., logistic, supply chain, CAD, etc)

3.3.3 Component specification in CICA

The process of component specification is adapted to fit manufacturing environment based on the methodology Cheesman and Daniels (2000) developed and the studies conducted on Objective-oriented development for manufacturing control systems (Bruccoleri and Diego, 2003).

In this work, the component-based system is specified in two levels: system level and component level. In the system-level (Figure 3.10), a graphic diagram shows basic information of components in system including component names and their relationships.

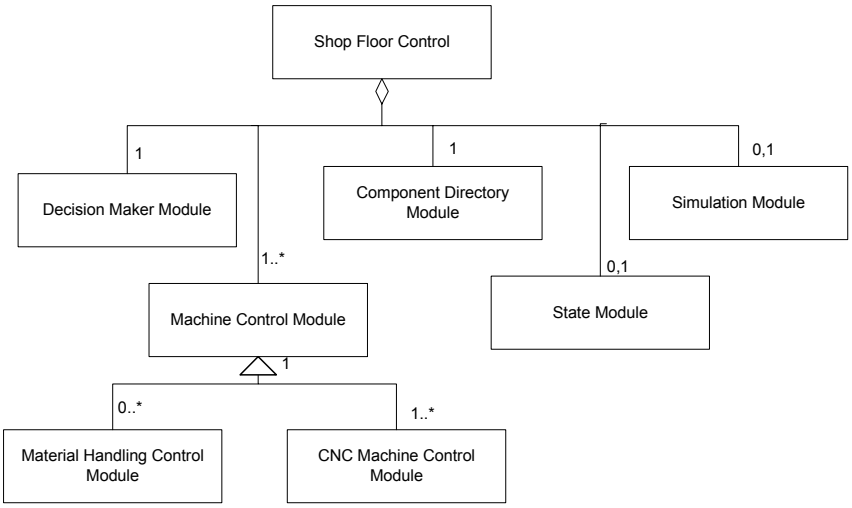


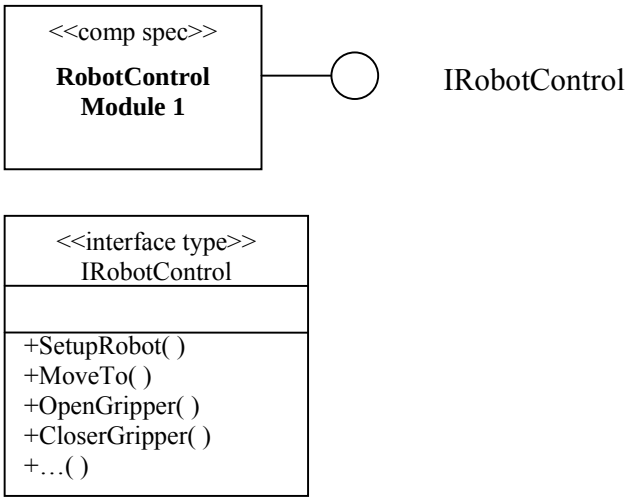
Figure 3.10 System-level UML description

As UML representations are used in different places in remainder of this report, a short explanation of the syntax is given. In Figure 3.10, every rectangle represents a software module in the system. Each line represents a relationship. Depending upon the symbol in the line (arrow, diamond, or no symbol), the line refers to a different kind of relation. The aggregation relation or "*has-a*" relation is represented by a line with a diamond. The specialization relation or "*is-a*" relation is represented using a line with an arrow. The "association"-relation is represented using a normal line. Both ends of the line usually have numbers associated with them. This is called the cardinality (0..*, 1..*, 0..1, 0, 1, or unspecified), and it represents the number of the modules involved in this relation.

A full definition of all UML notations can be found in D'Souza and Wills (1998b) and Fowler (2003).

In the component-level (Figure 3.11), each component is specified to necessary level of precision. The component specifications name interfaces that a component must implement and operations provided by each interface. The software interface of the components should be general and parameterized whenever possible. A software interface is termed general whenever it suppresses implementation-specific features, and reveals only the details required in controlling the underlying component. For example, the software interface of the robot software component does not reveal either the type of robot used or the details associated with implementing its services. Instead, this interface provides services that are general enough to be carried out by a large class of robots. Hence, any one of these robots could be enclosed by this component without affecting other components in the system; i.e., only the intervals of the component need to be changed in order to interface with the particular component enclosed.

Cheesman and Daniels (2000) used Object Constraint Language (OCL) to formalize the contractual conditions of an operation in an interface. It was used to define any constraints that it requires between these interfaces, and any constraints on the implementation of its operations. OCL, part of the UML, is based on proprietary designed format and can only be processed by special tools. Another drawback of OCL is that it is very hard for human to understand though it can be very precise. XML-based specification language can solve such problems. OCL specification can be “translated” into XML with information content yet structures remain unchanged. Such simple change can make the specifications machine-readable without specified tools. Thus, Document Type Definition (DTD), or XML schema, the XML-based specifications of software components can be easily located, discovered, or retrieved over Internet by other users. All operations of each interface are specified using XML with pre and post conditions. Such conversion from OCL to XML makes the specification easy to understand, however without losing accuracy.




```
<?xml version="1.0" encoding="ISO-8859-1" ?>
 <interface>
     <type>IRobotControl</type>
   …
     <service>OpenGripper
        <input>
                <parameter type="RobotId">Robot1</parameter>
        </input>
        <precondition>
              <RobotState>Activated</RobotState>
        </precondition>
      <postcondition>
              <GripperState>Open</GripperState>
        </postcondition>
      </service>
    ...
</interface>
```

Figure 3.11 Component-level UML descriptions

The graphic descriptions of software components and their relationships can also be equivalently represented by plain text. The UML diagrams visualize the modeling of software architecture, while the plain-text representation can be easily understood and distributed by computers. RMS designers, with system requirements, can search for software components that meets these requirements.

3.3.4 XML-based message format and web service based protocol

The evolution of component plug-and-play concepts and principles has recently led to the Web services paradigm. Web services are the set of protocols by which services can be published, discovered and used in a technology neutral, standard form. The web services paradigm employs the concept of service and the strategy to expose system capabilities to consumers as services.

The notion of a service is an integral part of component thinking. A component by definition provides a service to its environment. The service-oriented thinking represents a promising paradigm for building current and future business information systems. When combining components from different vendors, cost and effort often increase significantly due to the difficulties in exchanging data, coordinating subsystems or comparing the results. Web services standards provide broad interoperability among different vendors' solutions. The W3C's Web Services Glossary defines a web service as "a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format. Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards." (W3C-WS Glossary, 2004) As defined, the basic elements of the Web services protocol stack are the standards for interoperability, XML, SOAP, WSDL and UDDI that provide platform-independent communication of software resources across the Internet (W3C, 2004; Newcomer, 2002).

XML that states for eXtensible Markup Language is a flexible markup language representing a text-format layer that can be used to describe any type of data in a structured way. XML Schema defines the structure, content, and semantics of XML documents. Web Services Description Language (WSDL) provides a formal, computer-readable description of Web services. WSDL provides a way for service providers to describe the basic format of web service requests over different protocols or encodings. WSDL is used to describe *what* a web service can do, *where* it resides, and *how* to invoke it. The Simple Object Access Protocol (SOAP) is XML-based protocol that allows enables communication among Web services. It defines a set of rules for structuring

messages that can be used for exchanging information between peers in a decentralized, distributed environment. Universal Description Discovery and Integration (UDDI) directory is a registry of Web services descriptions. UDDI provides a mechanism for clients to dynamically find other web services. Users can search public UDDI directories (like the Yellow Pages for Web services) available on the Internet, or private directories to find available Web services. All these protocols (Figure 3.12) are not tied to any particular operating system or programming language. As such, they can be used as backbone for developing open, standards-based, and vendor-neutral reconfigurable system-level control framework.

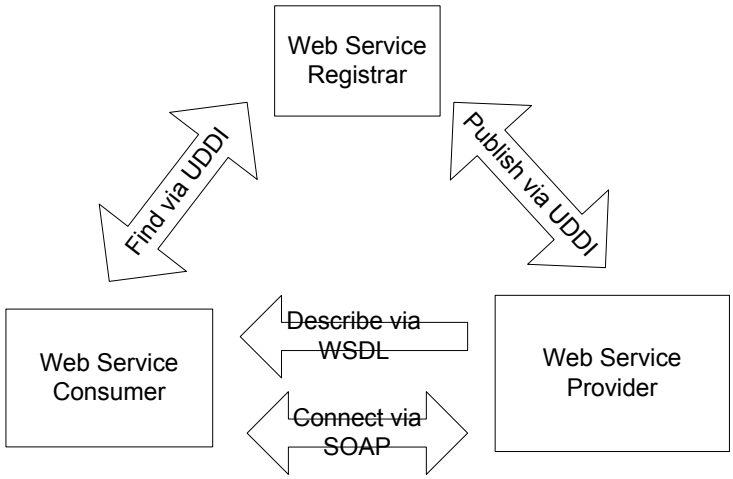


Figure 3.12 Three Basic Web Service Protocols

Based on open XML data format and open protocols, the framework proposed in this research has a higher-level of abstraction than current majority of component-based control software studies. The framework does not specify how software components are implemented and imposes no restriction on how components might be combined. In this framework, a component is a black box and it exposes only services to outside. A service is defined in Gorton and Liu (2004) as a coarse-grain, discoverable interface and associated implementation that acts as a façade for other applications, and typically interacts with other software systems in a loosely coupled fashion. Any changes of a component are made within the boundary without affecting other components. All the components will be loosely coupled and they are connected by open data format. Services

only communicate with one another by platform-neutral, standardized XML format messages.

Based on web services protocols, generic component interfaces is made possible. Such interfaces take XML strings as parameters, which conforming to a certain XML schema. The result is again specified in XML. Figure 3.13 is an example message in SOAP envelop.

```
< ?xml version = »1.0" ?>
<soap: Envelope xmlns:env = "http://www.w3.org/2002/06/soap-envelope">
 <soap: Header>
        <n:alertcontrol xml:n="http://www.fmslab.ise.vt.edu/alertcontrol">
              <n:priority>1</n:priority>
              <n:expires>2006-08-22ET14:05:00</n:expires>
        <n:alertcontrol>
 </soap:header>
 <soap:Body>
        <m: alert xmlns:m="http://www.fmslab.vt.edu/alertcontrol">
              <m:msg>job completed</m:msg>
        </m:alert>
 </soap:Body>
</soap:Envelope>
```

Figure 3.13 a SOAP message example

Owing to the lengthy XML format, SOAP can be considerably slower than rival middleware technologies such as CORBA, though this may not be an issue when only small messages are sent. Also, there are also two issues need to be addressed. First, there should be a clear definition of message types. Second, a mechanism should be available to handle the messages, so that messages can be passed to the specified classes and corresponding message handling function triggered.

The web services protocols are not necessary required in CICA. The web services protocols can be used when the interoperable problem among heterogeneous components

exist. The web service can be used as an adapter for the access to a service, which is implemented on other types of platform. And as the open protocols exist, the specifications of software components are more important than how the components are implemented. Figure 3.14 shows a generic component-based control framework based on web-services protocols.

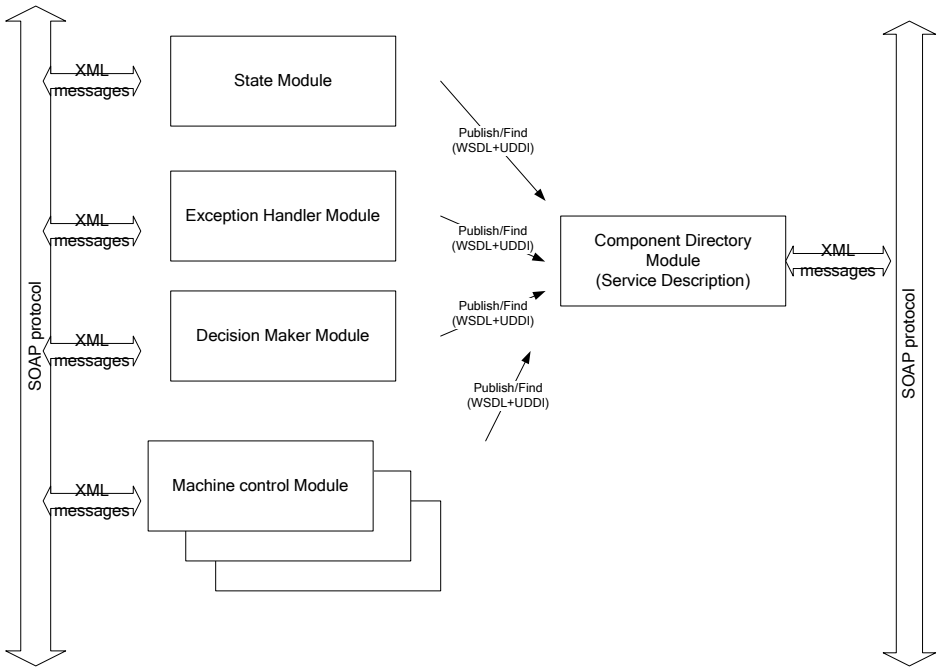


Figure 3.14 Component-based architecture based on web-services protocols

3.4 Experimental FMS and the Walli3 system

The necessary components for a component-based development are made available mainly from three sources: 1). Build component in-house; 2). Buy components as commercial off-the-shelf (COTS); 3). Wrap existing legacy assets in a component-like structure. As most today's control systems are built with inflexible structures, how to adapt legacy systems to integrate with components is an important issue.

The motivation of this research is originated from a laboratory enhancement project in the FMS research lab of the Grado Department of Industrial and Systems Engineering at Virginia Tech (the systems is now in the Flexible Manufacturing and Lean

Systems Lab at University of Texas at San Antonio). The project is aimed to make the control software of an existing FMS reconfigurable for supporting the better researches in the lab.

The flexible manufacturing cell consists of a cell controller, 2 CNC Lathes, 2 CNC Mills, 2 material handling robots and peripherals such as index tables, belt conveyors, and automatic part dispensers (Figure 3.15). WALLI3 is the cell control software that provides the overall supervisory control for the Robots, CNC Machines and peripheral and sub-devices. The software integrates all system components to an automated workcell.

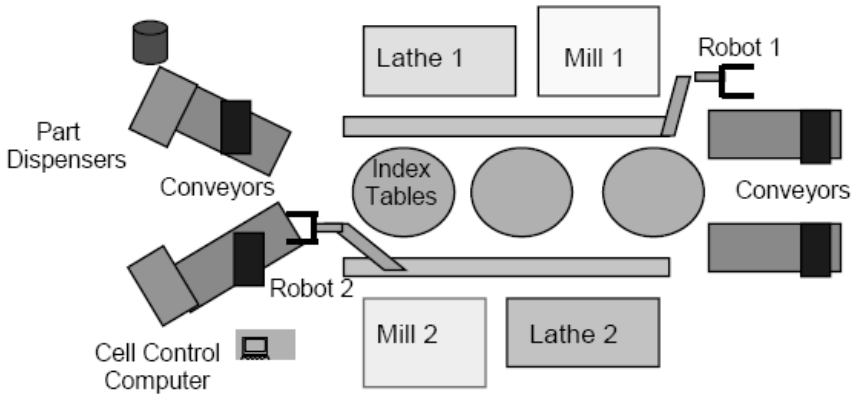


Figure 3.15 The layout of the laboratory FMS

The Walli3 control software is the main source of the inflexibility of the FMS, because of its tight coupling of the functions, complex control logic, and imbedded scheduling as normally seen in traditional FMS control software. Like most FMSs, the laboratory FMS is sold as turnkey systems by integration vendors. Control software is usually developed and supported by their subcontracting software houses. FMS users must contact FMS vendors and their software subcontractors for control software modification when the physical system and production requirement are changed. Such modifications are usually time-consuming and can be very expensive.

A more desirable solution would be to allow FMS users to modify the control software easily when needed and most control modules can be reused without re-programming each production setup from scratches. The component-based intelligent

control architecture proposed in this research is a promising design to overcome the inflexibility of FMS control software. With CICA, the control software of a manufacturing system is made up of multiple software modules. The machining centers in manufacturing systems are no more controlled by supervisory control software, but by a set of dedicated software control modules. Such control software will be easy to construct and modify.

3.7 Summary

In this chapter, an overview of proposed Component-based Intelligent Control architecture (CICA) has been discussed. The component-based technologies are used to enable reconfigurable software architecture for CICA. The main software components in CICA include components corresponding to main system functionalities and major physical equipments. By employing emerging open data format and open standards, the implementation and interoperability of components can be made much easier. For this reason, the specifications of software components in CICA are more important than how the components are implemented. The UML-based component specification makes the control architecture easy to be designed, distributed, and reconfigured. A component specification workflow is presented in this chapter. To test the effectiveness of proposed control architecture, a table-top FMS system is used as a testbed. In next two chapters, how intelligent agents are used to handle exceptions and bring reconfigurability into control logic of CICA will be presented in greater details.

Chapter 4. Agent-based exception handling

4.1. Overview of agent-based exception handling

The advantages of the proposed control architecture for RMS include scalability, reconfigurability, reusability, adaptability and fault-tolerance. The first three features are mainly provided by using component-based software technology, while the last two are achieved by using multi-agent paradigm.

Agent technologies have rapidly gained popularity in manufacturing area in recent years, largely due to their ability to allow manufacturing production systems to promptly react to dynamic demand variations and unexpected events. Agent-based systems are flexible and self-adaptive themselves, where agents may be added or replaced freely. Therefore, the system configuration can be continuously changed to accommodate changing requirements.

Traditionally, manufacturing systems are commonly established with centralized hierarchical control software. Centralized solutions are generally more efficient in a deterministic environment. The agent-based methodologies have advantages in non-deterministic environment, however, with the price of high communication load and unpredictable global performance. In this component-based intelligent control architecture, agents are activated only when exceptions occur. Compared to a pure agent-based scheduling system, such exception handling mechanism avoids the risk of missing the global optimal point and lessens system burden when system is operating under normal condition.

The agent-based exception handling process is the focus of this chapter instead of types of exceptions. In this research, an exception is defined as an event occurs in a manufacturing system, which can affect the performance of the manufacturing system and is not considered in normal scheduling of production processes. Exceptions can be machine breakdowns, material shortage, changes in job priorities, changes in job arrival rates, dynamic introduction of new jobs, etc. The real-time exception handling process mainly concerns providing manufacturing systems with scheduling decisions after the

exception arises. The agent-based exception handling is composed of following consecutive stages:

Exception detection and report

Error detection deals with the detection of exceptions, either by an operator or by the control program. Clearly, in order to prevent their propagation through the system, exceptions should be detected as early as possible. The efficient ways of detecting exceptions include: use of sensors, time-out, state check, supporting systems, etc (Beek 1993). Once an exception is detected, a report with information of exception type, happen time, and source, etc., is sent to the agent management module for further treatment.

Exception diagnosis and impact evaluation

Since a single exception can cause many exceptions to spread through the system, exception diagnosis deals with the determination of all exceptions, which are related to the detected exception. Impact evaluation makes decisions on whether exception handling is needed or not by evaluating impacts of detected exceptions. When exceptions are mis-reported or impacts on manufacturing process are not significant (effects stay within predefined tolerance), exceptions are recorded and only alerts are sent to operators without activating autonomous agent-based exception handling.

Agent activation and exception handling

When the available control decisions are unable to satisfy the unplanned situation in manufacturing system after an exception is detected, agent-based exception handling will be activated. Staff agents will organize negotiations based on the exception information. Staff agents announce tasks to feasible resource agents, and award a contract to the resource agent with the best bid. Contract Net Protocol is used in facilitating the negotiations.

Agent deactivation

Since, an agent-based system is unlikely to generate global optimal decisions, and performance is usually unpredictable, centralized control decisions should take over control whenever the system is recovered (e.g., after fixing broken machines) or a new stable state of the system is reached (e.g., introduction of new orders become constant).

The basic agent-based exception handling process is shown in Figure 4.1.

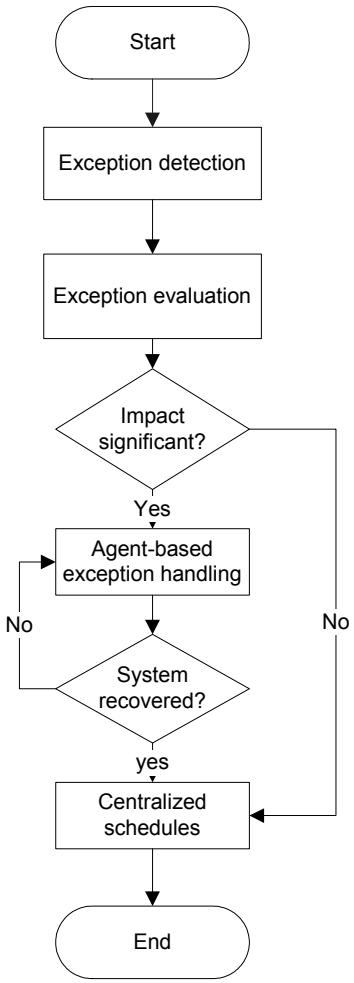


Figure 4.1. Agent-based exception handling flowchart

Just like other functions in system being implemented as software components, the exception-handling function is implemented as a software component in CICA architecture. As shown in Figure 4.2, the exception-handling component includes interface, agent system, yellow page service, event list and knowledge base. The agent

system is the core of this component, which is responsible for providing scheduling decisions in the event where exceptions occur. All the other functions are designed to assist agents to make decisions. The Yellow Page Service records all agent information and provides list of feasible service providers when agents are looking for service. The event list keeps record of events happened and those that are likely to happen in system. It is used to help agents in making decisions when agents have conflicting interests. More details about the event list will be described later in this chapter.

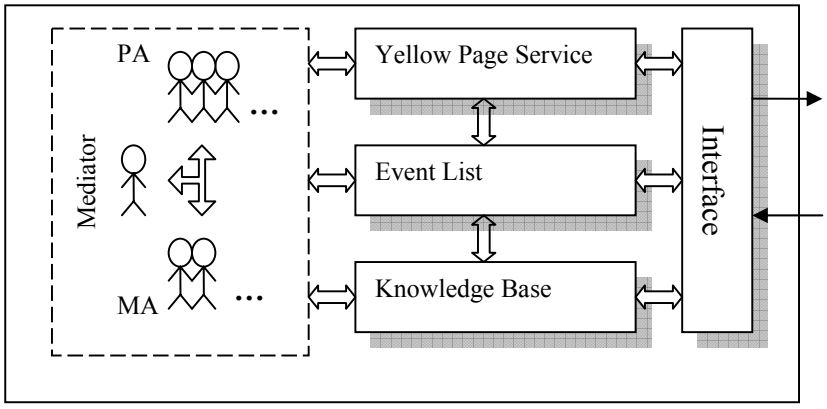


Figure 4.2. Internal structure of exception handling module in CICA architecture

The rest of this chapter describes how the agents work together to make scheduling decisions after the agent-based exception handling mechanism is activated.

4.2. Agent-based decision-making process

The shop floor control basically emphasizes sequencing and scheduling of jobs. Scheduling is an optimization process where limited resources are allocated over time. It is difficult and complex particularly when it takes place in an open, dynamic environment. In such an environment, duration and service time may not be known precisely, machines may become unavailable and additional resources can be introduced at any time. The starting time and the processing time of a task are also subject to variations. Because of its highly combinatorial aspects, the scheduling problem is computationally complex, with most scheduling problems being proved as NP-complete.

Traditionally, manufacturing systems are commonly established with centralized hierarchical control software. This type of approach is most suited for a completely

deterministic environment. The centralized systems have limited ability to model flexible manufacturing systems where the parts (jobs) can follow alternative routes available. It is not flexible enough to cope with the dynamic changes of manufacturing systems.

Centralized solutions are generally more efficient in a deterministic environment. However, distributed computations are sometimes easier to understand and easier to develop, especially when the problem being solved is itself distributed. Distribution can lead to computational algorithms that might not have been discovered with a centralized approach.

Recently, agent technology has been considered as one of the most promising approaches for designing and implementing manufacturing control systems. In contrast to traditional manufacturing systems using centralized control architectures, agent-based distributed manufacturing systems utilize decentralized architectures. MAS break complex problems into small and manageable sub-problems to be solved by individual agents co-operatively. Agents are autonomous entities and they are assigned with one or more goals, and it is their own responsibility to complete their goals within time limit. To achieve this, they can observe and act upon their environment, and they can communicate and cooperate with other agents. Unlike static programs that rely on pre-assumptions on the environment or their cooperation partners, agents are considered individuals who are able to react appropriately (and often intelligently) to change. This makes agent systems very flexible and reconfigurable. The characteristics of an agent such as autonomous, reactive, cooperative, modular, and of a multi-agent system such as extendible, flexible and reconfigurable, have proven great advantages for their use in shop floor control.

In this research, agents are designed to make dynamic, real-time scheduling decisions once exceptions occur in the system. Agents that represent physical shop-floor components such as parts, machines, and material handling equipments are considered as basic agent types. There are also auxiliary agent types such as mediator agent, which are able to help negotiation among agents. All of the tasks related to scheduling in this system are then conducted by autonomous agents that are capable of interacting and negotiating with each other to bid for jobs and make decisions. A desired global behavior is achieved by coordination/cooperation among agents. Despite the numbers of literatures on agent-based scheduling, some issues in area of agent-based scheduling are not or less

exploited. Three of such issues are emphasized in this research: consideration of precedence constraints in production process, reduction of communication overheads, and local decision conflicts resolution based on game theoretical models.

For ease of description and without loss of generality, the global objective of the scheduling problem studied in this research is to minimize the total completion time. Based on the studies in the field of DAI, there is no explicit way to relate the local objectives of a decision entity to global system performance. Since, each agent only attempts to achieve its own goals without considering the global goals, there may be a contradiction problem between the local objective and the overall system performance. Simulation is a common technique to evaluate the performance of agent systems.

An agent-based scheduling system usually can be modeled as a product-driven approach or a resource-driven approach. As the name implies, in the product-driven approach the agents drive the production. The part agent, playing a central role, will make sure that the product, for which it is responsible, is going to be manufactured through the shop floor. In the resource-driven approach, machines actively look for work to be performed, thus known as resource-driven. This approach makes it possible for machines to maximize their capacity utilization. When the two approaches are compared, the product-driven approach generally has more advantages over the resource-driven approach. The product-driven approach has been proven to be more robust, flexible and fault tolerant. More comparisons of the two approaches can be found in Berger *et al.* (2002). In this research, a product-driven bidding-based approach is used. A bidding cycle starts when a part enters the system or completes an operation. Part agents broadcast requests to all feasible machine agents, while machine agents respond with an estimate of the time it will take to complete this processing. Part agents then collect proposals and choose the best one based on how likely the proposal is to help them satisfy their due date commitments. Each part agent repeats this process until all of its operations are completed.

The negotiation among autonomous agents in this research is arranged via a bidding scheme based on Contract Net Protocol. Contract Net Protocol is the most widely used cooperation mechanism in manufacturing domain. Contract net protocol is quite simple and can be efficient, nonetheless, there are some well-known problems related

with it, such as high communication load and local interest conflicts. An extended version of the general contract net protocol is proposed in this research. The details of extended contract net protocol are described later in this chapter.

The following sections of this chapter outline the construction of basic agent types, their decision strategies and interaction protocols.

4.2.1 Main agent types

In this research, there are three main physical agent types: Part Agents (PA), Machine Agents (MA) and Material Handling Agents (MHA) that are associated with individual parts, machines and material handling equipments, respectively. Part agents (PA) and resource agents (RA) are usually utilized to represent the two basic concerns exist in manufacturing systems: 1) resource aspects such as driving the machine at optimal speed and maximizing its capacity, and 2) product and process related to be performed to achieve a good quality product. There are also auxiliary agent types such as the mediator agent to support agent decision-making processes (Figure 4.3). The mediator and sub-mediator agents have the function of filtering and re-directing information between the part agents to resource agents. The mediators and sub-mediators route messages among agents based on the content of the messages, and coordinate the control of multi-agent activities. In this research, the mediator agent is also used to resolve the conflicting interests among different PAs during a period of time.

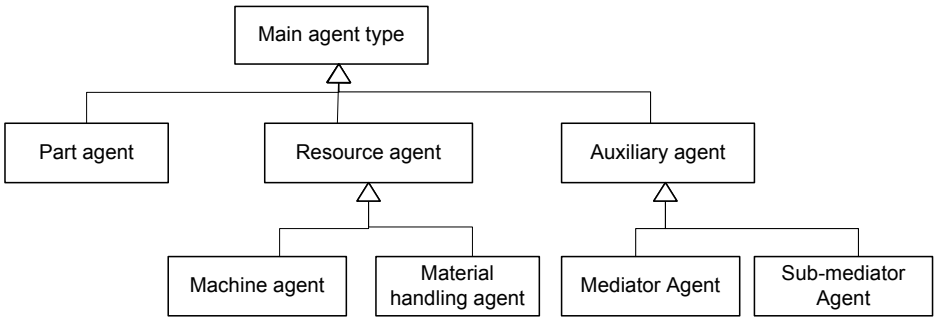


Figure 4.3. Basic agent types

Each part agent (PA) is a piece of software program associated to the corresponding part circulating in the system. They are associated to each individual

product and control the processing of tasks applied to the products, along the whole production cycle. Each PA also encapsulates the information necessary to manage the processing of individual parts such as the part's process plan, machining requirements, processing status, and due date, etc.

PAs are in charge to autonomously select the best machines for each operation. The goal of the PA is usually to obtain the requested operations by minimizing part flow time and the cost of service.

Main attributes of a Part Agent include:

- Part ID
- Part Type
- Due Date
- Process Plan
- Priority
- Current location

Parts from same part type have same routings. The part process plan contains the sequence of operations to be performed. Each part enters the manufacturing system with a due date. The objective of a part agent is to complete processing before its due date. The location of a part is used by material handling agents to determine transportation time. The internal structure of a part agent is depicted in Figure 4.4.

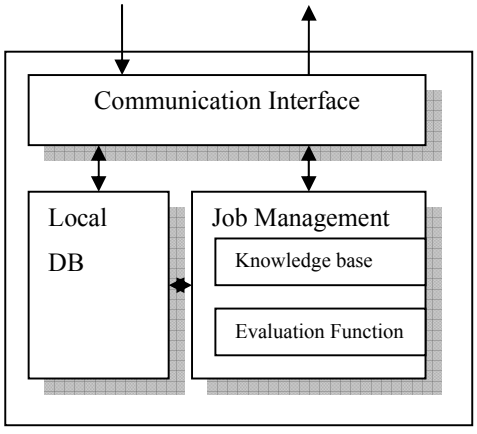


Figure 4.4 Internal structure of Part Agent

Machine Agents manages the individual workstations in the manufacturing system and contain information concerning the workstations' machining capabilities and status. Machine Agents correspond to machine tools (CNC, wash, etc). A Machine Agent keeps states of the corresponding machine tool and is responsible for the decisions made related to the selection of parts for processing. The attributes of the Machine Agent include machine status and the parts waiting in the buffer.

The main attributes of a Machine Agent include:

- Machine ID
- Machine Type
- Buffer limit
- Number of available buffer spaces
- Parameters (speed factor, etc)
- Location
- Total reserved operation time

All the machines are grouped together based on their processing capabilities. A part with a particular process requirement can be processed by any one of the machines in the groups, which can perform the given operation. The location of a machine in the manufacturing system is used for material handling agents to calculate transportation time. The buffer limit is the maximum number of parts, which may be placed in a queue for a machine. If the buffer limit is reached, the machine can no longer accept a part. The common objective of a machine agent is to maximize its utilization rate. The internal structure of a machine agent is depicted in Figure 4.5.

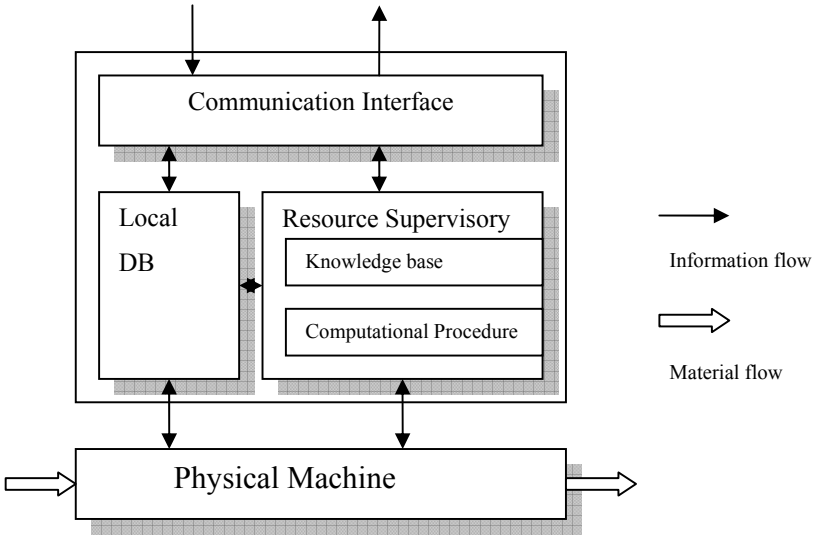


Figure 4.5 Internal structure of Machine Agent

In most research, the material handling devices are commonly neglected or assumed to be always available and transportation times are included in machining times. However, the load/unload time and transportation time of material handling devices could be significant and need to be considered as critical factors in real cases. Material handling agents participate negotiations with material handling information.

The main material handling devices types include: robot, AGV, etc. In order to incorporate the transportation time in agent decision-making process, the locations of parts, machines and material handling devices need to be is determined on the shop floor.

The main attributes of a material-handling agent include:

- Material handling device ID
- Material handling device Type
- Parameters (speed factor, etc)
- Current location
- Total reserved transportation time
- Last reserved location

The internal structure of a material handling agent is very close to a machine agent depicted in Figure 4.5.

Resource mediator and sub-mediator agents play important roles in the operation of multi-agent scheduling processes in this research. A mediator agent fulfills two main functions. First, it acts as the interface or facilitator of a group of machine agents. Second, it can facilitate and supervise the interactions between agents under its custody. It can be used to resolve the conflicts in concurrent multi-agent negotiations.

All requests from part agents to resource agents are first sent to the resource mediator. The resource mediator will route each request to right resource agents. By treating different negotiation scenarios differently, the mediator agent can help reduce number of communications and resolve conflicting interests among PAs.

The part agent requests all the resource that can perform each operation, and therefore, each resource can receive several requests and each operation can be "negotiated" by several resources. The decisions made by one agent will affect other agents' decisions, traditional agent-based systems rarely considered this problem. The mediator agent can help facilitate the concurrent negotiations between multiple PA and MA. A microeconomics model - game theory is used to analyze and resolve the potential conflicting interests among PA when they are competing for the same MA. A resolution methodology is made based on the game theoretical matrix. More details of this can be found in the later part of this chapter. The internal structure of a mediator agent is depicted in Figure 4.6.

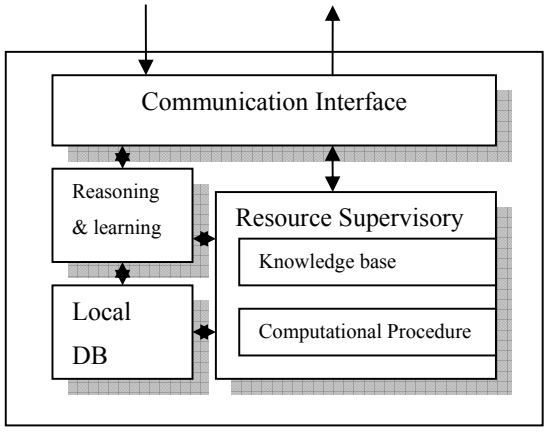


Figure 4.6 Internal structure of a mediator agent

4.2.2 Stepwise agent decision-making process

In agent-based scheduling systems, agents only use local information available at decision time to make decisions. In a dynamic environment, a scheduling decision can be obsolete in short time. To have better reactivity and adaptability in high dynamic environment, a single-step production reservation (Reaidy *et al.* 2006) is usually employed in multi-agent scheduling systems. A stepwise decision process is utilized for agent systems in this research. There are three steps in the stepwise scheduling decision process. In the first step, PA selects one operation out of its process plan as the next operation by following a "next operation selection rule". Since PA considers only one operation in each bidding cycle, communication caused by agent negotiations will be greatly reduced. In the second step, after deciding the next operation to be processed, PA acquires MA and MHA to process it by negotiation. The decision is made through cooperation among agents via a bidding scheme based on contract net protocol. In the third step, MA selects a part to process from its buffer based on new shop floor information (compared to the information used in step 2). Due to the stochastic and dynamic nature of a shop floor, the actual events in the shop can be considerably different during the time a part waiting in a buffer. Therefore, step 3 can be used as a corrective action to enhance the degraded performance of the original decision.

In summary, the stepwise decision process includes:

- Step 1: part agent selects the next operation for the part.
- Step 2: part agent sends request to candidate machines for the selected operation and makes decision to choose the machine with shortest sum of machining time, committed time of queued parts and part transport time.
- Step 3: Machine agent select parts in buffer based on updated shop floor information.

Step 1: Next-operation selection for PA.

One widely neglected factor when using agent-based scheduling is precedence constraints in a part's process plan. Traditionally, scheduling problem is often formulated as a general optimization problem with constraints. The precedence constraint is one of

the three main types of constraints in a mathematical programming to solve an optimal scheduling problem: resource capacity constraints, disjunctive constraints and precedence constraints. The precedence constraint restricts operations' sequences. As in agent-based scheduling process, each contract occurs between one PA and one MA, the first two constraints can be naturally satisfied. However, the precedence constraints have been commonly overlooked in most research. Part agents usually broadcast their call-for-proposals to request bids for all necessary operations. Since, there are typically precedence relationships between operations, not all operations can be assigned to machines in one bidding cycle. The unnecessary process requests from all operations will cause big communication burden and conflicting issues. To solve the problem, only one operation will be selected from a group of operation candidates at beginning of each negotiation cycle to call for proposals.

A precedence constraint is usually represented by an arrow between two operations in a precedence graph. If operation O_i cannot begin its execution until operation O_j has completed its execution, we write $O_i \prec O_j$. In this case O_j is said to be a predecessor of O_i, and conversely, O_i is a successor of O_j. Precedence constraints force operation O_j not to be started before all its immediate predecessor operations have been finished.

Because of the existence of precedence constraints, it is inefficient to let all PAs send out requests in each bidding cycle. A PA is "qualified" to send out requests for one operation only when the predecessors of the operation are all finished. There could be a group of such operations. To further reduce the communication burden and conflicting chances, only one operation is selected from the "qualified" operations based on certain criteria. The group of "qualified" operations are named active operation group in this research.

Active operation group is defined as the group of operations in the process plan of a part, which has no unfinished predecessors at the time.

At any time t_{now}, the active operation group includes operations in the set:

$$A_t = \{O_i: t_j < t_{now}, \ \forall O_j \prec O_i\} \quad (4.1)$$

where t_j is the start time of operation O_j;

$j=1,2,...n;$

n is number of operations in PA's process plan.

The active operation group of a part agent is updated at the beginning of each bidding cycle. Operations that are immediate successors of the last operation will be added to the active operation group while at the same time the last operation is deleted from the active operation group. One operation will then be selected from active operation group following "Next Operation" selection criteria:

- In current active operation group
- The operation requiring the same machine with its predecessor operation will be selected with high priority
- Otherwise, select the operation requiring a machine with least workload committed (number of parts to produce or total cumulative operation time)
- Randomly select the one when multiple operations qualify via the above selection procedure.

By definition of active operation group, and "Next Operation" selection criteria, there is always only one operation for each PA in each bidding cycle. Such a design can significantly reduce the number of communication messages and chances of communication conflicts in PA-WA negotiations. Figure 4.7 shows a simple example of product process plan.

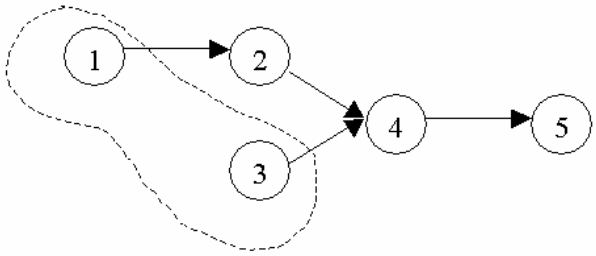


Figure 4.7. "Active operation group" update in a simple production process plan

When a PA enters system with the process plan as shown in Figure 4.7, the active operation group will be initiated with operations {1, 3}. If the operation 1 is selected as the first operation based on the next operation selection rule, the active operation group will be updated to {2, 3}, in which operation 2, 3 have no unfinished predecessors. The process proceeds until the part leaves system.

Step 2: Negotiation between PA and MA

In this step, local scheduling decisions are made through negotiations among scheduling agents. After choosing the next operation to be processed in last step, PA needs to make the decision to choose machines and material handling equipments to carry out the selected operation. A model of negotiation among scheduling agents is presented in this section. The purpose of negotiation is to reach an agreement about the assignment of operations to machining and material handling equipments.

Based on the FIPA (the Foundation for Intelligent Physical Agents) contract net protocol (Figure 4.8), the PA decision algorithm works as follows. Let *Ms* be the set of workstations (machines) in the manufacturing system. Suppose that a part completes an operation on a workstation. As soon as the part is ready for a new operation, the associated PA selects the subset of workstations $M \subset Ms$ that can execute the next operation (according to yellow page service in the exception handling module). Then the PA sends a service request to each of the MA associated to the m machines in the set *M*.

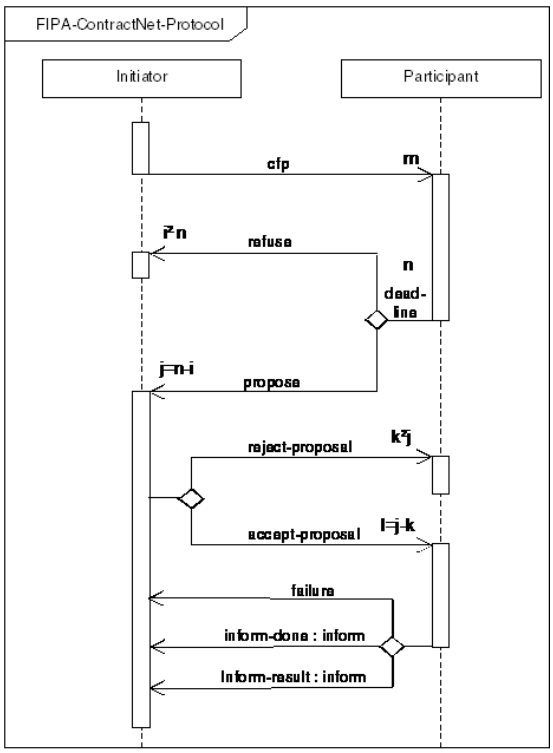


Figure 4.8. FIPA Contract Net Interaction Protocol

The *m* MAs consider the request and estimate completion time for the operation. Out of these responses *j* MAs propose contracts with estimated completion time while $i = n - j$ MAs refuse to propose. The PA, which initiates the negotiation, evaluates the proposals and selects the MA with the earliest estimated completion time to perform the task. The remaining $k = j - 1$ proposals are rejected. Once the PA accepts a proposal, the winning MA acquires a commitment to perform the task. If the PA identifies an accepted offer, it immediately sends a reservation request to the offering workstation (and MA will update its scheduling records). On the contrary, if no offer is satisfactory, the PA starts renegotiating. After the MA has completed the task, it sends a completion message to the initial PA. In the case that a participant is unable to complete the task, a failure message will be sent. This message will usually state the reason for failure, such as a transportation accident or a machine failure.

The estimated completion time $T_{ec}(i)$ for an operation estimated by MA_i is based on

- Pallet-Fixture preparation time t_p
- Transportation time t_t
- Load/Unload time t_l
- Tool change t_c
- Machine processing time t_m

The estimated completion time

$$T_{ec}(i) = max\,(t_{res}(i),\, t_t(i)) + t_m(i) + t_c(i) + t_p(i) \qquad (4.2)$$

Let set H_i be the set of material handling equipments which are able to transport a part to machine *i*. To estimate transportation time, MA needs to know which material handling equipment in set H_i is used. In this research, the material handling equipment with the least total reserved transportation time is selected.

$$t_t(i) = \min\{\, t_{res}(h) + t_l(h) + [dis(L_p, L_{res}(h)) + dis(L_p, L_m(i))]/S(h),\ h \in H_i \} \qquad (4.3)$$

where, $t_{res}(i)$: reserved operation time of resource *i*; L_p ,$L_{res}(h)$, $Lm(i)$ are the location of Part, last reservation location of Material handling equipment *h,* and location of machine *i*. $dis(L_1, L_2)$ is the distance between location L_1 and L_2. $S(h)$ is the speed of material handling equipment *h*.

The control logic of PA and MA is shown in Figure 4.9 and 4.10 respectively,

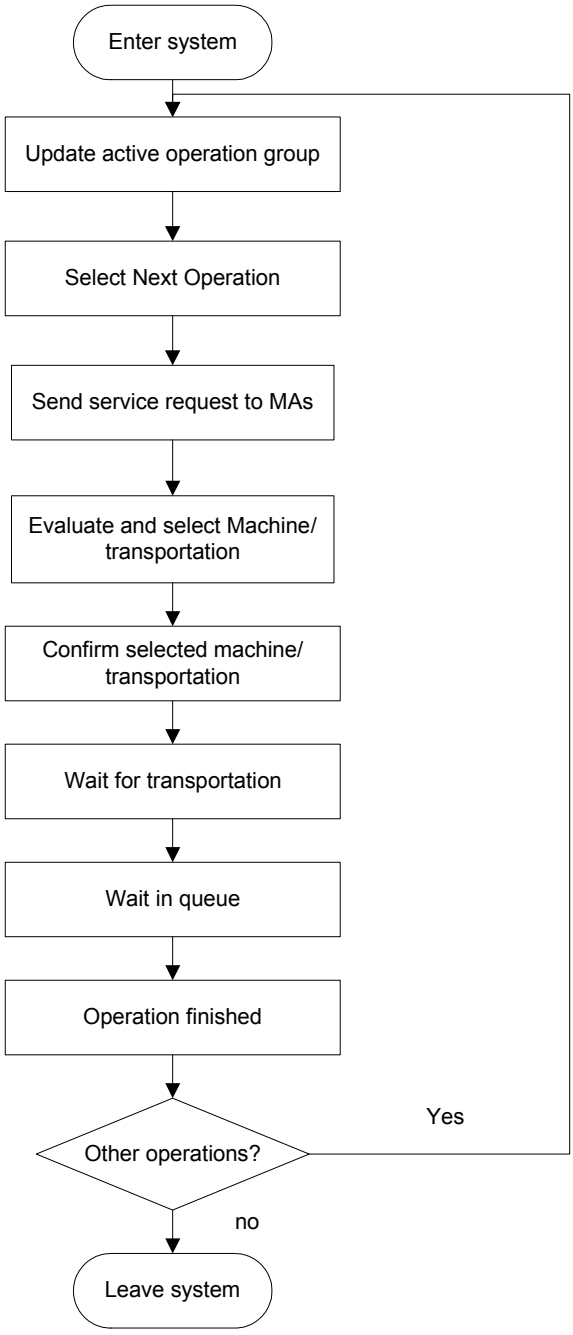


Figure 4.9. PA decision flow

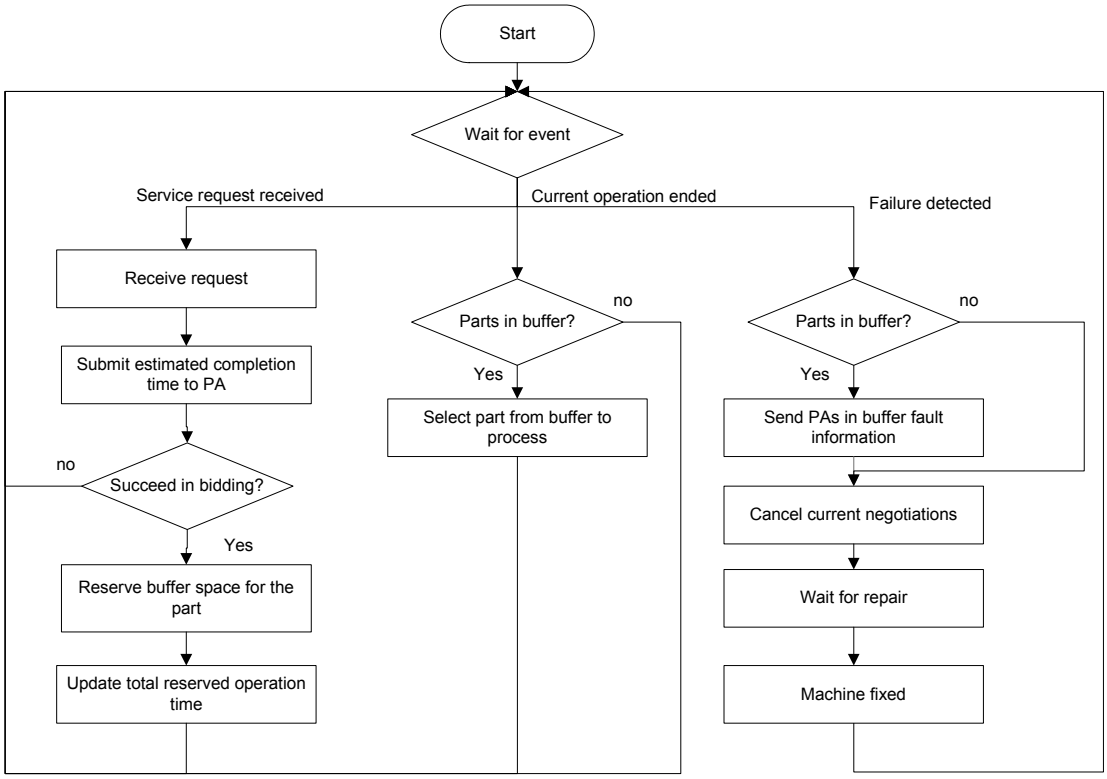


Figure 4.10. MA decision flow

To reduce the communication burden and resolve conflicting interests among agents, an extended contract net protocol is presented in the next section.

Step 3: Machine agent select parts in buffer based on predefined dispatching rules

As mentioned before, agent systems suffer from myopia problem because agents only use local information available at decision time to make decisions. After PA assigns its operation to MA/MHA and gets confirmed, the corresponding part is then transported to the selected machine and being processed immediately if the machine is idle. Otherwise, the part will be waiting in the machine's buffer space. During the waiting time, the actual events in the shop can be considerably different. The decision made by negotiation between PA and MA/MHA may not be valid any more. Corrective actions need to be done to enhance the degraded performance of the original decision. Each time a machine completes an operation, the corresponding MA selects parts in buffer based on specific dispatching rules according to new information at the time. Global scheduler can

also be designed to intervene the local decisions by changing the order in which these parts in buffer are processed.

4.3 Summary

In this chapter, the agent-based exception handling process in CICA architecture is presented. The agent-based exception handling is implemented as a functional module in the CICA architecture. In such a design, unless an exception is detected, the manufacturing system with CICA architecture is controlled by centralized decision-making module due to the fact that centralized methodology is more efficient in a deterministic environment. The agent-based system take over control of system once an exception is detected. The main agent types are summarized and the negotiation process among the agents is presented. A new stepwise decision process with consideration of part process plan is used in the PA-MA negotiation. The detail design of different agents and their decision logic are also defined in this chapter.

In the next chapter, some neglected problems related with agent-based decision process is analyzed and resolved. A comprehensive experimental analysis is also provided in the next chapter.

Chapter 5. Agent-based Scheduling with local conflict resolution

Although there are many desirable features in agent-based systems, there are some well-known weaknesses of agent systems, such as communication load and local interest conflicts. Despite the numbers of literatures on agent-based scheduling, these problems are not or less exploited. In this chapter, these problems are studied based on classification of basic agent negotiation scenarios.

Generally speaking, negotiation processes among autonomous agents are costly in terms of the communication and computation overhead. When the number of agents becomes large, the amount of message grows dramatically. Agents may spend more time on processing messages than actually doing the work. In the worst case, the system may stop being flooded by the messages. If there is not an efficient way to manage negotiation, the negotiation overhead will become a bottleneck, which could considerably deteriorate the system performances. Nonetheless, the effects of large number of agents and excessive communications are rarely studied. It is worthwhile to investigate the communication overhead of agent negotiations.

Another problem of a Contract Net-based system is that agents only have local views of the overall problem, therefore conflicts may exist among agents' decisions. Autonomous agents have their own goals and preferences and are likely to follow their own self-interest in decision-making process. However, when they compete for a set of shared resources, it is not sufficient for an agent to try to optimize its own local preferences. Instead, agents need to consider the preferences of other agents when they make decisions on their activities. This current negotiation protocol by itself does not avoid conflicts when multiple tasks negotiate with the same resources at the same time. The majority of the contracting algorithms in literature do not consider the concurrent negotiation between multiple PAs and MAs.

In a negotiation situation, the number of participants on each side is an important characteristic of the negotiation. Different combinations, ranging from one-on-one bargaining to multiple participant negotiations, could affect the negotiation process significantly.

In this chapter, negotiation scenarios are classified based on the numbers of agents involved in a negotiation. Potential conflicts can happen in one of the basic negotiation scenarios. A "look ahead" technique is used to capture such potential conflicts. The behavior of agents in conflicts is then studied by game theory. A mediator agent is employed to resolve conflicts by utilizing a metaheuristic named Tabu Search. In the end of this chapter, simulation results are used to illustrate the methodology.

5.1 Basic scenarios of agent interactions

In this research, the basic Contract Net Protocol is extended to solve the aforementioned problems. PA-MA interaction scenarios are sorted into three basic categories according to participant agent numbers (Figure 5.1). By treating these basic interaction scenarios differently, potential conflicts can be detected and resolved. At the same time, communication load can be considerably reduced.

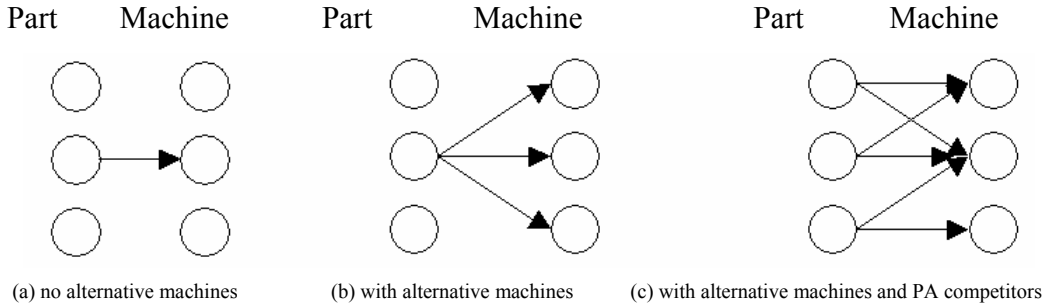


Figure 5.1. Three basic PA-MA negotiation scenarios

The three basic interaction scenarios are categorized in PA-MA negotiations: in the first type, an operation of a PA can only be processed by one MA. In this case, no negotiation is needed. The MA will be directly contracted. The other two types handle the situation when the operation of a PA can be processed by multiple MAs.

Scenario 1*: one-to-one PA-MA interaction scenario*

If an operation of a PA can only be processed by one particular MA, negotiations are unnecessary for task contraction between the PA and MA. Instead, without announcement, the PA awards a direct contract with the MA, which is appropriate for the

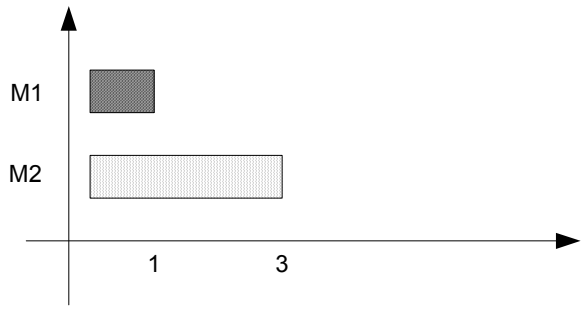
task. Among all the feasible MHAs that can help in finishing the operation, the MHA with the least reserved operation time is selected for transportation. Therefore, then the selected MA and MHA update their reserved operation time.

Scenario 2*: An operation of a PA can be processed by more than one MA, and in a predefined time period, these MAs won't be requested by other PAs.*

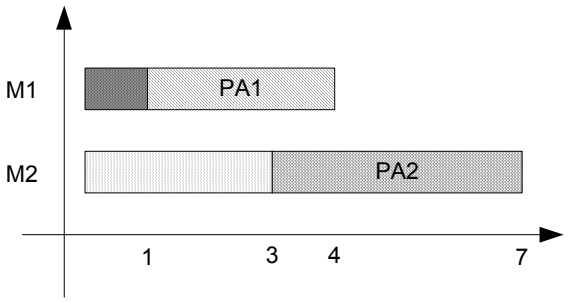
When there are no PAs competing for the same MAs, the general Contract Net Protocol are used. By referring to the yellow page service, if the next operation of a PA can be processed by more than one machine, the event calendar will be checked to see if any of the machines will be requested by other PAs in a predefined time period. If none of these machines will be requested, which implies that the PA's decision on this operation won't affect others', the PA sends requests to all the MAs corresponding to the feasible machines. The MA and MHA will be selected based on the shortest estimated finish time as described in the Step 2 of stepwise decision-making process in the last chapter.

Scenario 3*: An operation of a PA can be processed by more than one MA, and in a predefined time window, these MAs will be requested by other PAs.*

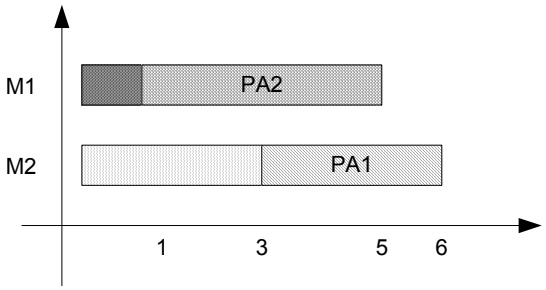
In multi-agent systems, these agents are self-motivated and try to maximize their own benefits. Each agent is self-interest centered, meaning that the final solution may be the best for each agent involved, but may not necessarily be the best for the group as a whole. The problem can be illustrated by an example shown in Figure 5.2.



(a)



(b)



(c)

Figure 5.2. A simple example to illustrate agents' conflicting interests

Figure 5.2 (a) illustrates before part agent PA1 and PA2 arrive, the reserved operation time of machine agent M1, M2 are 1 and 3 time units, respectively. Machines of M1 and M2 are identical machines and can perform both PA1 and PA2's next operation. The processing time of PA1's next operation is 3 time units. The processing time of PA2's next operation is 4 time units. PA1 sends out requests for its operation at time 0.5 and PA2 sends out requests at time 1. For this simple example, the negotiation time and communication time are neglected.

As PA1 sends its request earlier, so it has the privilege to select the most desirable machine to fulfill its interest and M1 will then be selected. Though compared with the later coming PA2, PA1's job can be executed in a shorter time and may even have lower priority, M1 is occupied by the part corresponding to PA1 before PA2 sends its request. When the batch consists of only two parts (PA1 and PA2), apparently the scheduling result shown in figure 5.2 (c) has shorter batch completion time (locally) than the one shown in figure 5.2 (b) with better machine load balance.

As illustrated in the simple example, the actions of one agent often affect other agents. Self-interest centered agents, acting without concerning other agents, may cause a negative effect on the overall efficiency of the outcome. The contract net protocol by itself does not avoid interest conflicts when multiple tasks wanting the same resources at the same time. When several PAs try to access simultaneously to several machines offering the same service, it may also cause deadlock (Ramos 1996).

5.2 "Look ahead" technique

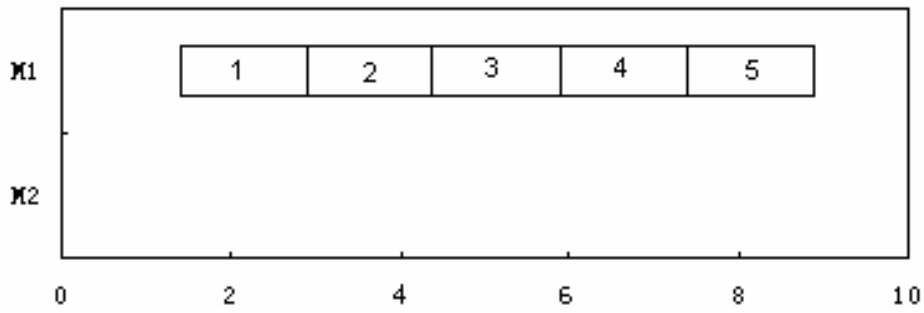
As mentioned in the last section, potential interest conflicts occur when a PA' next operation can be processed by more than one MA and there are other PAs competing for same resources in near future. A "look ahead" technique is used to determine if such potential conflicts exist by examining contents of the event calendar in system in a "look ahead" time window. All jobs that are ready to be executed plus all "soon to be ready" jobs are considered in a conflict resolution situation. The event calendar in system records a list of scheduled events. The event calendar is automatically updated with following events: ready time of incoming PAs, finishing time of the PA currently under processing, i.e. we assume that the system has knowledge of jobs arriving within the look-ahead window.

If conflicts among agents can be successfully resolved in "look ahead" time windows, the local scheduling results inside time windows will be improved. The improvements on local scheduling results can eventually lead to a better global performance overall. To achieve the objective, the size of "look ahead" time window needs to be properly defined. A larger time window is able to consider more potential conflicts, which usually implies that getting a result with better global performance, however, with more computation burden (as will be mentioned later in this chapter, the complexity of scheduling problem in a time window grows exponentially). The extreme case of a large time window is the whole production time period. When the extreme case occurs, the system is then no longer an agent-based system but the central one that schedules all jobs ahead of production time. Scheduling problems in large time windows are complex and generate inflexible results. On the contrary, a small time window may miss potential conflicts. A possible length of a time window can be defined as the

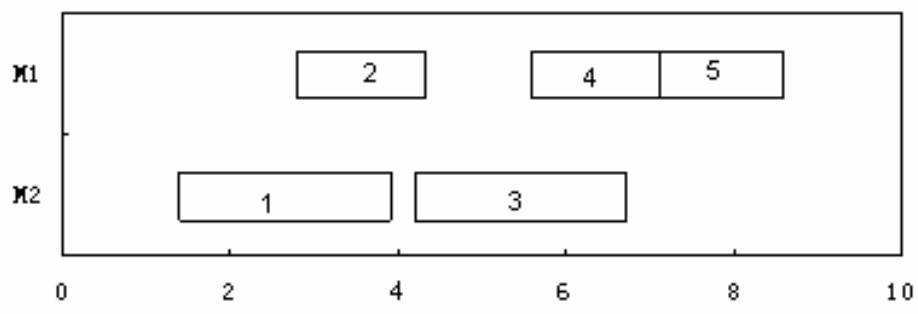
minimum of all processing times. The definition of such a time window length guarantees an active schedule (Sridharan and Zhou, 1996). As we are using single-step production reservation method, the look-ahead time window also reduces the potential chances of deadlock.

In such a case, the length of a time window is defined as, $t_w = \min \{t_i \mid i \in I\}$, t_i is processing time of job i and I is the set of all jobs in system.

The following example is used to illustrate how the "look ahead" technique is applied. There are $i = 5$ identical jobs, $j = 2$ machines in a manufacturing system. Ready time of the jobs $r_i = 1.4 * i$. Processing time of the jobs on each machine are $p_1 = 1.5$, $p_2 = 2.5$. Then the definition of "time window", $t_w = min\ (p1, p2) = min\ (1.5, 2.5) = 1.5$. Both machines (M1, M2) are idle at time zero. Figure 5.3 (a) is the scheduling result when PAs make decisions by themselves without intervention. Figure 5.3 (b) illustrates the scheduling result that time windows are used in the decision process. Notice that the time interval between conjunct jobs is 1.4 < 1.5, there are two scheduling process happened in "look ahead" time windows with job 1, 2 and job 3, 4 respectively.



(a)



(b)

Figure 5.3 Improvement of scheduling result by using "look ahead" time windows

From the results shown in Figure 5.3, the scheduling results in both time windows are better off after “look ahead” technique is used. The improvements in local schedules of each time windows eventually lead to improvements on the overall scheduling result. The local schedule arrangements can lead to better global performance, however, cannot guarantee an optimal result. The optimal schedule in this example has a makespan of 8.5 (Figure 5.4) instead of 8.6 as shown in the Figure 5.4(b). The optimal solution is able to be reached if a larger time window is used.

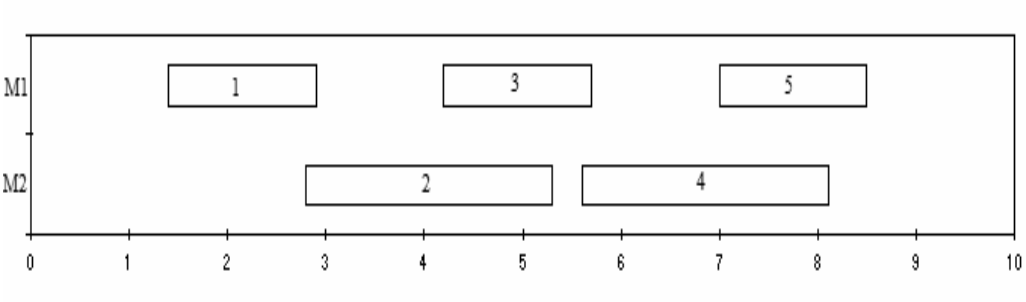


Figure 5.4 The optimal schedule with shortest makespan

When negotiation processes in PA-MA interactions are classified into three basic scenarios, the flowchart of PA decision process is shown in Figure 5.5.

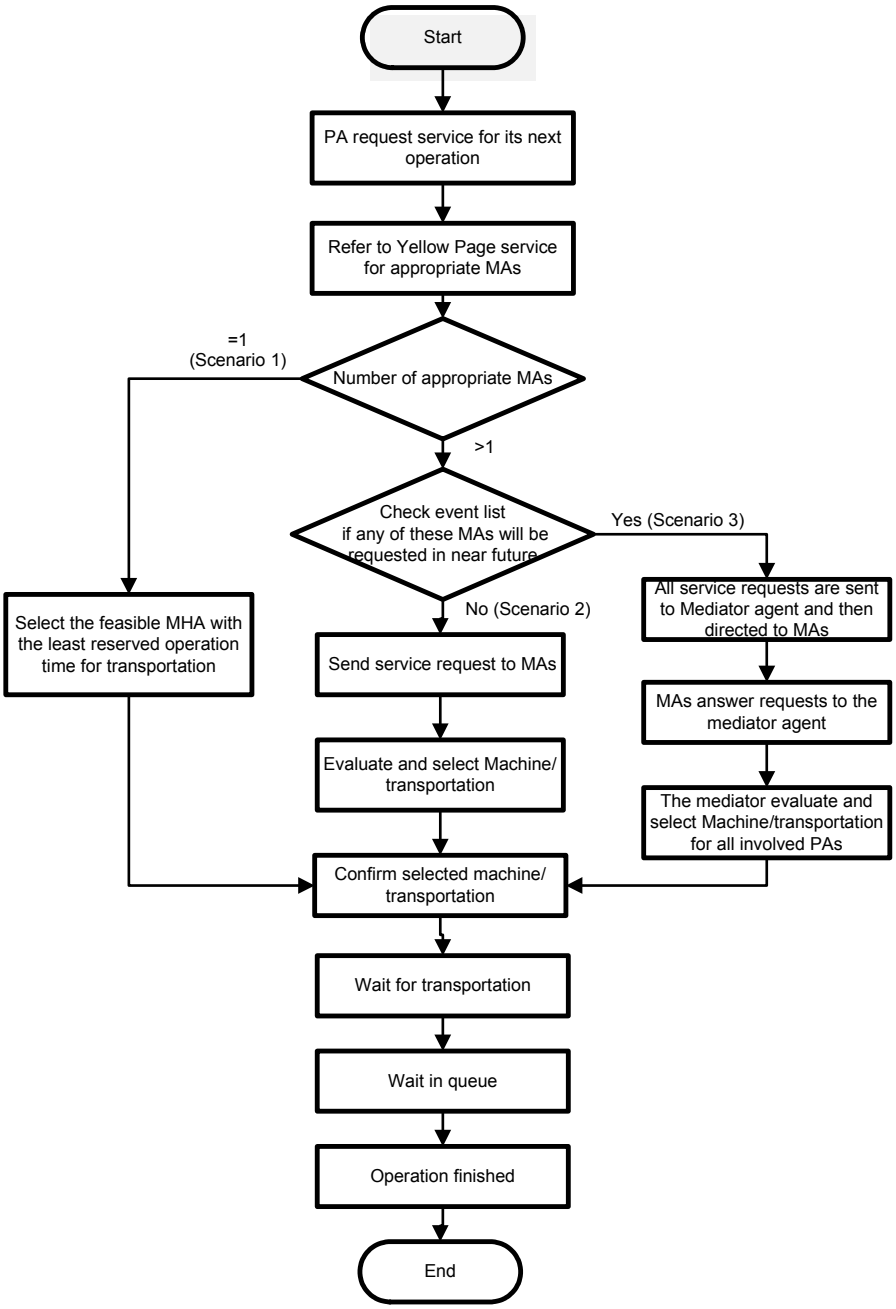


Figure 5.5. PA decision process in the three basic negotiation scenarios

5.3 Game theoretical analysis of local interest conflicts

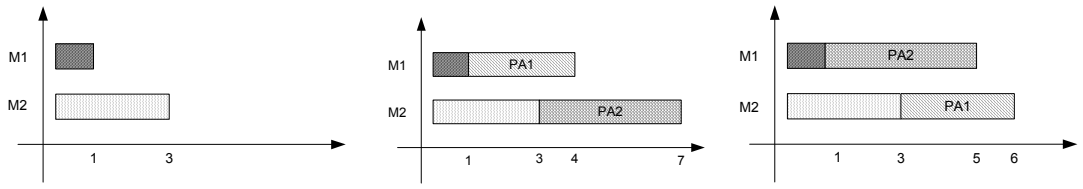
While the "look ahead" technique is used to detect potential conflicts, the analysis and resolution of conflicts among agents pose another challenge. A potential way that can be used to determine the behavior of all the agents involved in a conflict is the application

of game-theoretic techniques. Game theory has been considered as an important formalism for studying strategic and cooperative interaction in multi-agent systems in recent years.

Game theory can be used to study mathematical models of conflict and cooperation between multiple interacting agents (whether artificial or human). In a game-theoretic analysis, an interactive situation is described as a game: an abstract description of the players (agents), the courses of actions available to them, and their preferences over the possible outcomes. Each player has its own interests and wants to maximize its own utility, and thus the players may have local conflicting interests.

Game theory can be used to formally model multi-player game and prescribe rational strategies to the different players. Studies have shown the behavior of the interacting agents can be explained in terms of Nash Equilibrium (NE) and that global system performance correlates in some way to the payoffs of these equilibriums. The abstract models of game theory have been used as a basis for the agents' interaction protocols by some researcher.

Competition among agents for shared resources in game theoretic terms can be modeled as normal form games. Agents in such models are considered as players working toward their own self-centered goals and taking part in an *N*-player game. The objective of each player is to maximize its payoff. In scheduling problems, the payoffs can be set to the estimated completion time for each player if all players play the specified strategies. Game theoretic models are divided into two main types: "non-cooperative" and "cooperative" models based on whether the agents work towards a common goal. In a non-cooperative game, the behavior of the interacting agents can be explained in terms of Nash Equilibrium (NE). However, the outcome of a non-cooperative game dictated by individual "rationality" may not satisfy a criterion of collective rationality.



(a) (b) (c)

Figure 5.6. A simple example to illustrate agents' conflicting interests

Table 5.1 depicts the normal form game corresponding to the instance of Figure 5.2 (the group of figures is redrawn in Figure 5.6 for easy reference). The row headings are the strategies available for player P1. That is, "M1", "M2" at the beginning of a row indicator that P1 chooses to enter the queue of M1 and M2 respectively. The column headings similarly describe the strategies available for player P2. Each entry in the matrix lists the players P1, P2 in that order. The payoffs correspond to the estimated completion time if the players play the specified strategies. The solution (M1, M2) is the NE of the game. This is the result when agents try to maximize their own interests, and as we already know, it is not optimal according to the global performance.

Table 5.1. Formal game model of two PAs

	M1	M2
M1	(-4, -8)	(-4, -7)
M2	(-6, -5)	(-6, -10)

Although the agents can act and reach goals by themselves, it may be advantageous for them to work jointly as in a "cooperative game". In cooperative games, players can make binding agreements and choose their strategies jointly. A set of players that coordinate their strategies in order to promote their joint interest is called a coalition. When makespan is the only system objective in scheduling, all the PAs competing for resources in a time window can easily form a coalition ("grand coalition" as defined in game theory). Players act in order to get the largest joint payoff guided by their collective interest. It is possible that an agent can agree to a contract even if that makes the agent worse off (have a later completion time) as long as the social welfare in the system increases (shorter makespan). The utility function for the grand coalition can be defined

as a vector of expected completion time of all machines involved. The completion times are arranged in a monotonic decreasing order. The vectors corresponding to different strategies are compared in such a way: the values of the first member in the vectors are compared and the one with the smallest first member is chosen. If there is a draw, then the second one is compared, and so on. All possible strategies are compared and the best strategy to the group of all agents is then selected. According to Table 5.2, the player with the strategy (M2, M1) has the highest payoff (the shortest completion time of system). This is what we know the optimum solution from Figure 5.6.

Table 5.2. Payoffs to different strategies of grand coalition

Strategy	Payoff
(M1, M1)	(-8, -4)
(M1, M2)	(-7, -4)
(M2, M1)	(-6, -5)
(M2, M2)	(-10, -6)

*Notice the members in each vector are no longer the completion time of jobs, but the expected completion time of machines. They are same in this simple example.

Although cooperative game decisions made by agents are able to resolve the conflicts and achieve good global performance, the cooperation among agents requires substantial computation and additional communication. To apply game-theoretical model, besides matching agent systems with definitions of game-theoretical model, and defining utility functions, how to identify equilibrium strategies and developing low complexity computational techniques for searching for appropriate strategies are also needed. Due to the complexity of finding strategies, the number of agents in an interaction is usually small (less than a dozen agents).

A central controller can be employed to collect information from agents and resolve conflicts, instead of letting agents cooperate by themselves. The scheduling problem in a time window is of small scale and can be effectively solved by a centralized methodology without affecting the flexibility of whole system. The mediator agent is designed to fulfill the central controller's role.

Whenever a PA sends a request for service, the request is sent to the mediator agent. If the service can be performed by multiple MAs, the mediator looks ahead in the event

calendar. When potential conflicts are detected, the mediator agent acts as a supervisor to collect information from all agents involved and decides the best strategy for all the part agents in the group. When modeling the competition among agents as normal form game , the group of PAs in a time window that compete for resource can be considered to work cooperatively as in a grand coalition. The mediator agent, acting as the "social planner" in system, evaluates and determines the best strategies for the agents in the coalition (Figure 5.7).

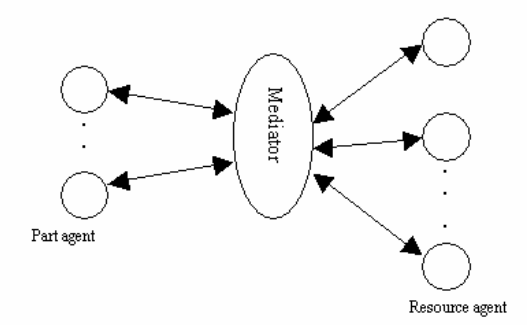


Figure 5.7. Mediator agent's role in PA's resolution of conflicting interests

The mediator can use the definition of the utility function for the grand coalition to make decisions for all the agents involved in a "time window". The mediator agent chooses the schedule with the best value of the utility function for the grand coalition. When resolving conflicts, the mediator agent records all the involved PAs and MAs: PA_i, $i=1,...n$; MA_j , $j=1,..., m$ and request for all necessary information. All PAs can be considered as a grand coalition, and the strategies adapted by each PA together form the strategy for the grand coalition. The mediator agent can use the definition of the utility function for a grand coalition to determine best decisions for all PAs.

The utility function of each strategy is defined as a vector $v = (t^{(1)}, t^{(2)},..., t^{(m)})$, where $t^{(i)}$, $i=1, ..., m$ is the ith largest completion time of m machines when a specific strategy is employed. The length of the vector equals the number of machines involved.

The expected completion time of each machine is calculated by

PA_i, $i=1,...n$; : ith Part Agent (n Part Agents having conflicting interests)

MA_j , $j =1,..., m$: jth Machine Agent (m Machine Agents involved)

Mh_{ij} : set of material handling equipments which can travel between location of PA_i and location of MA_j

r_j : ready time of PA_i before assigning to any machine

r_{ij} : ready time of PA_i on machine j (material handling time included)

t_{ij} : processing time of PA_i on machine j

$t_{res}(j)$: reserved operation time of machine j

$t_t(i, j)$: transportation time from PA_i's current position to machine j

Out of all the PAs assigned to machine j in a strategy, σ is defined as the set of all PAs which have $r_{ij} < t_{res}(j)$. The reserved operation time of machine j is updated to be

$$t_{res}(j) = t_{res}(j) + \sum_{i \in \sigma} t_{ij}$$

The rest of jobs are compared with the updated reserved operation time. A new σ is generated and reserved operation time is updated again. The process continues until the PA with earliest r_{ij} has $r_{ij} > t_{res}(j)$ or all jobs are considered in $t_{res}(j)$. In the former case, the reserved operation time is updated to be $t_{res}(j) = r_{ij} + t_{ij}$. And the process continues with new generated σ if there are any unconsidered jobs. In the end, the expected completion time of machine $j = t_{res}(j)$

To determine the ready time of each job on machines, start from the job with earliest r_j, the ready time of PA_i on machine j, r_{ij} is calculated as

$$r_{ij} = Max\{\min_{h \in MHij} [t_{res}(h)], r_i\} + t_t(i, j) \quad (5.1)$$

And the reserved operation time of material handling equipment h is updated to be

$$t_{res}(h) = r_{ij} \quad (5.2)$$

The process that a mediator agent utilizes to solve scheduling problems within time windows based on utility functions can be summarized as following:

Step 1, When a part agent's next operation can be executed by more than one machine, the mediator agent checks the event list to find out if any of these machines will be requested by other part agents in the near future. If it is the case, the part agent contact the mediator agent with a list of all part agents, resource agents involved.

Step 2, The mediator agent then contacts all part agents and collects information such as locations of current parts, etc.

Step 3, The mediator agent calculates the utility function for all strategies the agents can have. The strategy with highest payoff is selected.

Step 4, the mediator announces the decision to all part agents, and the part agents contract with resource agents. All the agents update their records.

Agent-based scheduling with local decision arrangement in time windows intends to have scheduling results that have advantages of central and distributed schedulers. Agents are used to provide scheduling decisions for complex and dynamic production environment. Whenever there are detectable potential conflicts that agents are not capable to handle themselves, a central scheduler (mediator) takes over control.

Another advantage of using a mediator agent is that communication messages can considerably be reduced. The mediator agent collects information from involved agents and makes decisions for them, hence saves the cost of negotiation messages among agents. During a general Contract Net Protocol negotiation process, when there are one initiator and *m* contractors, the total communication messages include at least *m* call for proposal messages, *m* bids from contracts and *m* decisions sent from the initiator to the *m* contractors. There are at least 3*m communication messages in each general CNP negotiation process. To compare the communication messages of a general CNP with the extended CNP that treats three basic negotiation scenarios accordingly, assume in a multi-agent scheduling system, there are N1 scenario one negotiations, N2 scenario two negotiations, and N3 scenario three negotiations respectively. If a general CNP is used, the total communication message will be $3*\sum_{i=1}^{(N1+N2+N3)} m_i$, where m_i is the number of resource agents involved in *i*th negotiation. With the classification of the basic negotiation scenarios, the total communication messages are reduced to be $2*N1 + 3*\sum_{i=1}^{N2} m_i + 2*\sum_{i=1}^{N3} m_i$.

As described in this section, the mediator agent can use the definition of utility function for a grand coalition to resolve conflicts among agents. However, the enumerative methods become complex as the number of agents grows. The numbers of strategies in comparison grow exponentially and easily become intractable. An efficient heuristic is needed to solve the scheduling problem for the involved agents within a “time window”.

5.4 Tabu search-based local scheduling decision arrangement

A simplified situation of the scheduling problem inside a "look ahead" time window is that if when PAs compete for machines, they compete for the same group of machines, the scheduling problem within a time window can be modeled as minimization of makespan on unrelated parallel machine scheduling with ready time (R|ri|Cmax). Scheduling problems related to R|ri|Cmax have been extensively studied in scheduling literature. When the assumption is relaxed, the scheduling problem in a time window can be more complicated. However, the study results from these researches can be used for the resolution of conflicts among agents in a time window.

The problem of unrelated machines, where the processing time of each job differ on every machine, is well known to be a NP-hard problem, even in the simplest case of two identical machines. Heuristic algorithms have been extensively developed to solve real world problems with very good results. Researchers proved that exact algorithms and approximation methods with theoretical guarantees are outperformed by simple iterative local improvement algorithms (Survey: Mokotoff, 2001).

Improvement algorithms are based upon local search in neighborhood. They initiate with a feasible solution as starting-point and try to improve it by small iterative changes. The iterative improvement can be achieved by means of many different processes. Three popular algorithms are: Simulated Annealing (SA), Tabu Search (TS), and Genetic Algorithms (GA).

Tabu search has probably been the most tested local search concerning general job shop problems. Research work shows among the popular iterative improvement algorithms (genetic algorithm, simulated annealing, Tabu search), Tabu search is more preferable (Glass *et al*., 1994; Jozefowska *et al.* 1998,). TS is not only able to generate better solutions in short time period, TS also performed best in finding optimal solutions more often and having smallest deviation from optimal for all problem size. The performance of TS is also not significantly affected by numbers of jobs.

Tabu search is designed as a very general and adaptive approach for solving hard combinatorial optimization problems, and can be applied on any local search procedure (Glove, 1986). It is an iterative neighborhood search technique and the algorithm

explores through the solution space by moving to the best neighboring solution that is not visited before. A move on the tabu list is called a tabu move and its duration is governed by the size of the tabu list.

Occasionally, a good solution may be lost if the movement to reach it is contained in the current tabu list. Such a loss can be avoided by allowing the tabu list to be overridden according to an aspiration function. If the objective function value of a neighbor is no worse than the value of the aspiration function for the current solution, then the move is eligible for acceptance, irrespective of whether the neighbor is tabu. Therefore, at each stage of the algorithm, an admissive move is either a non-tabu move, or a tabu move satisfying the aspiration criterion.

The structure of a general Tabu Search algorithm can be outlined as following,

Starting point: Generate an initial feasible schedule and compute its objective function.

Neighbor Search: Select a feasible neighbor of current schedule and compute its objective function.

Acceptance test: Test whether or not to move from current schedule to its neighbor.

Termination test: Decide whether to stop the algorithm. If the algorithm is terminated, then output an appropriately chosen schedule; otherwise, return to the neighbor search step.

In order to apply Tabu search to solve the scheduling problem inside "time window", fundamental elements of Tabu search such as starting point and neighbor search methods need to be defined.

The starting schedule is obtained by using a greedy approach (Piersma and Dijk, 1996). The greedy approach assigns each job to the machine on which it has minimal processing time. When there is more than one machine with the smallest processing time, choose the machine with smallest makespan in the partial schedule.

Two types of neighbor are defined for every schedule generated in search process:

1). **Reassignment Neighbor**: Assign one job from a machine with maximum makespan to another machine and

2). **Interchange Neighbor**: Interchange the machine assignment of a pair of jobs.

The search for a neighbor schedule is performed first in reassignment neighbors and then in interchange neighbors.

The Aspiration Criteria is defined as when Objective function value of a neighbor is no worse than the current best. Researchers also suggest a dynamic tabu list size to be more robust than tabu list with fixed size in recent applications. As the numbers of jobs and involved machines in the scheduling problem within a time window are not static, both tabu list size and tabu search cycles are defined to be dynamic. In this research, the Tabu list is defined to be (n - 1) for both types of neighborhoods, and the Tabu search cycles are defined to be (n*10), in which n is the number of jobs involved in Tabu search.

The following example illustrates how TS is used to improve local scheduling results and eventually improve the overall schedule. In this example, there are two machines and ten randomly generated jobs belonging to three job types in the system. The processing time of the three job types on the two machines are listed in Table 5.3. Information of the ten jobs such as job types and ready times are listed in table 5.4. When applying the "look ahead" technique, it is not difficult to find out there are two time windows in this scheduling problem, which include job 2, 3 and job 7, 8, 9, 10 respectively (as shown in two groups of grey columns of Table 5.4).

Table 5.3. Processing time of different job types on machines

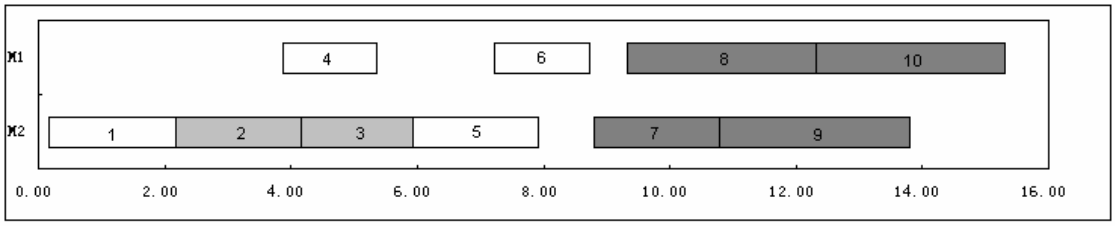
	M1	M2
Type 1	1.5	3
Type 2	3	2
Type 3	2.5	1.75

Table 5.4. Job information

Job ID	1	2	3	4	5	6	7	8	9	10
Job Type	2	2	3	1	2	1	2	2	1	2
Ready Time	0.17	1.71	2.74	3.86	5.66	7.22	8.79	9.32	9.49	9.94

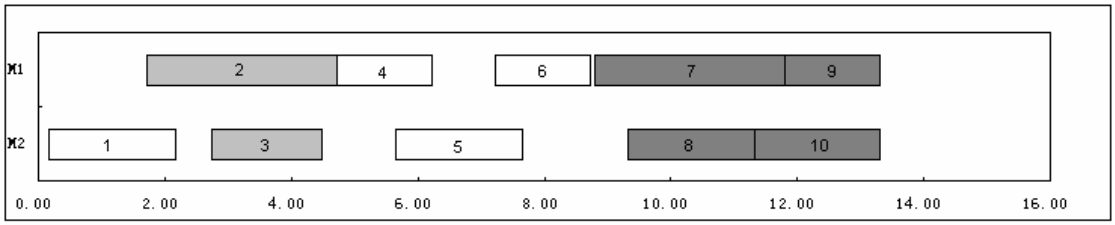
The scheduling results for the problem are compared for agent-based decision process with/without using "look ahead" technique. Figure 5.8 (a) shows the scheduling result of a multi-agent system without using "look ahead" technique. Figure 5.8 (b) shows the scheduling result of the multi-agent system that uses Tabu Search technique in

"look ahead" time windows. Cmax is the makespan value. U1/U2 are machine utilizations of machine 1 and 2 respectively.



Cmax = 15.32; U1 = 58.7%; U2 = 83.2%

(a)



Cmax = 13.32; U1 = 78.8%; U2 = 73.2%

(b)

Figure 5.8 (a) Schedule made by agents without using "look ahead" technique

(b) Schedule made by agents with using Tabu search inside "time windows"

By using TS in the "look ahead" time windows, the scheduling result shown in Figure 5.8 (b) has shorter completion time (13.05% reduced) and more balanced machine utilization. In fact, Tabu search can be very efficient in solving small scale scheduling problems. In a study of the single machine sequencing problem (jobs with ready time, NP-hard), the percentage of TS that can reach optimum value reported in a study is more than 90% within a limited computation time. And when the optimum value is not reached the error is on the average, of 2.3% only.

By using TS in the "look ahead" time windows, a PA's decision process will be updated as following,

Step 1. Enter system

Step 2. Update active operation group

Step 3. Select next operation

Step 4. Check if any other agents compete for resources in the "look ahead" time window

If yes, wait for the mediator agent to use Tabu search to make decision and then go to step 7

If no, go to step 5

Step 5. Request service from MA

Step 6. Evaluate and select machine/transportation

Step 7. confirm selected machine/transportation

Step 8. wait for transportation

Step 9. Wait in queue

Step 10. Operation finished

Step 11. Check if any operation left

If yes, go to step 2

If no, exit system

In order to fully test the effectiveness of using TS in "look ahead" time windows to improve scheduling results in flexible/reconfigurable manufacturing environments, more sophisticated experiments are tested in the next section with consideration of computation time, significance of improvements, etc.

5.5 Experiment design and results

5.5.1 Simulation model

In this section, a small hypothetical manufacturing system is modeled to evaluate the performance of the proposed agent-based scheduling methodology. The system used as a test-bed includes two types of machines (Lathes and Milling machines are taken as examples). Each of the machines in the system is capable of performing various operations. The material handling inside the system is carried out by two robots. This simulated manufacturing system is similar in structure to the manufacturing cell in

flexible manufacturing lab at Virginia Tech University. The details of the manufacturing system model are shown in Figure 5.9, 5.10, Table 5.5, 5.6.

The hypothetical manufacturing system used for this research is developed using the Arena discrete-event simulation package, while the control logic and interaction protocols of agents are implemented using VBA incorporated in Arena. The simulated manufacturing system and algorithms are implemented (the pseudo codes of the algorithms can be found in appendix) and run on a Pentium 3 PC having a 600 MHz CPU and 1GB memories.

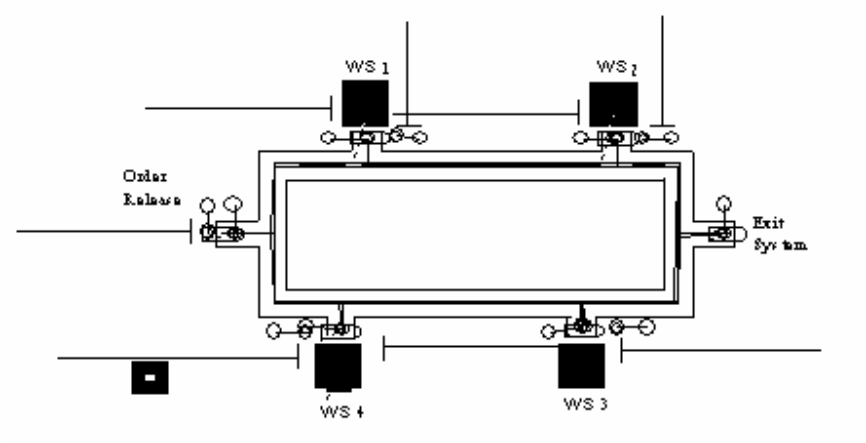
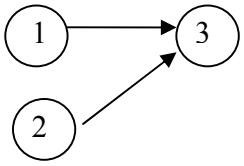


Figure 5.9. Simulation model of the hypothetical FMS in Arena

Type 1:



Type 2:

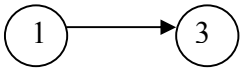


Figure 5.10. Part processing sequences

Table 5.5. Distances between stations (in inches)

		To					
		Order release	WS1	WS2	WS3	WS4	Exit System
From	Order release		37	74			
	Cell 1	155		45	92	129	
	Cell 2	118	139		55	147	
	Cell 3	71	92	129		45	155
	Cell 4	34	55	92	139		118
	Exit System	100	121	158	37	74	

*Average speed, 100/min; loading/unloading time, 0.25min

Table 5.6. Operation processing times (in minutes)

	WS1	WS2	WS3	WS4
Part Type 1	6	4	8	5
Part Type 2	9	3	18	8

The experiments are designed to test the effectiveness of proposed multi-agent decision system compared with traditional dispatching rules. The performance metrics tested for the multi-agent system is real-time response to disturbances (machine breakdowns), effectiveness of the algorithms (CPU time consumed for algorithms), and system burden (number of messages circulated in decision process).

5.5.2 Experiment design

To evaluate the effectiveness of the proposed agent-based scheduling process in a FMS/RMS environment, a group of sophisticated experiment has been designed based on the simulated FMS model. Two most popular used dispatching rules (FIFO, SPT) in today's FMS systems and the proposed stepwise agent-based decision process both with and without "look-ahead" capability are compared. MAS and MASTW are used to represent the multi-agent system without using "look ahead" technique and the multi-agent system with using the technique respectively. The conditions, simulated scenarios, and characteristics tested are similar for all the approaches such that a reliable comparison can be obtained.

The main performance metric that is used to evaluate the performance of the different approaches is the total completion time of all the jobs (200 jobs are used in this simulation) in the system (or makespan of all the jobs). Parameters of a job, such as, part ready times, and part types are all randomly generated. A 40% job type 1 and 60% job type 2 ratio is kept across all the simulation replications. The average of makespan from 10 simulation replications is used for comparison. When comparing the two agent-based scheduling approaches (MASTW and MAS), the CPU time consumed is monitored for comparison of computational effectiveness. Also the message numbers used by agent to generate scheduling decision are also recorded for evaluation purpose.

Different simulation scenarios have been design to evaluate the performance of all the approaches. One group of experiments is performed in conditions that all machines and robots running at full capacity without considering potential breakdowns. Another group of experiments is performed to analyze the response of different methodologies to perturbations – the failure of one machine is considered in system. In each of these two scenarios, three different introduce rate (slow, medium, fast) of parts are tested for all approaches, which is used to analyze the relation of performance indicator with the manufacturing load. Also the influence of the time window length is examined in simulations. Three different lengths of time windows are used: *p, 2*p, 5*p*, Where $p = \min \{p_i \mid i \in I\}$, p_i is processing time of job i and I is the set of all jobs in system.

Assume parts that enter the system follow Poisson arrival process. The time between job arrivals into the system follows an exponential distribution having a mean of λ. Three different λ values (4, 6, 10) minutes are used to represent different manufacturing load scenarios.

In the experimental test that considers the occurrence of unexpected disturbances, the machine breakdowns of machines are simulated using time periods from exponential distributions. Mean Time between Failures (MTBF) and Mean Time to Repair (MTTR) are all from exponential distribution with means of 120 and 20 respectively.

CPU time is monitored for the computation time consumed by CPU for TS when MASTW approaches are used. QueryPerformanceCounter() is a Windows API timer function that can be used to record elapsed time by comparing the returned values at two different points in time.

A statistical multiple comparison procedure is used to test the significance of difference among the outputs from different approaches. Two popular methods: Bonferroni's method and Tukey's method can be easily implemented in SAS software. All the tests of significance are carried out at $\alpha = 0.05$ level. This level of significance is a value that separates results typically seen when a null hypothesis is true from those that are rare when the null hypothesis is true. The level of significance is the probability of those rare events that permit investigators to claim an effect. When we test at the 0.05 level of significance, the probability of observing one of these rare results when there is no effect is 5%. For each replication of experiments, a group of outputs is generated for each approach used in comparison.

5.5.3 Simulation results

The results obtained from the experimental tests, using the simulation scenarios described previously, are presented and discussed in this section. Table 5.7 summaries the average performance metrics from the ten replications from each different simulated scenario.

Table 5.7. Performance of evaluated approaches for different scenarios

(a). $\lambda = 4$, no machine breakdowns

	SPT	FIFO	MAS	MASTW		
				tw = p	tw =2 * p	tw =5 * p
Cmax	925	911	891	851	823	788
CPU	-	-	-	7.25	11.5	31.2
Msg. #	-	-	1451	1076	954	813

(b). $\lambda = 6$, no machine breakdowns

	SPT	FIFO	MAS	MASTW		
				tw = p	tw =2 * p	tw =5 * p
Cmax	958	984	943	893	873	851
CPU	-	-	-	8.9	10.4	21.8
Msg. #	-	-	1521	1043	987	762

(c). λ = 10, no machine breakdowns

	SPT	FIFO	MAS	MASTW		
				tw = p	tw =2 * p	tw =5 * p
Cmax	1008	1016	973	953	933	901
CPU	-	-	-	7.5	11.3	23.2
Msg. #	-	-	1553	1162	1001	721

(d). λ = 4, with machine breakdowns

	SPT	FIFO	MAS	MASTW		
				tw = p	tw =2 * p	tw =5 * p
Cmax	1102	1007	923	901	880	827
CPU	-	-	-	9.3	12.2	44.5
Msg. #	-	-	1552	1282	1006	711

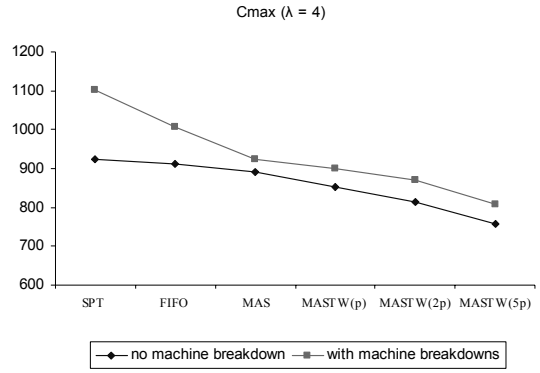
(e). λ = 6, with machine breakdowns

	SPT	FIFO	MAS	MASTW		
				tw = p	tw =2 * p	tw =5 * p
Cmax	1204	1197	989	934	924	912
CPU	-	-	-	10.1	14.5	42.5
Msg. #	-	-	1421	1028	993	652

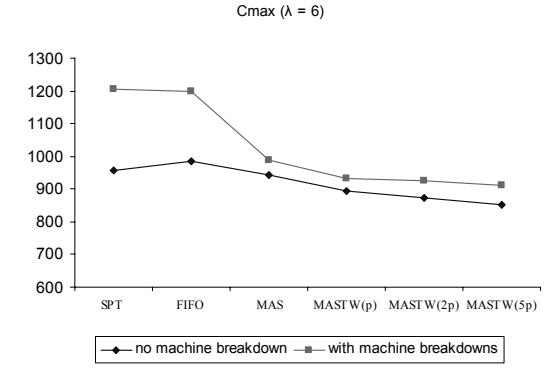
(f). λ = 10, with machine breakdowns

	SPT	FIFO	MAS	MASTW		
				tw = p	tw =2 * p	tw =5 * p
Cmax	1297	1203	994	965	951	931
CPU	-	-	-	11.2	13.1	45.8
Msg. #	-	-	1390	1098	978	741

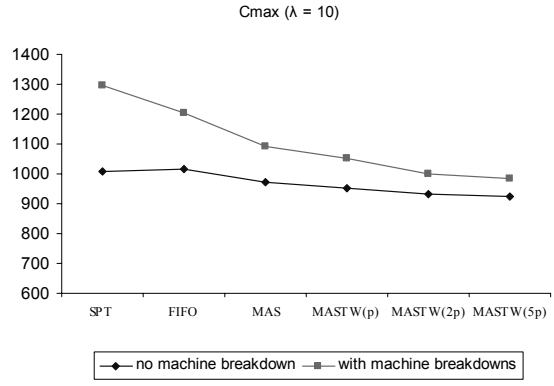
From the simulation results (Figure 5.11), in both experiments, compared with simple dispatching rules, the agent-based scheduling using the proposed methodology has shorter completion time. The difference becomes more significant when system is under perturbation. The simple reason is that agent-based systems are more suitable in dynamic environments.



(a)



(b)



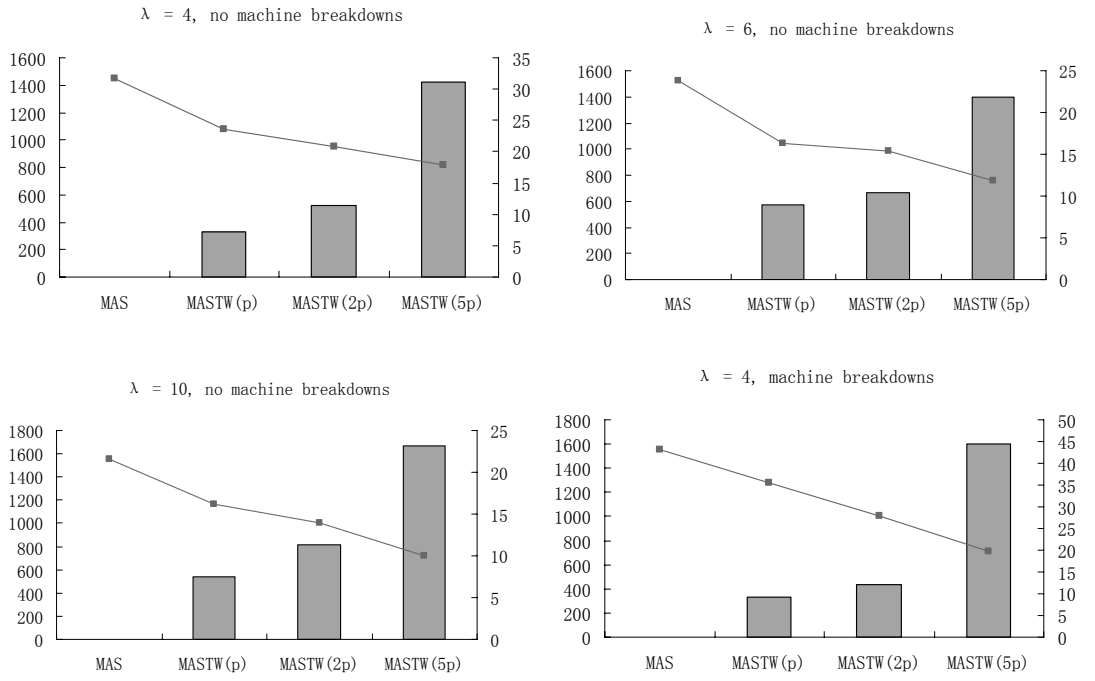
(c)

Figure 5.11. Makespan values of different approaches under different manufacturing load

Another observation can be read from the graphs is that the performance of agent-based scheduling with “look ahead” shows large improvement over the one without using

"look ahead" techniques. The improvement can be explained in a similar way by using the tactically delayed schedules instead of non-delay ones in certain circumstances (Sridharan and Zhou 1996): The model considers at each decision point not only the jobs waiting for that machine in the line, but takes into account other future jobs that need to use that machine and which may become a bottleneck for the shop. In certain cases, it is reasonable not to choose a job from the line waiting for a machine when it becomes available for use, but to keep the machine idle until the bottleneck job is ready be served on that machine. Such a policy results in decrease of make-span.

When compared with MAS, the MASTW is more favorable when system has a higher work load. Based on the simulation results (Figure 5.12), different time window lengths have a big impact on performances. When the time window is larger, a better scheduling result can be reach, however with the CPU time ramp up very rapidly. MAS and a centralized offline scheduling process can be viewed as two extreme cases of MASTW: one has a 0 time length "look ahead" time window, whereas the other has a large enough time window to "view" all of the jobs.



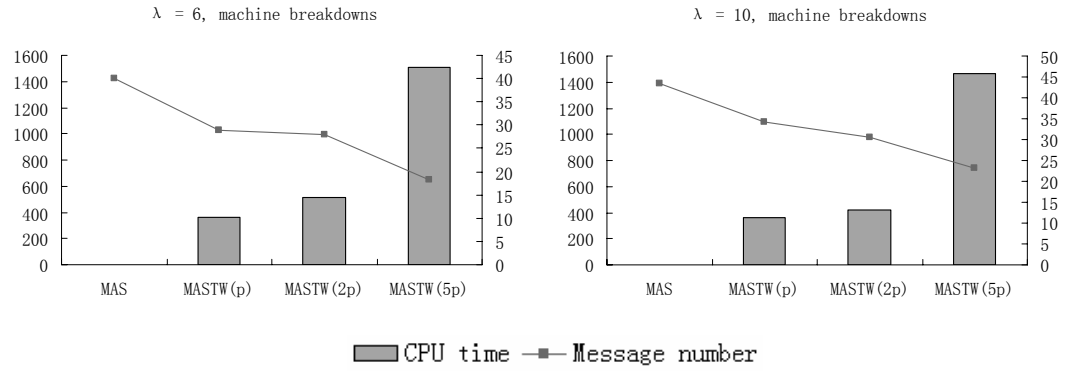


Figure 5.12. CPU times and message numbers consumed in agent-based approaches

The significance of differences among experimental results from different approaches is tested using Tukey's multiple comparison method. The full analysis results calculated based on SAS codes are listed in appendix C.

5.6 summary

In this chapter, some of the commonly neglected problems have been studied and resolved. Based on classification of the three interaction scenarios, local interest conflicts among agents can be detected and resolved. By using TS in "look ahead" time windows when there are potential conflicts. Scheduling problems in "look ahead" time windows are relatively small and can be solved effectively by TS. Tabu search results provide local arrangement of agent-based scheduling decisions. The centralized offline scheduling methodology and traditional multi-agent systems can be considered as two extreme cases of agent-based scheduling with using "look ahead" time windows. The time window length of a centralized offline scheduling method is long enough to cover all the jobs, whereas traditional multi-agent systems are using a time window with length of zero. Simulation results reveal that the proposed methodology has advantages over traditional dispatching rules and multi-agent systems without "look ahead" capability. When time window length is increasing, a better scheduling result can be achieved, however, with cost of more computation load.

Chapter 6. Prototype Implementation and Performance Evaluation

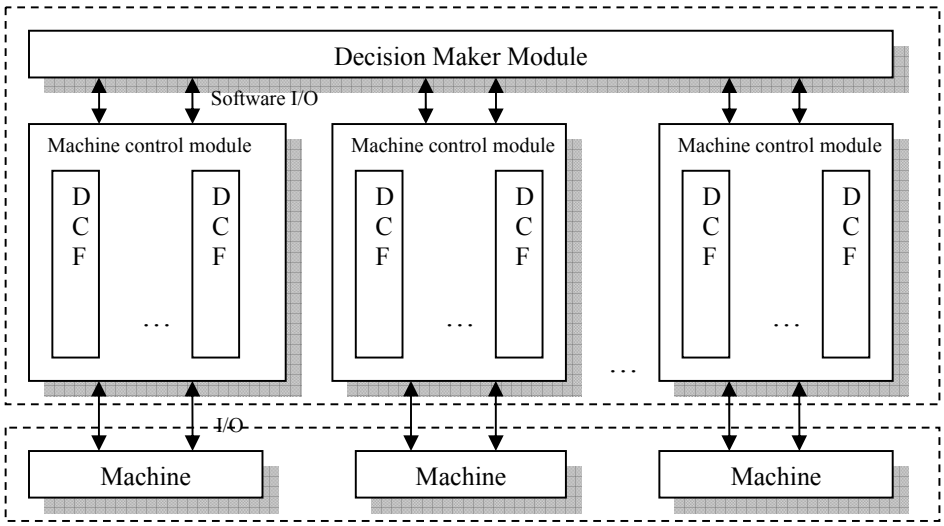
In the previous chapters, a Component-based Intelligent Control Architecture that enables reconfigurability in system-level control for RMS has been designed. The design and development of CICA utilize components throughout the development lifecycle. Intelligent agents are used to handle disturbances that occur in the system. Reconfiguration in a control system with CICA architecture can be readily made both in software layer and intelligence layer, along with physical changes of machineries in a manufacturing system when the market demand changes.

This chapter presents the test and evaluation of the CICA manufacturing control reference architecture. The method is tested by using it to upgrade the existing control software for the table FMS in FMS lab at Virginia Tech. In the following section, an application of the proposed architecture on the example has been detailed. The performance evaluation of the CICA architecture has been evaluated. Because of its inherent modular design, the reference architecture can be easily extended to be the backbone of next generation Enterprise Application Integration (EAI). The final section summarizes this chapter.

6.1 Development of device controls from legacy systems

The laboratory FMS is used as the testbed for development of control software with CICA architecture. Instead of using its original rigid integrated structure, the design of the new control software will use a decentralized approach as shown in Figure 6.1. The machine control modules are composed of device control functions (DCF). These functions perform direct controls on a machine. FMS vendors typically integrate the machine control functions into overall system control software.

Figuring out device control functions is the very first step for upgrading a legacy system to a modular design. A device control is a stand-alone executable program using a set of control functions to control a machine. The device controls can also adapt easily to machine control modules that can hook with other modules in a larger software system.



*DCF: Device Control Function

Figure 6.1 Structure of modular control software

Under the request of FMS users, FMS vendors normally will provide the documentation on how devices in FMS are connected, and the control parameters of each device even including the circuit diagram of the hardware interfaces in addition to the overall control software. However, source codes of the control software may or may not be provided. When source codes are not provided, development of device control programs without vendors' help often time-consuming and difficult. It is possible that source codes of the control software are provided when FMS are installed as in the case of Walli3 control software. In such case, when physical changes of an FMS are considered and the FMS user is hesitant to pay a steep price and to also wait for a long time for the vendor to make necessary changes to the overall control software, developing device controls to enable the building of a reconfigurable control system can be a good solution. Flexible control software, instead of the integrated (dedicated) control software can also be easily used to support research efforts such as testing a scheduling algorithm, which only uses a subset of the full-scale FMS.

FMS control software packages are developed by different vendors specifically for different FMS, they can be coded in different programming languages and they can employ any different logic control structure. Although FMS control software tends to be fairly complicated when taking into account the typical complex operational scenarios of

an FMS, there are still quite a few common features identifiable in such a software system. It is easy to find that many parts of source codes are redundant and therefore discarded while developing individual device controls. The exact functions to control a device are often concise. We decompose the integrated (dedicated) Walli3 control software and use the control functions in it to build device controls.

A device control requires the loading of configuration files, setting up the communication between the device and cell controller, and sending control demand to the device. To develop a device control, it is necessary to find out these functions in the source codes. For a windows control software program, a systematic procedure to develop device control is formalized as follows:

Step 1. Get acquaintance to the control software. The familiarity of the control software features is necessary to understand functions of codes (such as special features of windows). This will enable the discarding of redundant codes later.

Step 2. Find files (*.dll, *.ini, etc) that need to be loaded when the control software's executable file is started. Only when these files are correctly loaded, the source codes of control software can be compiled and debugged. These files are usually needed to be loaded in device control programs in the same way.

Step 3. Find the communication method used in FMS control software. Both physical and software communication methods are helpful to understand the source codes. Communication is always very important to control a device.

Step 4. Find initialization codes in control software by debugging source codes. If the files in step 2 are successfully loaded, the source codes should be able to be compiled and executed. FMS control software need to be initialized at the beginning of execution In this part of codes, communication between cell controller and FMS devices and device parameters are established. Once these codes are identified, they are added in the same way as in overall control software to the device control programs.

Step 5. Find the functions to control a device. The definition of the functions will help us understand where the functions are defined, and names of parameters. At this time, the exact meanings of parameters in the functions are usually unknown.

Step 6. Make the device control program run. With the codes from former steps, the device control program is compiled, corrected, and made to run. Also standard

interface for other software modules needs to be provided. The standard structure (interface) of device control program is currently under study.

Step 7. Find the meaning of the parameters in step 5. Physical experiments need to be performed to decide the parameters. Normally, the names of the parameters in the source codes can give clues about what the parameters are. Changing the parameter once a time, the behavior of a device will provide evident on what the parameter is.

Step 8. This is a very important step. After the device control is developed. It is crucial to document the information of all former 7 steps (initialization files, functions, parameters, interfaces, etc). The document will help users know how to run the device control and how other programs can call the device control.

When there are plenty of comments in the source codes will be great helpful and make it much easier to develop the device control programs. But even when there are limited comments, the development of device control can still be done following the above steps. Codes of device control programs are extracted from vendor-provided supervisory control software. The control software are tested and validated by FMS vendors. Device control programs can be seen as sub-programs from overall control software.

WALLI3 is written in Visual Basic for windows. A number of *.dll files, Visual Basic controls and one configuration file “walli.ini” are loaded when Walli3 starts. The analysis of visual basic controls shows they only provide functions of user interfaces. These user interface feature will not be used in developing device control program. Walli.ini defines initialization information including network addresses of the FMS devices. As we use the same software structure with the overall control software is used, Walli.ini should be included and loaded in the same way as in the device control programs.

Devices in the FMS are connected to cell controller by a proprietary RJ45 serial communications board. The FMS uses a proprietary serial transmission across a link formed from four twisted wire pairs with RJ45 style plugs at each end. Standard windows control “MSComm.vbx” is used to setup communication between FMS devices and cell controller. Information about the API of Windows controls can be found exhaustively in

Microsoft documents. The information of the functions in the control greatly helps for the analyzer to understand communication codes using the Windows control in WALLI3.

Dynamic-link libraries (DLLs) are very important in old windows programs. WALLI3 uses several standard windows dll files and a vendor-furnished “cyber.dll”. All the communication and control routines are stored in “cyber.dll”. So “cyber.dll” must be correctly loaded in device control programs so that all the communication and control routines in it can be used.

The control functions are found by debugging the source codes. For example, the function to control a robot in FMS is represented as “send_rj45_mesg 2, mCNTRL, 10”. The declaration of the function can be found in “walli.glb” as,

Declare Sub send_rj45_mesg Lib "cyber.dll" (ByVal devno As Integer, ByVal mesgaddr As Integer, ByVal mesg As Integer)

Since all the codes needed to control a robot are located they can be used and inserted in the same place as in the overall cell control software. Once device control program is tested to be capable of controlling the device, physical experiments can be carried out to decide the meaning of the parameters in the control functions. In the definition of the function in “walli.glb”, it is easy to conjecture that the first parameter is device number of the robot to be controlled, the second parameter is the message address of the robot, and the third one is the (control) message to be sent. All the three parameters are integer value. From the source codes, it is not difficult to find the values of the last two parameters. Set the first parameter with different values, the device number of a robot can be obtained by simply observing which robot is controlled.

Device controls are made as executable programs. With device controls, users are able to manually control devices in FMS without having to run the cell control program. This also allows users to make sure all desirable functions in device controls work properly. The user interface of a robot arm control program is shown in figure 6.2,

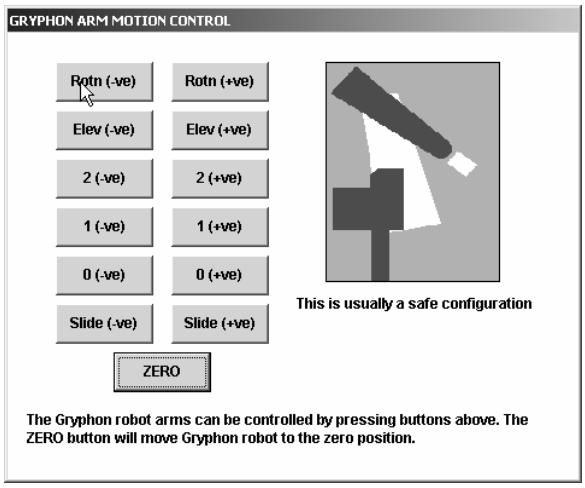


Figure 6.2 Example of device control program's user interface

After the device control software is successfully built up, a complete and easy understand document is very important. The document records initialization and control functions and their parameters (meaning and value) of the device control. Then other programs can easily call incorporate these functions and control the device.

6.1 Implementation of the prototype

The usability and applicability of the method are tested by applying the method to develop a new control system from original rigid control software for the FMS test bed as described in Chapter 3. A small control software demo based on CICA, which is named as Walli3+, has been developed with main elements and techniques introduced in this dissertation.

The main software control modules that have been developed include a web-based manufacturing order interface, a component management module, an agent-based disturbance handling module and a group of machine control modules. Following a process as described in Chapter 3, device controls for main equipments (CNC machines and robots) are implemented based on analyzing source codes of original Walli3 control software.

The original Walli3 control software was developed in a 16bit programming language, which is incompatible with most software nowadays. XML format messages are used as glue for it to interact with other software components. In order to integrate it

with other software modules, an XML parser is attached to the device controls form the machine control module. Figure 6.3 shows the interactions between web-based GUI and machine control modules using XML messages.

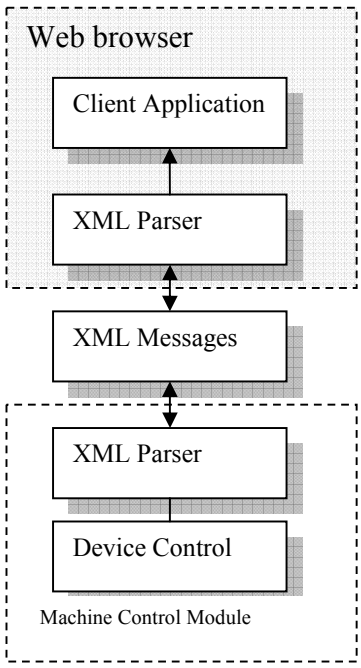


Figure 6.3. Integration between web-based GUI and machine control module

The web-based GUI is designed to put simple manufacturing orders to mimic an e-business environment the current manufacturing systems are facing. Component management module is developed to keep a record of all software components in the system. It also provides yellow page service to all the software components. The web services protocol - WSDL is used to define the services provided software components. An example of WSDL description of a service is listed as following:

```
<definitions>
    <message name="getRobotPosRequest">
        <part name="pos" type="xs:string"/>
    </message>
    <message name="getRobotPosResponse">
        <part name="pos" type="xs:string"/>
    </message>
```

```
<portType name="MoveRobotTo">
        <operation name="getRobotPos">
                <input message=" getRobotPosRequest "/>
                <output message=" getRobotPosResponse "/>
        </operation>
</portType>

<binding type="MoveRobotTo" name="b1">
        <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
          <operation>
                    <soap:operation
                    soapAction="http://example.com/getRobot"/>
                  <input>
                     <soap:body use="literal"/>
                  </input>
                  <output>
                     <soap:body use="literal"/>
                  </output>
           </operation>
        </binding>
</definitions>
```

An agent-based management module is developed to provide exception handling functionality. Figure 6.4 is an overview of the software system.

Because of the modular design of CICA and using of web services protocols, different control modules are implemented with different software languages, and can be run on different computers. For example, the web-based GUI is developed using Microsoft ASP.NET 2.0. The component management component is developed using VB.NET, and the agent-based exception handling module is developed based JAVA

language. All the modules exchange XML format messages based on web service protocols.

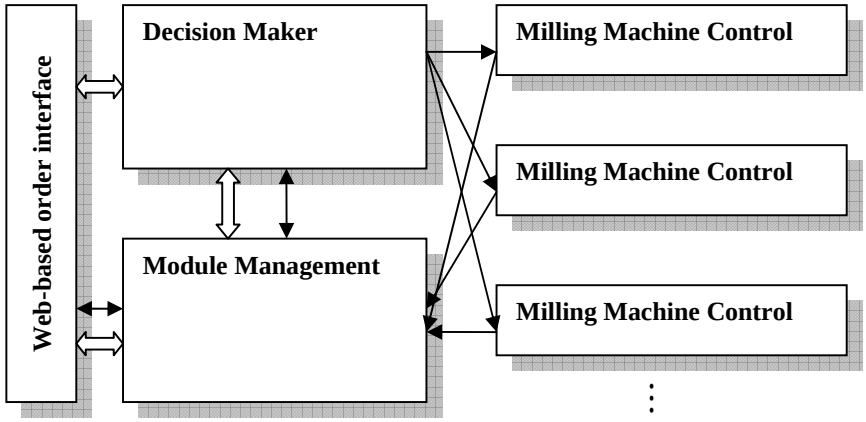


Figure 6.4. Overview of the Walli3+ software system

As compared with the existing Walli3 control system and most popular design of control systems, the control software based on CICA architecture can be easily developed, maintained and extended.

6.2 Performance evaluation of CICA

There are no universal accepted methods for the evaluation of reference architectures revealed in existing literature. The choice of the performance measurements to be used highly depends on the business domain. For some applications the throughput is the major performance indicator, but other can consider the robustness or service quality the focus in the performance measurement.

In the manufacturing context, performance measurements addressing the relevant aspects to evaluate a manufacturing control system can be classified as qualitative or quantitative. The quantitative measures are based on different production performance indicators, such as the manufacturing lead time, machine utilizations, the waiting time, the throughput and the WIP, etc. Chapter 5 has already given a quantitative evaluation of agent-based scheduling performances of CICA. The evaluation results reveal that the agent-based scheduling process with using “look ahead” technique has advantages over

simple dispatching rules and most current agent-based systems. When a global optimal algorithm is available for the manufacturing system, the agent-based exception handling process is able to have the advantages of both achieving global optimum by using centralized methodology and at the same time, the flexibility of handling unknown exceptions in the system by using the agent-based system.

The qualitative measures are of a more subjective nature and reflect properties of the manufacturing control solution, such as the agility, flexibility and robustness, which cannot be directly obtained from the production data.

Wyns (1999) listed three possible ways of evaluation to assess the quality of a reference architecture in his research dissertation: "requirement satisfaction evaluation", "direct comparison" and "stakeholders critiques". The "requirement satisfaction evaluation", which is claimed to be the most scientific out of all the three approaches, is a method that lists a number of requirements that the solution should satisfy and then measures the satisfaction of these requirements quantitatively or qualitatively. The "direct comparison" approach is the approach that compares characteristics of two competing products directly. The "Stakeholders critiques" approach is based on feedbacks from customers, plant managers, control software developers, etc. How the product is accepted by public can be determined from the comments. The following part of this section utilizes the three approaches to evaluate the CICA performances.

In the first chapter of this dissertation, a list of requirements has been given for the system-level control of RMS. These key features include:

- Reconfigurability
- Reusability
- Robustness
- Disturbance handling
- Extensibility

Figure 6.5 displays the map between the fulfillment of reference architecture requirements and technologies that have been used in CICA.

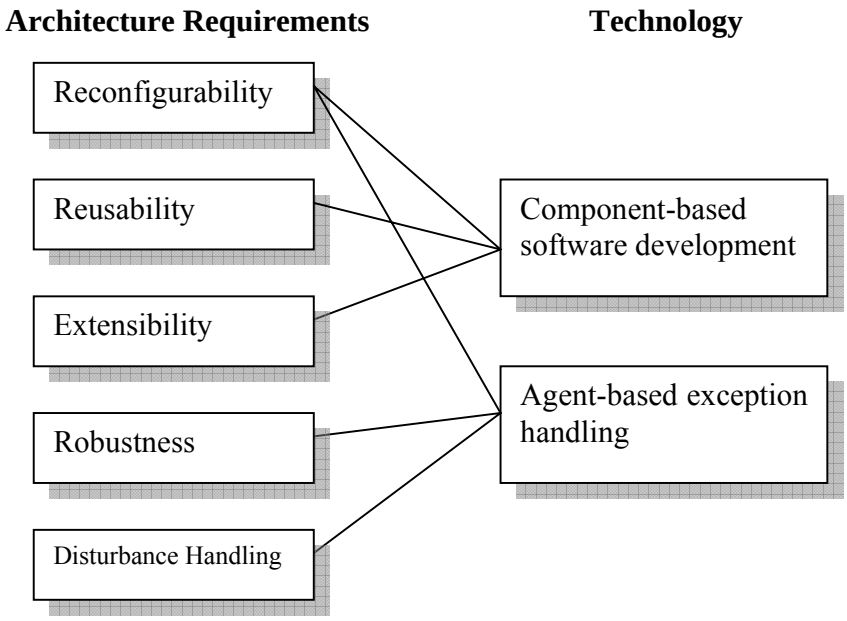


Figure 6.5. Mapping of architecture requirements to techniques used in CICA

Besides the listed requirements, CICA can also fulfill other important architecture requirements that are commonly used to evaluate performance. One of such requirements is low building and maintenance cost. By using CBSD, CICA leads to a short developing time, low development cost, re-use of components, and the ease of debugging of manufacturing control software. It employs a low complexity, homogeneity of concepts, and intuitive terminology.

CICA is also able to fulfill the requirement for compliance with existing systems by providing an easy way to integrate with legacy systems. This translates to a requirement to support migration and evolution. Flexibility with respect to product changes, process and equipment changes, and volume changes, requires the system architecture to be extendable and adaptable. The migration and evolution supported by CICA is able to fulfill the need of such flexibility.

Another requirement that a generic reference control architecture needs to stress is functional completeness, which is the ability to incorporate a wide range of algorithm solutions for resource allocation and technology planning. Functional completeness can be illustrated by mapping all functionality of different manufacturing systems into the

architecture: monitoring, maintenance planning, fault detection, NC program generation, etc.

The software demo based on CICA architecture comparing with original Walli3 control software provides a direct comparison between CICA reference control architecture and traditional centralized/hierarchical architecture. The Walli3+ is able to demonstrate all the listed architecture requirements. It is much easier to make different changes, which was impossible with Walli3. It is now possible for researchers in the laboratory to use partial FMS for research activities. The software issue is no long the big hurdle when adding new equipments or new technologies into the system. Besides the big flexibility brought by CICA, Walli3+ is also more robust because of the implementation of agent-based exception handling that is able to handle different disturbances that would stop the whole system before. When comparing CICA with another popular control architecture - holonic manufacturing control systems, CICA has the advantage of easiness to develop, and being align with software industry standards. The holonic way of development introduces holoarchy concepts, which bring huge complexity to system design. Current studies on design of holonic systems also only start from scratch. The ability of incorporating legacy systems is largely unknown. The CICA architecture, however, is easy to integrate with legacy systems with simply adding adapters for most times. The component-based software development of CICA also has significant advantage over the traditional pure object-oriented software development. A more detailed comparison between component-based development and object-oriented development can be found in (Szyperski 1998, Mat 1999)

The current study of this research is conducted totally inside the laboratory without any industry involvement. Hence, the “stakeholder critiques” approach is hard to be applied. However, the researchers from FMS lab show great interest to use the new FMS with upgraded control software. One major advantage of the new design based on CICA is the flexibility of conducting research on different granular level of the FMS. The involvement of experts from manufacturing industry is necessary to make the CICA concept successful. The experts can built a full version control systems based on CICA concepts. A survey can be provided for the experts and users for performance evaluation. A group of metrics, rating levels, and evaluation criteria can be chosen for each quality

characteristic. A special procedure can be developed to summarize the measured values for the selected metrics. Based on the survey output and fuzzy logic, numerical weights can be added for quantitative evaluation of qualitative performances.

In certain scenarios, metrics to measure software systems can also be used to evaluate manufacturing control reference architectures sometimes. Most current control systems in practices or research are based on object-oriented design. CICA is based on a component based design instead. According to Bill Gates, the component based develop is "the Industrial Revolution of software". He believes the component concept in software industry is same as the "specialization of resources, standards for interchangeable parts, and streamlined assembly tools have been used in other industries for hundreds of years to speed the development of highly complex products". Table 6.1 lists the major difference between components and objects.

Table 6.1. The major differences between objects and components

	Objects	Components
Implementation	Bound to OO languages (C++, Smalltalk, etc)	Can be written in any language
Modularity	Tight couplings (implementation inheritance)	More loosely coupled than objects (interface inheritance) Functional aggregation
Granularity	Fine granularity in object decomposition	Large granularity
Accessibility	Limited forms for accessing (methods)	Interface oriented

As compared with OOP and other software development approaches, CBSD assists in reducing the development cost and time-to-market, as well as improve maintainability and reliability. One of the key problems in CBSD is how to evaluate/certify the quality of individual components and the component-based software systems assembled. This remains an open topic in CBSD research. Torre (2004) gave an overview of software

component evaluation approaches. He noticed most proposals nowadays were not validated with any industry-level experiments.

6.3 Enterprise integration based on an extension of CICA

The CICA reference architecture is designed for system-level control for future manufacturing systems – RMS. However, the design can be easily extended to be backbone for the overall enterprise infrastructure. In the end of 1990s, the term Enterprise Application Integration (EAI) has been created to address the need to integrate both intra and inter-organizational systems through incorporating functionality from different applications (Figure 6.6). Another critical reason for the emergence of EAI is the desire of organization to meet the rapid changes of business environments. EAI technologies enable enterprise to cope with changes in the market both promptly and flexibly by allowing further development of information systems to adapt changes while making use of existing software resources. EAI can be defined as the unrestricted sharing of data and business processes amongst any connected applications and data sources in the enterprise

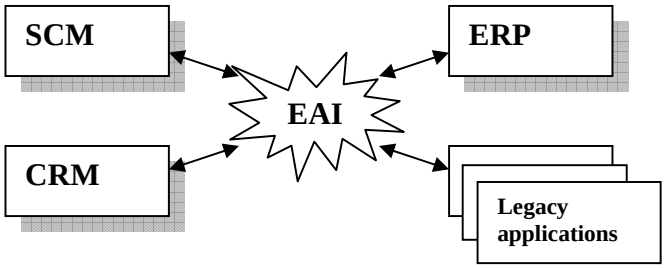


Figure 6.6. Machining features and their tool access directions

As companies continue to add new technologies to their information systems, and get more involved in e-commerce, they face the daunting task of integrating disparate IT applications. EAI approaches mainly happen in three levels: Data level integration, Application level integration and User Interface level integration. Traditional EAI approaches focused on converting and exchanging data between applications. First generation EAI is based on point-to-point connections. However, the point-to-point connections create tightly coupled applications. Business information systems became

complex and hard to manage as the size of systems grows. The Brittle of point-to-point approaches lead to the development of adaptors and connectors. In this way, all the applications within an organization can communicate and exchange data as needed through middleware, rather than through customized application-to-application integrations. Gorton and Liu (2000) summarized types of middleware useful for integration: Message Queues, Publish-subscribe messaging, Application servers, Message brokers, and Business process automation.

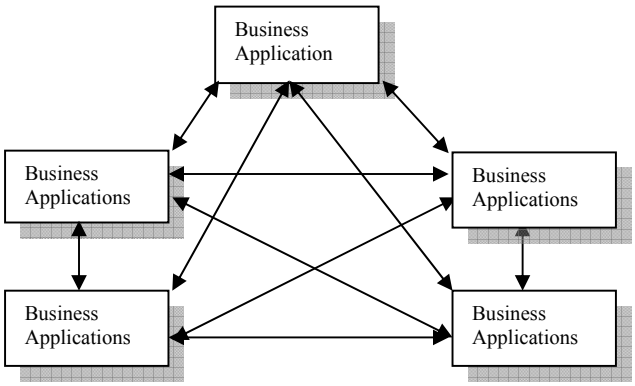
Numerous software companies have developed EAI products based on widely utilized architectural patterns, such as Java J2EE, Microsoft .Net, etc. However, it is still hard for applications based on different architecture patterns to communicate nowadays. Open, standards-based, and vendor-neutral platforms with which one can perform system integration across the enterprise will be an important prerequisite for the future success of EAI implementation.

Tillman (2001) stated only Modular relationships provide firms with greater flexibility, timely response to market changes, and a greater competitiveness in the short run. The EAI tools will eventually provide a "plug-and-play" environment for business information systems, where systems configuration rather than systems programming will be the main consideration. Current EAI technologies based on open standards such as CORBA, J2EE and .NET provide a modular framework for enterprise information architectures. However, existing literatures are mostly dealing with EAI frameworks based on one of the specific architectural patterns, such as CORBA, J2EE or .NET. Another trend in EAI is the movement away from information-oriented to service-based integration. The Web services technology allows pieces of software to communicate without the interoperability problems that the traditional EAI solutions have difficulties to solve. With minimal programming, the Web services technology can also be used to easily and rapidly wrap the legacy enterprise applications and expose their functionality through a uniform and widely accessible interface over the Internet. Web services are a promising paradigm for effective enterprise-scale system development and integration. These self-contained and self describing business-driven functional units can be plugged-in or invoked across the Internet to provide flexible enterprise application integration within the Service-Oriented Architecture (SOA) (Kaye, 2003).

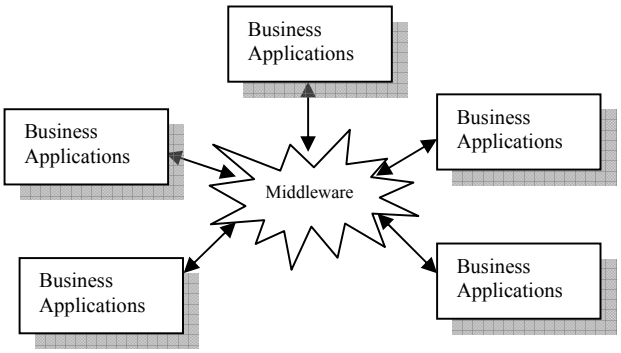
The component and service-based EAI framework has the same features with CICA as a highly distributed, loosely coupled and open-standard based. Components will be loosely coupled and they are connected by open data format. Services only communicate with one another by messages. Messages are sent in a platform-neutral, standardized format delivered through the interfaces. XML is the data format that meets this constraint. XML has already become an accepted standard for data interchange. XML is used to exchange data between applications independently of their implementation languages and runtime environments. The benefit XML provides is that rather than using a proprietary internal data format, an XML EAI tool uses a completely open and self-describing data format.

Reengineering legacy applications for component and service-based EAI framework is typically accomplished by building an adapter that exposes the legacy application's functionality through open protocol. For example, a Web-service adapter is a wrapper program that talks to the legacy application using its native protocol at one end and talks to the rest of the world in SOAP at the other end.

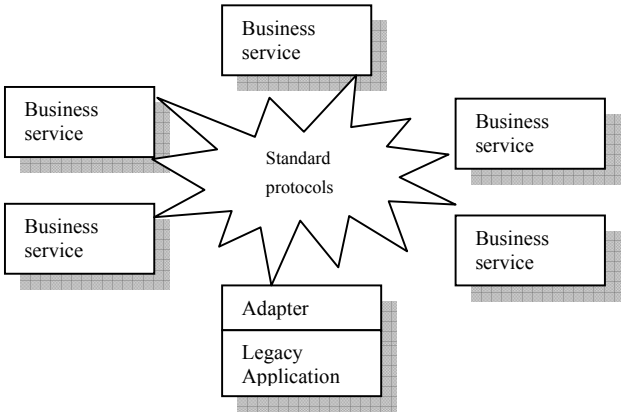
Compared with the first generation EAI's point-to-point connection, and the current middleware architecture, next generation of EAI architecture will be based on standard interaction protocol and open format data message (Figure 6.7). Software components will be very well specified and only services are exposed to outside.



a. Point-to-Point approach



b. Middleware approach



C. Component and service -based approach

Figure 6.7. Comparison of different EAI frameworks

Not only the component-based development side of CICA can be utilized in the next generation EAI, the agent-based systems used in CICA can also be universally applied in different modules in the system. The agents are able to interact with each other automatically with producing favorable outputs. The ability to handle dynamics makes agents most suitable in a current fast-changing business environment.

Figure 6.8 depicts an overview of an enterprise infrastructure based on CICA alike EAI framework.

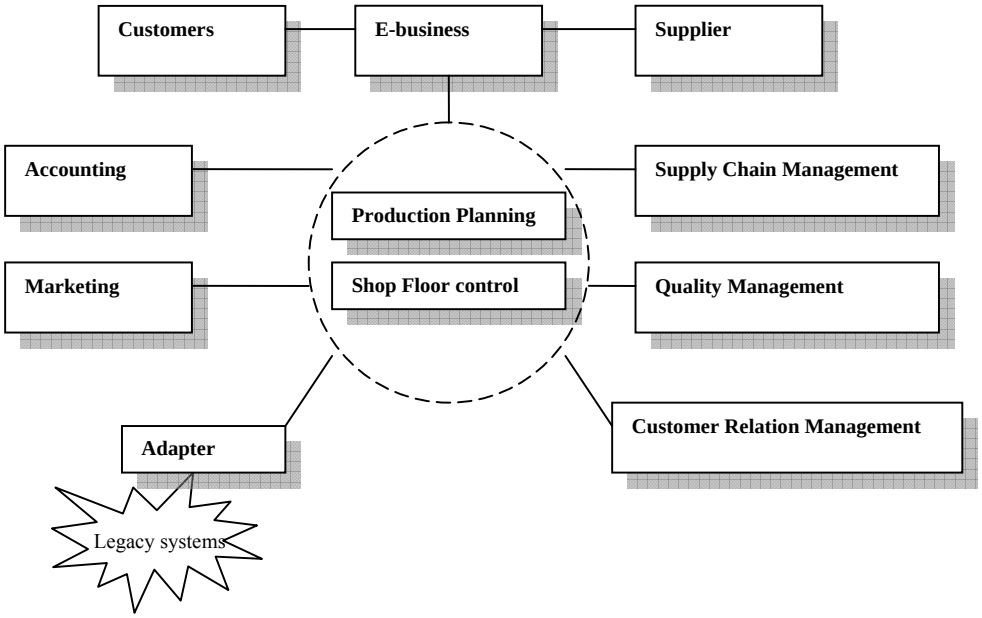


Figure 6.8. A full view of an enterprise infrastructure based on CICA

6.4. Summary

In this chapter, we present the test and evaluation of our method based on the usability of the method, users' evaluations of the method and potential impact and added values the method brings when compared to with existing development methods. The laboratory FMS is used as the testbed for development of control software with CICA architecture. The device control functions (DCF) that perform direct controls on a machine form the main elements in the machine control modules. A generic way of developing device controls from legacy control systems is presented. The software demo based on CICA architecture comparing with original Walli3 control software provides a direct comparison between CICA reference control architecture and traditional centralized/hierarchical architecture. The new control system based on CICA architecture is able to demonstrate all the architecture requirements listed in the first chapter. In the last section of this chapter, the CICA concept is also shown to be easily extended for base of tomorrow's EAI framework.

Chapter 7. Conclusions and Future Work

This chapter summarizes the conclusions and main contributions of this doctoral dissertation. Additionally, it indicates future research and development directions that would help to bring the advantages of the CICA reference architecture into industrial practice.

7.1 Summary of conclusions

Nowadays, manufacturing enterprises are facing frequent changes nowadays. Such changes include more customized demand, increased product variety, newly enforced laws, etc. RMS is a new manufacturing paradigm that is developed to address these changes confronting with new manufacturing environments. It has been widely recognized as one of the most important approaches to increase competitiveness of manufacturing enterprises. Extensive research efforts have been conducted in reconfigurable machining systems so far, however, manufacturing system-level reconfigurable design has not been done as evidenced by literature to date. In reality, the system-level reconfigurability is the key element that enables a RMS to achieve its promises.

Currently, the manufacturing control systems are adapted in response to the changes on a case-by-case basis by introducing expensive, time-consuming, ad-hoc solutions. These are not only too expensive to create, but also equally too cumbersome to maintain since any future change will require an adaptation to this ad-hoc solution. The most popular hierarchical and heterarchical reference architectures do not offer this required re-configurability.

A more profound approach is to construct the manufacturing control system such that changes can be more easily incorporated. Therefore new manufacturing control systems shall be based upon a generic, re-configurable manufacturing control reference architecture, which is independent of logistical and technical aspects. As RMS is designed to accommodate changes, its control systems need to be easily reconfigured along with changes in machinery of the system when system reconfiguration is needed.

This research is among the first few attempts that specifically study system-level control reference architecture for RMS. The component-based design and agent-based exception handling enable CICA architecture to provide an easy way to enable reconfigurabilities in both software structure and control logic for RMS control systems. The reference architecture is able to fulfill all the requirements listed for a RMS system control without introducing extra complexity as other new emerging methodologies such as holonic manufacturing control systems.

There are already studies that propose to use component-based development methodology to develop manufacturing control systems. However, most of these studies are based on a certain component-based framework, such as, .NET, or CORBA. By using web-services protocols and XML format messages, the proposed CICA allows integration among different components without knowing the language or operation systems they are based on, as long as their interfaces fit. In that sense, the specification of a component is more important than how it is implemented. A UML-based precise specification process for component development is presented in this dissertation.

Agent-based systems are used to handle exceptions because agent-based design has advantages to deal with dynamics. However, its performance is hard to predict and can be inferior to a centralized decision methodology under stable conditions. The agent-based design adds reconfigurabilities to control logic in control systems. A stepwise decision process with details of agent types and their decision logic has been designed for agent-based decision process. Some widely neglected problems in the agent-based design such as communication burden, local interest conflicts have also been studied in this research. Three basic scenarios are classified for agent negotiation situations. A "look ahead" technique is used to capture potential local interest conflicts among agents. The philosophy of the new decision process to resolve the conflicts is utilizing a methodology that is between two extreme cases of centralized methodology and fully decentralized agent-based decision process. A metaheuristic – Tabu search shows effective to solve the scheduling problems inside "time windows". The experimental analysis shows that the agent-based decision system using TS in "look ahead" time windows outperforms those using simple dispatching rules and most current agent-based systems.

A small control software demo based on CICA is developed for a table-top flexible work cell in FMS lab at Virginia Tech (the systems is now in the Flexible Manufacturing and Lean Systems Lab at University of Texas at San Antonio). A generic methodology to develop device controls from legacy systems is developed in this study. Based on the machine control modules developed from the device controls and other main functional modules, the new control system shows to be easily developed, maintained and extended than the original control software. The researchers using this flexible manufacturing cell will have much more flexibilities to change the system to accommodate their own research. This is impossible to do by using its original vendor-provided, proprietary control software. The CICA control architecture is developed to provide true reconfigurable system-level control for RMS, however, the same framework can be easily extended to be the backbone of the future enterprise infrastructure.

7.2 Future research directions

The use of the CICA reference architecture shall lead to the development of high quality manufacturing control systems that are suitable for current rapidly changing manufacturing environment. The aim of this research is to eventually lead to the reference control architecture applicable in real manufacturing environments.

To achieve this objective, domain experts from manufacturing and software engineering sectors need to work together to conduct further studies into this architecture. The implementations shall also be developed in close co-operation with customers to learn in an early phase about their requirements and the acceptance of the proposed solutions. It is necessary to initiate an industrial development activity to bring CICA-based manufacturing control systems into practice.

Some future research topics as the continuation of this research include standardization of component specification and inter-communications, quality assessment of components and final control software, and a more detailed study of exception definition and handling.

Currently, only limited functions have been developed in the software demo due to the limitation on developer's work hours. Due to the extensibility of the CICA architecture, it is easy to add new feature/functions to existing control software. A more

sophisticated control software based on CICA can provide more insights into its performance.

References

1. Ambra, C., Oldknow, K., Migliorini, G., and Yellowley, I. (2002) "The design of a high performance modular CNC system architecture" *Proceedings of the 2002 IEEE Internatinal Symposium on Intelligent Control*, 290-296.
2. Arentsen, A.L. (1995) “A generic architecture for factory activity control”, Ph.D. Thesis, University of Twente, Enschede.
3. Ayres, R. U. (1992). "The diffusion of FMS." *CIM*, II, 197-248.
4. Aytug, H., Lawley, M. A., McKay, K., Mohan, S., and Uzsoy, R. (2005). "Executing production schedules in the face of uncertainties: A review and some future directions." *European Journal of Operational Research*, 161(1), 86-110.
5. Babiceanu, R. F. and Chen, F. F., (2006) “Development and Applications of Holonic Manufacturing Systems: A Survey” *Journal of Intelligent Manufacturing*, 17(1), 111-131.
6. Balasubramanian, S., and Norrie, D. H. (1996) "Intelligent manufacturing system control." *Canadian Conference on Electrical and Computer Engineering*, 570-573.
7. Basnet, C., and Mize , J. (1994). "Scheduling and Control of Flexible Manufacturing Systems: A Critical Review." *International Journal of Computer Integrated Manufacturing*, 7(6), 340-355.
8. Beek, D. A. v. (1993). "Exception Handling in Control Systems," Eindhoven University of Technology.
9. Ramirez-Serrano, A. and Benhabib, B. (2003). "Supervisory control of reconfigurable flexible-manufacturing workcells - temporary addition of resources." *International Journal of Computer Integrated Manufacturing*, 16(2), 93-123.
10. Berbers, Y., Steegmans, E., and Holvoet, T. (2002) "Agent-based Shop Floor Control using Jini." *Twentieth IASTED International Conference Applied Informatics (AI 2002)*, Innsbruck, Austria, 351-463.
11. Bertok, P., Haidegger, G., Kovacs, G., Letray, Z., and Mezgar, I. (1998) "Design aspects of reconfigurable manufacturing cells as building blocks of flexible

manufacturing systems." *Proceedings of IEEE International Symposium on Intelligent Control,* 751-756.

12. Brennan, R. W. F., M.; Norrie, D.H. (2002). "An agent-based approach to reconfiguration of real-time distributed control systems." *IEEE Transactions on Robotics and Automation*, 18(4), 444-451.
13. Brennan, R. W. and William, O. (2000) "A simulation test-bed to evaluate multi-agent control of manufacturing systems." *Winter Simulation Conference Proceedings,* 1747-1756.
14. Bruccoleri, M., Amico, M., and Perrone, G. (2003). "Distributed intelligent control of exceptions in reconfigurable manufacturing systems." *International Journal of Production Research*, 41(7), 1393-1412.
15. Bruccoleri, M., Pasek, Z. J., and Koren, Y. (2006). "Operation management in reconfigurable manufacturing systems: Reconfiguration for error handling." *International Journal of Production Economics*, 100(1), 87-100
16. Brussel, H. V., Bongaerts, L., Wyns, J., Valckenaers, P., and Ginderachter, T. V. (1999). "A conceptual framework for holonic manufacturing: Identification of manufacturing holons." *Journal of Manufacturing Systems*, 18(1), 35.
17. Bauer, A., Browden, R., Browne, J., Duggan, J., & Lyons, G., (1994) “Shop Floor Control Systems - From design to implementation”, 2nd edition, Chapman & Hall, ISBN: 0412581507.
18. Bongaerts, L. (1998) "Integration of Scheduling and Control in Holonic Manufacturing Systems", Ph.D. thesis, Katholieke Universiteit Leuven, Belgium.
19. Cakmakci, M., Mehrabi, M. G., and Ulsoy, A. G. (1998). "State-of-the-Art in Technologies Related to Reconfigurable Manufacturing Systems." Department of Mechanical Engineering and Applied Mechanics, University of Michigan, Ann Arbor.
20. Capretz, L. F. C., M.A.M.; Dahai Li. (2001) "Component-based software development." *Industrial Electronics Society, IECON '01. The 27th Annual Conference of the IEEE*, 1834-1837 vol.3.
21. Chan, F. T. S., Zhang, J., Lau, H. C. W., and Ning, A. (2000) "Object-oriented architecture of control system for agile manufacturing cells." *Proceedings of the*

2000 IEEE International Conference on Management of Innovation and Technology, ICMIT 2000, 863-868 vol.2.

22. Cheesman, J., and Daniels, J. (2000). *UML Components: A Simple Process for Specifying Component-Based Software*, Addison-Wesley Professional.
23. Chen, B.-C., Tilbury, D. M., and Ulsoy, A. G. (1998) "Modular Control for Machine Tools: Cross-Coupling Control with Friction Compensation." *Proceedings of the ASME-IMECE Dynamic Systems and Control Division*, Anaheim, California, 455-462.
24. Cheung, H. M. E., Yeung, W. H. R., NG, H. C. A., and Fung, S. T. R. (2000). "HSCF: a holonic shop floor control framework for flexible manufacturing systems." *International Journal of Computer Integrated Manufacturing*, 13(2), 121-138.
25. Chirn, J.-L., and McFarlane, D. C. (2000) "A Holonic Component-based Approach to Reconfigurable Manufacturing Control Architecture." *HolonMas00*, London.
26. Cicirello, V. (2001) "A Game-Theoretic Analysis of Multi-Agent Systems for Shop Floor Routing", tech. report CMU-RI-TR-01-28, Robotics Institute, Carnegie Mellon University, September, 2001.
27. Cook, S., and Daniels, J. (1994). Designing Object Systems, Prentice Hall.
28. D'Souza, D. F., and Wills, A. C. (1999). *Objects, Components, and Frameworks with UML - the Catalysis Approach*, Addison-Wesley.
29. Daeyoung, C., Park, C., Sukho, K., and Park, J. (1996). "Developing a shop floor scheduling and control software for an FMS." *Computers & Industrial Engineering*, 30(3), 557-568.
30. DeGaspari, J. (2002). "All in the family: Flexible machining systems give manufacturers a hedge on their bets." *Mechanical Engineering Magazine*, 124(2).
31. Deshmukh, T. M. a. A. (1999). "Caution! agent based systems in operation." *InterJournal of Complex Systems*, (265).
32. Dilts, D. M., Boyd, N. P., and Whorms, H. H. (1991). "The evolution of control architectures for automated manufacturing systems." *Journal of Manufacturing Systems*, 10(1), 79-93.

33. Duffie, N. A., and Piper, R. S. (1987). "Non-hierarchical control of a flexible manufacturing cell." *Robotics and Computer-Integrated Manufacturing*, 3(2), 175-179.
34. Dumitrache, I., Caramihai, S. I., and Stanescu, A. M. (2000) "Intelligent agent-based control systems in manufacturing." *Proceedings of the 2000 IEEE International Symposium on Intelligent Control,* 369-374.
35. Dumitrache, I. C., S.I.; Stanescu, A.M. (2000) "Intelligent agent-based control systems in manufacturing." *Proceedings of the 2000 IEEE International Symposium on Intelligent Control,* 369-374.
36. Ferber, J. (1999). *Multi-Agent Systems: An Introduction to Distributed Artificial* Intelligence, Addison-Wesley Professional, MA, USA.
37. Fowler, M. (2003). UML Distilled: A Brief Guide to the Standard Object Modeling Language, Addison-Wesley Professional.
38. Furness, R. J., Ulsoy, A. G., and Wu, C. L. (1996). "Feed, Speed, and Torque Controllers for Drilling." *Transactions of the ASME*, 118, 2-9.
39. Garro, O., and Martin, P. (1993). "Towards New Architecture of Machine Tools." *International J. of production resources*, 31(10), 2403~2414.
40. Gorton, I., and Liu, A. (2004) "Architectures and technologies for enterprise application integration." *Proceedings of 26th International Conference on Software Engineering, ICSE 2004*, 726-727.
41. Grabot, B., and Huguet, P. (1996). "Reference models and object-oriented method for reuse in production activity control system design." *Computers in Industry*, 32(1), 17-31.
42. Heikkila, T., Kollingbaum, M., Valckenaers, P., and Bluemink, G.-J. (2001). "An agent architecture for manufacturing control: manAge." *Computers in Industry*, 46(3), 315-331.
43. Hoagland, J. (2005) "From Object Oriented to Component Based Software." *http://www.components-online.com/Component/Software/default.htm*.
44. Howard, A., Kochhar, A., and Dilworth, J. (1998). "An objective approach for generating the functional specification of manufacturing planning and control

systems." *International Journal of Operations & Production Management*, 18(8), 710.

45. Huang, H. M., Messina, E., Scott, H., Albus, J., Proctor, F., Shackleford, W. (2001) "Open System Architecture for Real-time Control Using a UML Based Approach." *Proceedings of the 1st ICSE Workshop on Describing Software Architecture with UML*, Toronto, Canada.
46. Ito, Y. (1988). "The production environment of an SME in the year 2000." *Flexible Manufacturing for Small and Medium Enterprises*, 207~234.
47. Kalita, D., and Khargonekar, P. P. (2002). "Formal verification for analysis and design of logic controllers for reconfigurable machining systems." *Robotics and Automation, IEEE Transactions on*, 18(4), 463-474.
48. Katz, R., Min, B.-K., and Pasek, Z. (2000). "Open Architecture Control Technology Trends." *ERC/RMS Report 35*.
49. Kendall, E. A. (2000). "Role modeling for agent system analysis, design, and implementation." *IEEE Concurrency*, 8(2), 34-41.
50. Koestler, A. (1967). *The Ghost in the Machine*, Arkana Books, London.
51. Kolla, S., Michaloski, J., and Rippey, W. "Evaluation of component-based reconfigurable machine controllers." *Proceedings of the 5th Biannual World Automation Congress, 2002*. 625-630.
52. Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritchow, G., Brussel, V., and H., U., A.G. (1999). "Reconfigurable Manufacturing Systems." *CIRP Annals*, 48(2).
53. Koren, Y., Ulsoy, A.G. (2002). "Vision, Principles and Impact of Reconfigurable Manufacturing Systems." *Powertrain International*, 14-21.
54. Kotak, D., Wu, S., Fleetwood, M., and Tamoto, H. (2003). "Agent-based holonic design and operations environment for distributed manufacturing." *Computers in Industry*, 52(2), 95-108.
55. Kozaczynski, W., and Booch, G. (1998). "Component-Based Software Engineering." *Software, IEEE*, 15(5), 34-36.
56. Kraus S., (1997), "Negotiation and cooperation in Multi-agent environment", Artificial Intelligence, 94(3), 79-97

57. Lee, J. (1997). "Overview and perspectives on Japanese Manufacturing Strategies and Production Practices in Machinery Industry." *ASME J. Of Manufacturing Science*, 119, 726~731.
58. Lian, F.-L., Moyne, J. R., and Tilbury, D. M. (2000) "Implementation of Networked Machine Tools in Reconfigurable Manufacturing Systems." *Proceedings of the Japan-USA Symposium on Flexible Automation*, Ann Arbor, MI.
59. Lim, M. K., and Zhang, Z. (2003). "A multi-agent based manufacturing control strategy for responsive manufacturing." *Journal of Materials Processing Technology*, 139(1-3), 379-384.
60. Mansfield, E. (1993). "New evidence on the economic effects and diffusion of FMS." *IEEE Transactions on Engineering Management*, 40(1), 76-78.
61. Maturana, F. and Norrie, D.H. (1996) "Multi-agent mediator architecture for distributed manufacturing", *Journal of Intelligent Manufacturing* 7, 257-270.
62. Mehrabi, M. G., Ulsoy, A. G., and Koren, Y. (2000a). "Reconfigurable manufacturing systems and their enabling technologies." *International Journal of Manufacturing Technology and Management*, 1(1), 113.
63. Mehrabi, M. G., Ulsoy, A. G., and Koren, Y. (2000b). "Reconfigurable manufacturing systems: Key to future manufacturing." *Journal of Intelligent Manufacturing*, 11(4), 403.
64. Mehrabi, M. G., Ulsoy, A. G., Koren, Y., and Heytler, P. (2002). "Trends and perspectives in flexible and reconfigurable manufacturing systems." *Journal of Intelligent Manufacturing*, 13(2), 135.
65. Messina, E., Horst, J., Kramer, T., Huang, H., and Michaloski, J. (1999). "Component Specifications for Robotics Integration." *Autonomous Robots*, 6(3), 247 - 264.
66. Monostori, L. K., B. "Agent-based control of manufacturing systems." *Intelligent Processing and Manufacturing of Materials, 1999. IPMM '99. Proceedings of the Second International Conference on*, 131-137 vol.1.
67. Morton, Y. T., Troy, D. A., and Pizza, G. A. (2002) "An approach to develop component-based control software for flexible manufacturing systems." *Proceedings of the 2002 American Control Conference,* 4708-4713 vol.6.

68. NRC. (1998). *Visionary Manufacturing Challenges for 2020*, National Academy Press, Washington, DC.
69. NSF. (1996). "National Science Foundation Engineering Research Center Fact Sheets." *http://www.nsf.gov/pubs/2000/nsf00137/nsf00137l.htm*.
70. Overmars, A. and Toncich, D. (1996) "Hybrid FMS control architectures based on holonic principles," *International Journal of Flexible Manufacturing Systems*, vol. 8, 263-278.
71. Park, I., Tilbury, D., and Khargonekar, P. P. (1998) "A Formal Implementation of Logic Controllers for Machining Systems Using Petri Nets and Sequential Function Charts." *1998 Japan-US Symposium on Flexible Automation*, Japan.
72. Pasek, Z. J., Holtz, C., Benchetrit, U., and Wang, W.-l. (2000) "Infrastructure for system-level configuration and control of manufacturing devices." *2000 Japan-USA sysposium on Flexible Automation*, Ann Arbor, MI.
73. Prabhu, P., Sharit, J., and Drury, C. (1992). "Classification of Temporal Errors in CIM Systems: Development of a Framework for Deriving Human-Centered Information Requirements." *International Journal of Computer Integrated Manufacturing*, 5(2), 68-80.
74. Rao, H. A., and Gu, P. (1995) "Entropic measure to determine reconfiguration using integrated system design." *IEEE International Conference on Systems, Man and Cybernetics, 'Intelligent Systems for the 21st Century,* 518-523 vol.1.
75. Ramos, C. (1996) "A holonic approach for task scheduling in manufacturing systems." Proceedings of 1996 IEEE International Conference on Robotics and Automation, 2511-2516 vol.3.
76. Diep, D. Reaidy, J. (2004) "Scheduling agents in a distributed flexible manufacturing system", 2004 IEEE International Symposium on Industrial Electronics, 4-7 May, vol. 1, 739- 744
77. Reaidy, J.; Diep, D.; Massotte, P. (2003) “Management and control of complex production systems: co-opetition through game theory principles and agents based information systems”, Proceedings of IEEE International Conference on Industrial Informatics (INDIN 2003), 21-24 Aug, 217-223.

78. Reaidy, J., Massotte, P., and Diep, D. (2006). "Comparison of negotiation protocols in dynamic agent-based manufacturing systems." International Journal of Production Economics, 99(1-2), 117-130.
79. Rao, H. A., and Gu, P. (1995) "Entropic measure to determine reconfiguration using integrated system design." IEEE International Conference on Systems, Man and Cybernetics, 'Intelligent Systems for the 21st Century'. 518-523 vol.1.
80. Roy, D., Anciaux, D., and Vernadat, F. (2001). "SYROCO: A novel multi-agent shop-floor control system." *Journal of Intelligent Manufacturing*, 12(3), 295.
81. Sabuncuoglu, I., and Bayiz, M. (2000). "Analysis of reactive scheduling problems in a job shop environment." *European Journal of Operational Research*, 126(3), 567-586.
82. Sauter, T. and Massotte, P., (2001), "Enhancement of distributed production systems through softwareagents", *Proceedings. 2001 8th IEEE International Conference on Emerging Technologies and Factory Automation*, 267-272.
83. Shah, S. S., Endsley, E. W., Lucas, M. R., and Tilbury, D. M. (2002) "Reconfigurable logic control using modular FSMs: Design, verification, implementation, and integrated error handling." *Proceedings of the 2002, American Control Conference,* 4153-4158 vol.5.
84. Shen, W. (2002). "Distributed manufacturing scheduling using intelligent agents." *Intelligent Systems, IEEE [see also IEEE Expert]*, 17(1), 88-94.
85. Shen, W., and Norrie, D. (1999). "Agent-based systems for intelligent manufacturing: A state-of-the-art Survey." *Knowledge and information systems, an International Journal*, 1(2), pp 129-156.
86. Simon, H. (1982). "The Architecture of Complexity." The Sciences of the Artificial, The MIT Press, Cambridge, MA., 193-230.
87. Sridharan, S. V., and Zhou, Z. (1996). "Dynamic non-preemptive single machine scheduling." Computers and Operations Research, 23(12), 1183-1190.
88. Srivastava, B. (1998) "An Effective Heuristic for Minimising Makespan on Unrelated Parallel Machines", the Journal of the Operational Research Society, 49(8), 886-894

89. Storoshchuk, O., Wang, S., and Shin, K. G. (2001) "Modelling manufacturing control software." *Proceedings of IEEE International Conference on Robotics and Automation, ICRA 2001*, 4072-4077 vol.4.
90. Struebing, L. (1996). "Reconfigurable manufacturing systems to help U.S. firms compete." *Quality Progress*, 29(8), 14.
91. Su, J., and Chen, F. F. (2003) "Stand-Alone Device Control for Reconfigurable Manufacturing Systems." *international FAIM conference 2003*, Tampa, FL, U.S.A, 221~226.
92. Su, J., and Chen, F. F. (2005) "A Component and Service Based Enterprise Application Integration (EAI) Framework." *The 15th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2005)*, University of Deusto, Bilbao, Spain.
93. Torre. de Q. (2004) "Software Component Evaluation: an overview", Conferência da Associação Portuguesa de Sistemas de Informação (CAPSI 2004), Lisboa, Portugal.
94. Wang, C., Shen, W., and Ghenniwa, H. (2003) "An adaptive negotiation framework for agent based dynamic manufacturing scheduling." *IEEE International Conference on Systems, Man and Cybernetics,* 1211-1216 vol.2.
95. Wang, S., and Shin, K. G. (2002). "Constructing reconfigurable software for machine control systems." *Robotics and Automation, IEEE Transactions on*, 18(4), 475-486.
96. West, J. (2003). "How open is open enough? Melding proprietary and open source platform strategies." *Research Policy*, 32(7), 1259.
97. Wills, L.; Kannan, S.; Sander, S.; Guler, M.; Heck, B.; Prasad, J.V.R.; Schrage, D.; Vachtsevanos, G. (2001). "An Open Platform for Reconfigurable Control." *IEEE Control Systems Magazine*, 49~64.
98. Wooldridge, M., and Jennings, N. R. (1998) "Pitfalls of Agent-Oriented Development." *Proceedings of the 2nd International Conference on Autonomous Agents* Minneapolis, 385-391.
99. Wooldridge, M., Jennings, N. R. (1995). "Intelligent agents: theory and practice." *The Knowledge Engineering Review*, 10(2), 115-152.

100. Wyns, J. (1999). "Reference Architecture for Holonic Manufacturing Systems: the key to support evolution and reconfiguration."

101. Xia, F., Wang, Z., and Sun, Y. (2004) "Towards component-based control system engineering with IEC61499." *Fifth World Congress on Intelligent Control and Automation. WCICA 2004.* , 2711-2715 Vol.3.

102. Yook, J., Tilbury, D., Chervela, K., and Soparkar, N. (1998) "Decentralized, modular real-time control for machining applications." *Proceedings of the 1998 American Control Conference,* 844-849 vol.2.

103. Zhang, J., Chan, F. T. S., Li, P., Lau, H. C. W., Ip, R. W. L., and Samaranayak, P. (2002). "Investigation of the reconfigurable control system for an agile manufacturing cell." *International Journal of Production Research*, 40(15), 3709-3724.

104. Zhang, X., Balasubramanian, S., Brennan, R. W., and Norrie, D. H. (2000). "Design and implementation of a real-time holonic control system for manufacturing." *Information Sciences*, 127(1-2), 23-44.

105. Zhou, B., Wang, L., and Norrie, D. H. (1999) "Design of Distributed Real-time Control Agents for Intelligent Manufacturing Systems." *Proceeding of the 2nd International Workshop on Intelligent Manufacturing Systems*, 237-244.

106. Zhu, L.X., and Wang, F. Y. (2003) "Component-based constructing approach for application specific embedded operating systems." *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems* 1338-1343.

Appendix

Appendix A. Source codes of robot device control

```
Sub Form_Load ()
  inifile$ = app.Path + "\walli.ini"
End Sub

Sub Form_Activate ()
    Dim turbo_baseseg As Integer
    Dim io_baseseg As Integer
    Dim i As Integer



        i = loadBop()
        MsgBox "Returned " + Format$(i)

        io_baseseg = &H220

            i = isrj45(io_baseseg, 100)
            If i < 90 Then io_baseseg = 0
            chain turbo_baseseg, io_baseseg




        setdevtype 2, 14, 16
        setdevtype 3, 14, 17

        SetOnLine 2
        SetOnLine 3
End Sub

Sub keydown (Index As Integer)
```

```
        Select Case Index
        Case 0 To 9
           send_rj45_mesg 2, mCNTRL, 10 + Index * 2

        Case 11 'slide +
           send_rj45_mesg 2, mCNTRL, 56

             '
        Case 12 'slide-
           send_rj45_mesg 2, mCNTRL, 58

        End Select
    End Sub

    Sub keyup (Index As Integer)
      send_rj45_mesg 2, mCNTRL, 46
    End Sub
```

“Walli.GLB”

```
Declare Function GetPrivateProfileINT Lib "Kernel" (ByVal lpApplicationName As
        String, ByVal lpKeyName As String, ByVal nDefault As Integer, ByVal
        lpFilename As String) As Integer
Declare Function isrj45 Lib "cyber.dll" (ByVal baseseg As Integer, ByVal tries As
        Integer) As Integer
    Declare Function clearrj45 Lib "cyber.dll" ()
    Declare Sub chain Lib "cyber.dll" (ByVal baseseg As Integer, ByVal rj45seg As
Integer)
    Declare Sub unchain Lib "cyber.dll" ()
    Declare Sub SetOnLine Lib "cyber.dll" (ByVal dev As Integer)
    Declare Sub SetOffLine Lib "cyber.dll" (ByVal dev As Integer)
```

```
Declare Sub setrs232op Lib "cyber.dll" (ByVal dev As Integer, ByVal Op As Integer,
        ByVal dialog As Integer)

    Declare Function loadBop Lib "cyber.dll" () As Integer
    Declare Function unloadBop Lib "cyber.dll" () As Integer

Declare Sub setdevtype Lib "cyber.dll" (ByVal dev As Integer, ByVal devtype As Integer,
        ByVal devaddress As Integer)

Declare Sub send_rj45_mesg Lib "cyber.dll" (ByVal devno As Integer, ByVal mesgaddr
        As Integer, ByVal mesg As Integer)

    Global Const mCNTRL = &H50
```

Appendix B. Pseudo Codes for TS used in “look ahead” time windows

```
// Starting point: Generate an initial feasible schedule
Assign the machine with shortest processing time to each job
Calculate makespan and record it as Cmax
Initialize tabu list
Do {
 //search in reassignment neighborhood
 Sort the machine indexes in a non-increasing order
 Let M1 as the machine with maximum makespan
 Do {
        If the move is not in Tabu list then
                Assign the job to the other machine
                If new makespan < Cmax then
                        Record the move into tabu list and stop
                Else record the minimum makespan as tmpCmax
        Else stop
 } while a job assigned to M1 can be reassigned to another machine
 //search in interchange neighborhood
 Let M1 as the machine with maximum makespan
 Let M2 as the machine with minimum makespan
Do {
        If the move is not in Tabu list then
                Exchange the assignment of machines for the two jobs
                If new makespan < Cmax then
                        Record the move into tabu list and stop
                Else record the minimum makespan as tmpCmax
        Else set M2 as the machine with next small makespan
} while there are two jobs assigned to M1 and M2 can be exchanged
 Set Cmax ← tmpCmax
 Set initial schedule ← schedule with tmpCmax
```

Record the move that change Cmax to tmpCmax into tabu list

} while (stop criteria is not met)

Appendix C. p-values of Experiment Results

Tukey's multiple comparison method is implemented in SAS software to compare the means of all scheduling results from chapter five. A pairwise comparison examines the difference between two treatment means. All pairwise comparisons are all possible combinations of two treatment means. Tukey's multiple comparison adjustment is based on conducting all pairwise comparisons and guarantees the Type 1 experimental error rate is equal to alpha for this situation.

(a). $\lambda = 4$, no machine breakdowns

	SPT	FIFO	MAS	MASTW(p)	MASTW(2p)	MASTW(5p)
SPT	-	0.1732	0.0865	0.0492	0.0031	< 0.0001
FIFO	0.1732	-	0.0621	0.0613	0.0364	0.0008
MAS	0.0865	0.0621	-	0.0471	0.0215	0.0123
MASTW(p)	0.0492	0.0613	0.0471	-	0.1102	0.0652
MASTW(2p)	0.0031	0.0364	0.0215	0.1102	-	0.0612
MASTW(5p)	< 0.0001	0.0008	0.0123	0.0652	0.0612	-

(b). $\lambda = 6$, no machine breakdowns

	SPT	FIFO	MAS	MASTW(p)	MASTW(2p)	MASTW(5p)
SPT	-	0.1845	0.1020	0.0773	0.0140	0.0007
FIFO	0.1845	-	0.0673	0.0789	0.0468	0.0035
MAS	0.1020	0.0673	-	0.0557	0.0221	0.0124
MASTW(p)	0.0773	0.0789	0.0557	-	0.1283	0.0657
MASTW(2p)	0.0140	0.0468	0.0221	0.1283	-	0.1711
MASTW(5p)	0.0007	0.0035	0.0124	0.0657	0.1711	-

(c). $\lambda = 10$, no machine breakdowns

	SPT	FIFO	MAS	MASTW(p)	MASTW(2p)	MASTW(5p)
SPT	-	0.1879	0.1209	0.0782	0.0268	0.0112
FIFO	0.1879	-	0.1222	0.0702	0.0709	0.0134
MAS	0.1209	0.1222	-	0.1018	0.0414	0.0630
MASTW(p)	0.0782	0.0702	0.1018	-	0.1186	0.1099
MASTW(2p)	0.0268	0.0709	0.0414	0.1186	-	0.2111
MASTW(5p)	0.0112	0.0134	0.0630	0.1099	0.2111	-

(d). $\lambda = 4$, with machine breakdown

	SPT	FIFO	MAS	MASTW(p)	MASTW(2p)	MASTW(5p)
SPT	-	0.1673	0.0783	0.0243	0.0010	< 0.0001
FIFO	0.1673	-	0.0557	0.0543	0.0311	0.0010
MAS	0.0783	0.0557	-	0.0462	0.0206	0.0106
MASTW(p)	0.0243	0.0543	0.0462	-	0.1302	0.0712
MASTW(2p)	0.0010	0.0311	0.0206	0.1302	-	0.0312
MASTW(5p)	< 0.0001	0.0010	0.0106	0.0712	0.0312	-

(e). $\lambda = 6$, with machine breakdowns

	SPT	FIFO	MAS	MASTW(p)	MASTW(2p)	MASTW(5p)
SPT	-	0.1721	0.0838	0.0284	0.0017	0.0012
FIFO	0.1721	-	0.0605	0.1586	0.0313	0.0056
MAS	0.0838	0.0605	-	0.0553	0.0344	0.0237
MASTW(p)	0.0284	0.1586	0.0553	-	0.1104	0.1602
MASTW(2p)	0.0017	0.0313	0.0344	0.1104	-	0.0732
MASTW(5p)	0.0012	0.0056	0.0237	0.1602	0.0732	-

(f). $\lambda = 10$, with machine breakdowns

	SPT	FIFO	MAS	MASTW(p)	MASTW(2p)	MASTW(5p)
SPT	-	0.2203	0.1348	0.0770	0.0483	0.0378
FIFO	0.2203	-	0.1102	0.0894	0.0452	0.0367
MAS	0.1348	0.1102	-	0.1235	0.0588	0.0427
MASTW(p)	0.0770	0.0894	0.1235	-	0.1504	0.0734
MASTW(2p)	0.0483	0.0452	0.0588	0.1504	-	0.0551
MASTW(5p)	0.0378	0.0367	0.0427	0.0734	0.0551	-

Appendix D. Introduction to papers

"Proceedings of the 2003 international FAIM conference, Tampa, FL, U.S.A, 221~226"

Stand-Alone Device Control for Reconfigurable Manufacturing Systems

Jiancheng Su and F. Frank Chen

ABSTRACT

Almost all industrial flexible manufacturing systems (FMS) are equipped with vendor-designed and vendor-supported supervisory control software packages which ensure the overall system integration needed in highly automated environment. However, total reliance on such a control system may greatly curtail the usability of machinery and material handling equipment in an FMS since system components cannot be individually controlled via using the vendor cell control systems for stand-alone applications. In this paper, we present a method of developing device controls in FMS environment. First, general approaches to the coding of such programs are reviewed. The general procedure to analyze the typical supervisory cell control software of an FMS and the specific process of coding device controls are then described. Benefit of this proposed work and its contribution to the realization and support of next-generation reconfigurable manufacturing systems will also be discussed. An example case study using a laboratory FMS will be provided as an illustration.

“2004 IERC Conference, Houston, TX, U.S.A”

Control Software for Reconfigurable Manufacturing Systems: State-of-the-art Review and Future Trends

Jiancheng Su and F. Frank Chen

ABSTRACT

Control software plays an important role in modern manufacturing systems as it greatly affects cost and implementation time. Recently, a new manufacturing paradigm ¨C reconfigurable manufacturing system (RMS), which intends to provide manufacturers with manufacturing systems that have “exactly the functionality and capacity needed at the exact timing” is gaining popularity. RMS has the potential to meet the challenges of the modern manufacturing environment characterized with mass product customization and unpredictable market demands. Reconfigurable control software is a critical component of RMS, which enables RMS to react to product and demand changes rapidly and cost-effectively. A comprehensive review of research work on control at system and machine levels of reconfigurable manufacturing systems is provided in this paper. Future trends and further research efforts in control software needed to support RMS are also presented.

Keywords: control software, reconfigurable manufacturing systems, system-level, machine-level

“Proceedings of the 2004 international FAIM conference, Toronto, CANADA, 113~119”

Agent-based modular control architecture for reconfigurable manufacturing systems

Jiancheng Su and F. Frank Chen

ABSTRACT

A Reconfigurable Manufacturing System (RMS) is designed to cope with rapid changes cost-effectively in modern manufacturing environment. Control software for RMS should have a modular architecture and be open such that it can be easily upgraded once a system reconfiguration occurs in response to market changes. A new agent-based modular architecture for RMS control software is presented in this paper. Within this architecture, intelligent control modules are constructed as building blocks for RMS control software. Component-based software technology is selected to build software modules because of its inherent modularity. Agents in control modules act as exception handler, which are activated when unexpected events occur. Key features of this control architecture include: scalability, reconfigurability, agility, reliability, and fault-tolerance.

"3rd International CIRP Sponsored Conference on Reconfigurable Manufacturing Systems, Ann Arbor, MI, 2005"

Component-based System-level Control Architecture for Reconfigurable Manufacturing Systems

Jiancheng Su and F. Frank Chen

ABSTRACT

The hardware/software of a Reconfigurable Manufacturing System (RMS) need to be designed in a modular form. The system-level control with open architecture for RMS can hardly be found in existing literatures. New emerging software paradigm called Component-based Software Development (CBSD) can potentially be used to reduce software development costs and to rapidly assemble/configure systems. In this paper, a component-based, system-level control architecture is proposed for RMS. Formal component specification, which precisely describes and models software components, is critical in CBSD. An UML-based generic description is presented to comprehensively define software architecture and components.

"Proceedings of the 2005 international FAIM conference, Bilbao, Spain"

A Component and Service Based Enterprise Application Integration (EAI) Framework

Jiancheng Su, F. Frank Chen, and Dong-Won Kim

ABSTRACT

Enterprise application integration (EAI) refers to plans, methods, and tools of integrating applications and data sources in an enterprise. Traditional EAI solutions, done with middleware, application adapters, and numerous customer codes, are usually expensive and time-consuming. Two emerging information technologies, component-based development (CBD) and web services, can potentially address the challenges. A component and service based EAI framework is presented in this paper. The framework is a non-proprietary design with well-defined interfaces. This paper also improves a specification modelling method of enterprise software components. In this paper, the first section gives a brief introduction of background information. Then, state-of-the-art in EAI, concepts of CBD and web services are reviewed. In the third section, a new component and service based EAI framework is presented. Main contents this section include, 1) an overview of the framework; 2) An improved UML-based description of component specifications; and 3) major benefits and challenges of the proposed framework are listed. The last section of the paper is a summary of this paper.

"2006 IERC Conference, Orlando, FL, U.S.A"

Resolution of local interest conflicts in agent-based scheduling using the game theory approach

Jiancheng Su and F. Frank Chen

ABSTRACT

Recently, agent-based technology has been widely reported in research of production planning, scheduling and control problems. Agents are self-motivated and try to maximize their own benefits. When they compete for a set of resources, it is usually not adequate for an agent to try to optimize its own local preferences. The widely used contract net protocol by itself does not avoid such conflicts. Game theory can be used to study mathematical models of conflict and cooperation among interacting agents. In this paper, we use the game theory approach to analyze and solve the problems potentially caused by agents local interest conflicts in agent-based scheduling.

Keywords: Agent-based scheduling, local interest conflict, game theory

Vita

Jiancheng Su started his Ph.D. studies in the Grado Department of Industrial and Systems Engineering at Virginia Tech, Manufacturing Systems Engineering Option in August, 2001. He worked as a Graduate Research Assistant under the direction of Dr. F. Frank Chen. His main research interests include intelligent manufacturing control, component-based software technology, agent technology, E-manufacturing, enterprise application integration, artificial intelligence. Prior to coming to Virginia Tech, he received a B.S. degree in automation from the Qingdao University of Science & Technology, China, and a M.S. degree in automation from the Tsinghua University, China.