INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Université d'Ottawa · University of Ottawa

Agent-Based Management of a Task-Level Multi-Robot Assembly Cell

by

Jagdeep S. Basran, B.Eng., M.A.Sc.

A dissertation submitted to the

School of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering

School of Information Technology and Engineering (Electrical & Computer Engineering)

Faculty of Engineering
University of Ottawa

August 1997

Abstract

This dissertation presents a flexible coherent framework for managing the on-line, high-level operation of an assembly cell that has multiple robotic manipulators. This framework, which uses the concept of an agent as understood by the research subfield of multi-agent systems, supports the task-level scheduling and programming of assembly operations. The execution of an assembly operation was decomposed into the two distinct phases of part assembling and part fetching. Part assembling is concerned with how a robotic manipulator manoeuvres a part into its proper place within the overall assembly of a product. Part fetching has to do with suitably delivering a part to a robotic manipulator for part assembling. In this framework, the operation of a multi-robot assembly cell was distributed among four different types of agents: agent(s) responsible for part assembling, agent(s) responsible for part fetching, an agent to properly schedule the assembly operations, and an agent to control access to the shared physical space of the cell. Each of these agents was implemented in software using the object-oriented programming language C++. The physical-level of this framework was evaluated by physically implementing a proof of concept, assembling agent. The system-level of this framework was evaluated by integrating the software agents into a discrete event simulation and then simulating a tractable and prototypical example for different operating circumstances.

Acknowledgements

I thank NSERC, OGS, and TRIO for the financial support that was provided to me during the course of my doctoral research.

I acknowledge the Visual Information Technology group for the access to excellent library resources and the use of their robotic equipment while I was a guest worker with them. This group belongs to the Institute for Information Technology at the National Research Council of Canada in Ottawa. I thank Shadia Elgazzar and Jacques Domey of this group for supporting me during this time.

I thank Larry Korba for the many coffees and Friday lunches that we shared together. I thank Doug Taylor for his technical support with the Puma 500 robot and for the Friday lunches.

Several colleagues, who had either completed their doctoral degree or were doctoral candidates, motivated and encouraged me along the way. I thank Colin Archibald, Pierre Boulanger, Graham Eatherly, Guy Godin, Ramiro Liscano, Ann Parent, Gerhard Roth, and Elizabeth Stuck.

In particular, I thank Colin Archibald for his comments on my proposal and for explaining the details of his implementation before he left NRC. I thank Ann Parent for encouraging me to persist with my doctorate during periods of frustration. I am grateful to Elizabeth Stuck for her friendship and for fine tuning my writing skills. I thank Gerhard Roth for his unwavering support for hiring me for a one year contract at NRC and for the many exciting chess games that we have played together.

I thank Dorina Petriu for providing excellent guidance regarding discrete event simulations and for how to present the object-oriented implementation of my software. I thank my supervisor, Emil Petriu, immensely for supporting me both financially and personally during my doctoral program.

I am grateful to all five members of my family who encouraged and supported me during my graduate studies. I am indebted to my Mom and Dad for all that they have done for me. I thank Cindy and Paul for their emotional support and for ensuring that I always had good footwear. I thank Sharan for always asking about my thesis progress and for her assistance in completing my thesis during the last week.

To Diane, a deep-felt thank you for your love and your gift of life.

Dedicated to my parents

Table of Contents

List of Figures

List of Tables

List of Acronyms

AGV	Automated Guided Vehicle
AI	Artificial Intelligence
AM	Agile Manufacturing
ASA	ASsembling Agent
ASAs	ASsembling Agents
CE	Concurrent Engineering
CIM	Computer Integrated Manufacturing
CSM	Communicating State Machine
DAI	Distributed Artificial Intelligence
DES	Discrete Event Simulation
DPS	Distributed Problem Solving
FABRIC	Flexible Agent-Based RobotIC assembly Cell
FMS	Flexible Manufacturing System
FMSs	Flexible Manufacturing Systems
KQML	Knowledge Query and Manipulation Language
MAS	Multi-Agent System
MASs	Multi-Agent Systems
MRAC	Multi-Robot Assembly Cell
MRACs	Multi-Robot Assembly Cells
PPA	Part Presentation Agent
PPAs	Part Presentation Agents
SE	Software Engineering
SSA	Shared Space Agent
TSA	Task Scheduling Agent
UML	Unified Modeling Language

Chapter 1

Introduction

This introductory chapter includes an overview of the research problem, motivations for studying this problem, properties of an ideal solution, a synopsis of the proposed solution, contributions of this dissertation, and an outline of the remaining chapters of this dissertation.

1.1 Overview of the Research Problem

The research of this dissertation has an interdisciplinary nature, since it takes place at the intersection of the field of manufacturing, the field of artificial intelligence (AI), and the field of robotics. More precisely, this intersection occurs between manufacturing cells, the AI research subfield of multi-agent systems, and robotic assembly respectively.

The specific system under study in this dissertation is a multi-robot assembly cell (MRAC). The multiple robots of this cell manipulate parts into their proper place within the overall assembly of a product. Given that a MRAC is a complex, spatially and temporally distributed, physical part processing system, the specific research problem addressed in this dissertation is how to effectively manage the high-level operation of such a system.

The question now becomes what does it mean to manage a MRAC whose on-line operation is decomposed into the loosely coupled coordination of its components. In this case, management

of system operation means addressing the high-level aspects of control, communication, organization, scheduling, and coordination of these components. This dissertation provides and investigates a solution to this specific problem.

1.2 Motivations and Solution Discussion

A key motivation for studying the high-level management of a MRAC is to operate it at a higher level of abstraction. The reasoning behind this motivation is that there is a strong link between the level of automation of a manufacturing system and the level of abstraction at which instructions are input to such a system. Another motivating factor, which relates to this level of abstraction, is to provide techniques for the task-level scheduling and programming of assembly operations carried out by a MRAC.

Another strong motivation for studying this problem is that although researchers have studied many different issues regarding robotic assembly cells, the management of the on-line, high-level operation of multi-robot assembly cells (MRACs) remains a relatively unexplored problem. Another obvious motivation for studying MRACs versus single robot assembly cells is the potential of reducing the cycle time or throughput of the cell.

Ideally, the solution to manage a MRAC would have as many of the properties of being modular, extensible, reliable, efficient, robust, maintainable, and adaptable, as possible. Clearly, these properties are important considerations in deciding upon a solution.

The solution proposed and investigated in this dissertation makes a modest step towards this ideal solution. Specifically, our solution is a generic, agent-based framework for managing the on-line, high-level operation of a MRAC. This framework uses and applies concepts from the research subfield of multi-agents systems to the operation of a MRAC. In particular, the operation of a MRAC was distributed among four different types of agents: a task scheduling agent, m assembling agent(s), n part presentation agent(s), and a shared space agent.

The input to a MRAC managed with this framework is a directed graph of assembly

operations that respect the precedence relations of an assembly. The assembly operations of this graph are specified as task-level instructions. The physical execution of each assembly operation was decomposed in this cell into the two distinct phases of part fetching and part assembling. These two phases correspond with the responsibilities of the part presentation agent and the assembling agent respectively.

1.3 Contributions

This section summarizes the primary contributions of this dissertation that arose from investigating the problem of managing the operation of a task-level MRAC. These contributions are:

- an original agent-based framework for managing the on-line, high-level operation of a MRAC; its operation was distributed among four types of agents based, in part, upon the decomposition of an assembly operation into the two distinct phases of part fetching and part assembling;

- unique application of the contract net technique for coordinating a multi-agent system to a MRAC; specific contents of messages exchanged during contract net negotiation defined for the tasks of assembly operations;

- novel approach to the problem of task planning for an assembly robot with implications for task-level robot programming; this approach was encapsulated within the internal structure of an assembling agent;

- implementation of a proof of concept, assembling agent; this physical agent was implemented using a six degree of freedom, Puma 500, industrial manipulator with a force/torque sensor and a range sensor;

- development of a prototyping tool to simulate the discrete event behaviour of MRACs managed with our framework; this tool allows the comparison of alternative configurations of MRACs; and

- development of one of the few frame-based knowledge bases in the C++ programming language.

1.4 Outline

Chapter 2 begins by giving an overview of the fields of manufacturing, artificial intelligence, and robotics from the unique perspective of this dissertation. This chapter continues by surveying research that overlaps among these fields from this perspective.

Chapter 3 describes the agent-based framework for managing the on-line, high-level operation of a MRAC. This chapter discusses fundamental issues regarding this framework including a reiteration of the motivations for its proposal. This chapter also presents details on the overall operation of this framework, internal aspects of the four types of agents, messages exchanged between the agents, and the relationship of this framework with existing research.

Chapter 4 describes how the agents of this framework were implemented in software. In particular, this chapter details the implementation of these agents using the object-oriented programming language C++.

Chapter 5 details how an assembling agent was physically implemented. This chapter describes the physical setup and preexisting robotic programming environment that our implementation was based upon. This chapter also discusses the demonstration of a basic, peg in hole, assembly operation, and how our implementation relates to existing research.

Chapter 6 describes the integration of the software agents into a discrete event simulation, describes a tractable and prototypical example that was simulated, and presents the results of simulating this example in order to evaluate the system-level operation of our framework.

Chapter 7 gives conclusions and presents future research directions for this dissertation.

Chapter 2

Interdisciplinary Survey of Related Research

The research of this thesis takes place at the intersection of three fields of study: manufacturing, artificial intelligence, and robotics. This chapter gives an overview of these three fields and then surveys research that overlaps among these fields. That is, the interdisciplinary survey progresses from general to specific and aims to give an appreciation of the larger context within which our research occurs and to describe research related to our own.

Since each of the aforementioned fields is large, the most relevant aspect of each field is indicated in order to demarcate the unique perspective of this survey. With respect to the field of manufacturing, the level of manufacturing cells is most significant with an emphasis placed on robotic assembly cells. With respect to the field of artificial intelligence, the subfield of distributed artificial intelligence / multi-agent systems is most relevant. With respect to the field of robotics, more attention is paid to industrial robots that do assembly by manipulating parts.

2.1 The Field of Manufacturing

The field of manufacturing is one of the key application areas of the technologies of automation [Goetz, 1989]. This field has advanced through the establishment of different strategies to improve the overall operation of manufacturing systems. Each of these strategies, in turn, was an attempt to rectify deficiencies or improve upon the shortcomings existing in the previous infrastructure. This section briefly discusses each strategy of significance.

2.1.1 Flexible Manufacturing Systems

Historically, the first automation of manufacturing facilities relied on the introduction of automatic machinery to continually repeat a specific operation. Although this strategy, called fixed or hard automation [Goetz, 1989], was effective for mass production of products, it lacked flexibility and adaptability. The concept of a flexible manufacturing system (FMS) [Ranky, 1983; Draper Laboratory, 1984; Pao and Jelinek, 1988] was introduced and developed to improve upon the existing rigid systems. Flexibility in manufacturing allows rapid responses to be made to changing demands and circumstances. Examples include optimizing and adapting production schedules as a function of resource availability, producing variants of similar products, responding to changes in customer demands, coping with short product life cycles, and responding to market trends. For a more recent examination of the literature on FMSs and suggestions for future research, one can refer to [Gunasekaran et al., 1995].

The operation of a FMS is usually broken down into at least three distinct levels. We distinguish between the following four levels:

- factory or enterprise level
- shop or line level
- cell level
- component level

The factory or enterprise level involves the overall operation of the manufacturing system. The shop or line level is concerned with a major aspect of the factory, such as the efficient manufacture of a product family. This level is usually implemented as a collection of cells. A cell is a basic unit of production in a manufacturing system. The cell level is concerned with the effective operation of its collection of components. Depending upon the cell, components can be automated guided vehicles, buffers, CNC (computer numerical control) machines, industrial robots, lathes, pallets, parts feeders, sensor systems, tool changers, various workstation(s), etc.

Since the problem of scheduling is addressed in our robotic assembly cell, we mention two papers on scheduling of a FMS. Akella and Krogh [1987] used an analytical approach to examine

the scheduling problem for a hierarchical, multi-cell flexible assembly system. Graves [1987] used a heuristic approach to optimize the schedule for flexible assembly systems.

2.1.2 Computer Integrated Manufacturing

The strategy of computer integrated manufacturing (CIM) [Ranky, 1986a; Spur and Specht, 1987] followed shortly after the one of FMSs. These two strategies are clearly linked together. The emphasis of FMSs is to make the process of physical production as flexible as possible; whereas, the emphasis of CIM is to integrate all of the data, information, and knowledge surrounding a manufacturing system. Computer Aided Design (CAD) and Computer Aided Manufacture (CAM) are two technologies that support CIM. The primary purpose of this integration is to make it easier for technical staff to design and manufacture new products, for management to observe and analyze the complete operation of the enterprise, and for marketing to promote the products. Researchers continue to refine the details of the concept of CIM. A special issue on CIM that discussed current issues of CIM was published in [Desrochers and Silva, 1994]. In addition, Camarinha-Matos et al. [1995] presented a thorough taxonomy of CIM activities.

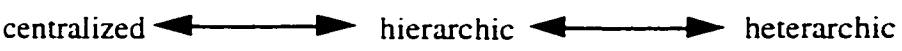


Figure 2.1: Types of control architectures.

The question of what type of architecture to use to integrate manufacturing activities is discussed here since this question also applies to our robotic assembly cell. [Dilts et al., 1991] gave an excellent assessment of the control architectures used for automated manufacturing systems. Figure 2.1 displays the type of control architectures reviewed in [Dilts et al., 1991]. These architectures historically evolved from left to right. In a centralized architecture, the control of the entire system is directly handled by a single global coordinator. In a hierarchic architecture, the control occurs at multiple levels with master/slave relationships between adjacent levels. In a heterarchic architecture, the control is distributed among locally autonomous entities that communicate on a peer to peer level. Refer to the original source [Dilts et al., 1991] for a more

detailed discussion of these types of architectures. More recently, Williams et al. [1994] presented complete descriptions of architectures for CIM.

2.1.3 Concurrent Engineering

According to [Nevins et al., 1989], concurrent design refers to the coordinated design of products and processes and using this approach could result in more efficient and effective manufacturing. The key issue is that the early design decisions critically affect the life cycle of the product and therefore, all of the departments of a company should work from the beginning on the design of new products [Nevins et al., 1989]. Concurrent engineering (CE) focuses on the parallel interaction of team members during various life cycles of the product [Ranky, 1994]. Two areas of study that are related to this strategy are Design for Manufacture (DFM) and Design for Assembly (DFA). From a CE point of view, CIM provides the infrastructure for a CE solution [Ranky, 1994].

2.1.4 Agile Manufacturing

The most recent strategy for improving manufacturing systems promotes the concept of agile manufacturing (AM). This strategy came out of a United States based initiative called the agility forum that was organized in 1991. For discussions about how AM relates to and differs from the earlier strategies, one can refer to [Burgess, 1994; Kidd, 1994; Noaker, 1994; Booth 1996]. This strategy was partly a response to the strategy of just-in-time lean production developed by the Japanese company Toyota [Booth, 1996] and it was also viewed as a replacement for the outdated mass production model [Burgess, 1994]. The characteristics of an AM system include the ability to quickly produce low cost, low volume, high quality, customized products [Burgess, 1994].

We indicate references that discuss three specific aspects of AM. Pant et al. [1994] discussed the problem of information integration from the perspective of AM but also relate it to the approaches of CIM and CE. The prospect of achieving AM in the automobile industry was discussed in [Tracy et al., 1994a, 1994b]. Wang et al. [1996] discussed an architecture for AM

and its interface to CIM.

2.2 The Field of Artificial Intelligence

The field of artificial intelligence (AI) is concerned with the implementation of certain aspects of human intelligence on a machine and it is a broad field with many research directions [Barr et al., 1982]. A complete discussion of this field is obviously beyond the scope of this section; however, we must be aware of it since the subfield of distributed artificial intelligence (DAI) / multi-agent systems (MASs) has its roots in this field. This section begins by introducing this subfield to the reader and giving a synopsis of the major topics being investigated by researchers. We also more closely examine the concept of an agent.

2.2.1 Distributed Artificial Intelligence / Multi-Agent Systems

DAI basically studies concurrency in all of its forms within the field of artificial intelligence (AI). Early developments of this subfield were published in a series of workshops that began in 1980 [Sridharan, 1987]. Bond and Gasser [1988] compiled a collection of seminal DAI papers and they also partitioned the study of this subfield into two primary branches, distributed problem solving (DPS) and multi-agent systems (MASs). DPS examines how the solving of a particular problem can be divided among cooperating modules, while MASs concerns itself with coordinating the behavior of a collection of intelligent agents [Bond and Gasser, 1988]. More general information on DPS is available in [Chandrasekaran, 1981; Decker, 1987; Durfee et al., 1989]. From this point, the research on DAI continued to evolve and many researchers have published on this theme including [Hern, 1988; Demazeau and Müller, 1989; Gasser and Huhns, 1989; Durfee, 1991; Chaib-Draa et al., 1992; Gasser 1992]. Moreover in 1992, Rosenschein presented a paper on how to best teach the body of knowledge known as DAI.

At the same time, research in the branch of MASs burgeoned so much that this branch now subsumes DAI instead of vice versa [Lesser, 1995]. That is, the terms, DAI and MASs, are now essentially interchangeable. Durfee and Rosenschein [1994] gave a good discussion of this

transformation in terminology while it was happening. Grant [1992] gave a good review of the techniques available to a MAS researcher. An excellent survey of both the theory and practice of MASs are given in [Wooldridge and Jennings, 1995]. These same authors also examined applications of agent technology [Jennings and Wooldridge, 1995]. A more extensive bibliography of MASs was compiled in [Wooldridge et al., 1995]. Currently, one of the most updated sources for information on these agents is on the web [AgentWeb, 1997].

A distinction is made between the micro and macro aspects of MAS research. Micro aspects have to do with all aspects of individual agents, such as agent theories, agent architectures, and agent languages. Macro aspects have to do with the behavior of groups of agents.

Figure 2.2 gives a basic breakdown of the major, micro and macro topics of MASs based upon our readings. An agent theory or model of self has to do with how do researchers conceptualize agents and how do agents conceptualize themselves [Wooldridge and Jennings, 1995]. The study of agent control architectures has to do with how to best implement agent theories in hardware and software [Wooldridge and Jennings, 1995]. Agent languages address the question of how to effectively program agents [Wooldridge and Jennings, 1995]. The next three micro topics are usually incorporated into the internal architecture of an agent [Ferguson, 1992; Müller, 1996]. We list them separately to emphasize their importance. Typically, an agent has to be able to construct a model of its environment. The quality of this model is limited by the perceptual capabilities of the agent. The topic of local resource management has to do with an agent properly managing its own resources such as energy, computing cycles, etc. An agent also usually has a model of its community or society in order to operate reasonably within its environment. There are strong interconnections between these micro topics listed in this figure. For example, an agent's model of self and its community is affected by the model of its environment that is constructed with its perceptual capabilities.

With respect to the macro aspects, the method of coordination that varies from cooperative to competitive is a key defining aspect of the behavior of groups of agents. Although researchers

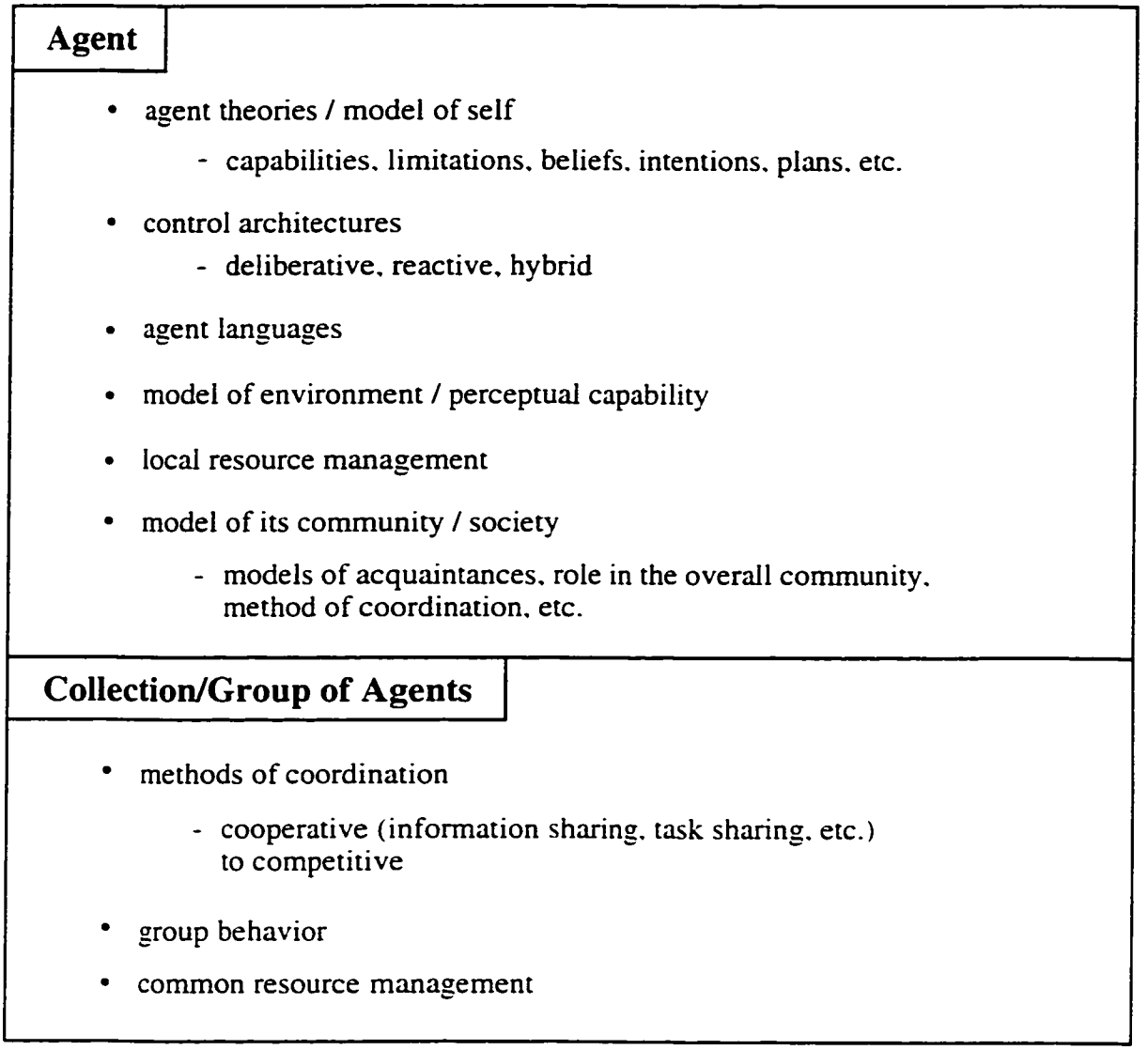


Figure 2.2: Synopsis of micro and macro aspects of multi-agent systems.

have investigated several different methods of cooperation, we listed only the two most fundamental [Smith and Davis, 1981; Jennings, 1994] in the figure. Durfee et al. [1989] provided a much more extensive interpretation of the different categories of cooperation from a DPS perspective. They categorized the methods of cooperative coordination into the following six approaches: negotiation, functionally-accurate cooperation, organizational structuring, multi-agent planning, sophisticated local control, and theoretical frameworks [Durfee et al., 1989]. Jennings [1994] developed a new approach to cooperation for MASs that used the concept of a joint action.

Joint actions were cooperative interactions where participating agents were responsive to the actions of others and mutually supportive. This approach was applied to the industrial problems of cooperative fault diagnosis of a particle accelerator and cooperative electricity transportation management.

We briefly discuss the contract net [Smith, 1979, 1980; Smith and Davis, 1981; Davis and Smith, 1983], a commonly applied method of cooperation. According to [Durfee et al., 1989], the contract net is a negotiation approach to cooperation. From a more general perspective, the contract net is a framework for cooperation in distributed problem solving, a high-level communication protocol for negotiation of tasks between contractees and contractors, a means for dynamic allocation or scheduling of tasks, or some combination of the above.

The blackboard structure [Lesser and Erman, 1980] was and continues to be a common approach for implementing a DAI/MAS system. Given that the two primary models of inter-agent communication are shared memory and message passing, this structure is typically introduced as the most widely used model for shared memory [Decker, 1987; Jennings, 1994]. In this structure, a group of expert systems, knowledge sources, or agents cooperate by retrieving, storing, and modifying the information of a global database or blackboard.

The topic of group behavior examines various facets of the overall operation of the group of agents. The topic of how the common resources are shared or managed amongst the agents is also an important issue to address.

2.2.2 Concept of an Agent

The definition of the term agent varies considerably. Consider the following list of definitions:

> "In this dissertation, an *agent* shall be considered to be any goal-directed computational process capable of robust and flexible interaction with its environment." [Ferguson, 1992, p. 2, emphasis in original]

> "An agent is a self-contained problem-solving entity that exhibits the following

> properties: autonomy, social ability, responsiveness, and proactiveness." [Jennings and Wooldridge, 1995, p. 357]
>
> "An **agent** is anything that can be viewed as **perceiving** its environment through **sensors** and **acting** upon that environment through **effectors**" [Russell and Norvig, 1995, p. 31, emphasis in original]
>
> "Internally, it [the black box model of an agent] is described by a function f which takes perception and received messages as input and generates output in terms of performing actions and sending messages" [Müller, 1996, p.8]

These definitions complement one another and together give a more complete interpretation of this term. A hardware and/or software-based entity must have the properties specified in the second definition in order to pass the weak notion of agency [Jennings and Wooldridge, 1995]. The stronger notion of agency implies also having mentalistic properties, such as knowledge, belief, intention, and obligation [Wooldridge and Jennings, 1995]. The last two definitions are much more compatible with the robotics perspective that this dissertation takes.

To further examine the concept of an agent, we present our own taxonomy for categorizing agents that was inspired in part from a recent survey article on software agents [Nwana, 1996]. This article split the research on agents into two main strands [Nwana, 1996]. The first strand was what we described in section 2.2.1 and this article emphasized that initially this strand concentrated on macro aspects [Bond and Gasser, 1988; Chaib-Draa et al., 1992], while later the focus switched to the micro aspects [Wooldridge and Jennings, 1995]. According to [Nwana, 1996], the second strand of agent research began with the work on actors [Hewitt, 1977; Hewitt and Jong, 1983; Hewitt and Inman, 1991]. An actor is essentially a concurrent object that can communicate via message passing with acquaintance actors. This survey article continued by giving a good review of this second strand of software agents that has flourished since about 1990.

In our categorization, an emphasis is placed upon the application area for which the agent was designed. Figure 2.3 displays our taxonomy of agents. In this taxonomy, the agents that resulted from the DARPA (Defense Advanced Research Projects Agency) knowledge sharing

effort [Neches et al., 1991; Genesereth and Fikes, 1992; Gruber, 1992; Finin et al., 1994; Genesereth and Ketchpel, 1994; Patil et al., 1994; Mayfield et al., 1995; KQML, 1997] are categorized as AI agents. Interface agents typically provide assistance to users with computer tasks [Lashkari et al., 1994]. Another term that researchers use for these software agents is a softbot or software robot [Etzioni and Weld, 1994]. The category of network agents encompasses any agents that make use of a network of computers and/or hardware. This category varies from agents for network management [Reinhardt, 1994] to agents that perform tasks on the internet [Nwana, 1996]. Internet examples include a softbot interface to the internet [Etzioni and Weld, 1994; Etzioni et al., 1995] and an agent that coordinates a visitor of a laboratory using the internet [Kautz et al. 1994]. A mobile network agent can move from computer to computer and properly execute its tasks [TRIO, 1997]. Although the adjective mobile is used to modify network agents and robotic agents, it should be clear that this mobility occurs in very different environments. Network mobility means transporting an executing program from one computer to another while robot mobility means properly navigating in a physical environment. We defer the discussion on robotic agents until section 2.6 of this chapter.

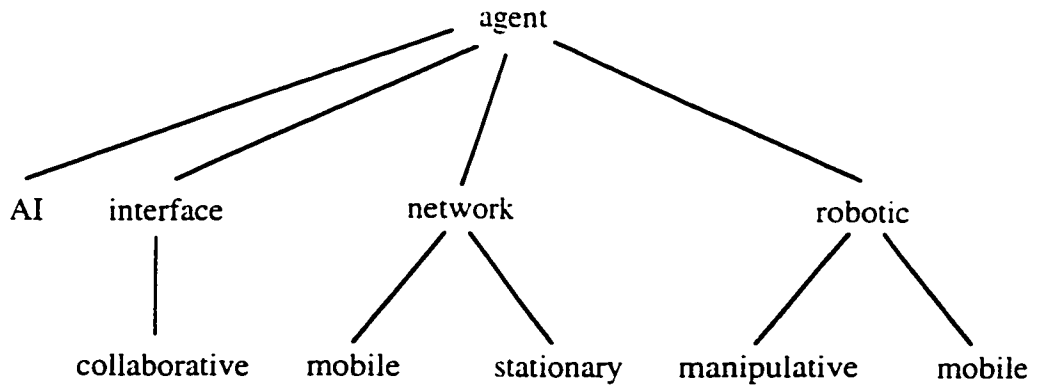


Figure 2.3: Taxonomy of agents.

2.3 The Field of Robotics

The field of modern robotics only recently emerged when the ability to construct an actual physical robot became feasible [Saveriano, 1988]. Although developments in several fields were necessary to make this construction possible, developments in the two key fields of automatic control and of computers were critical. This section gives a perspective on robotics before discussing assembly and robotics.

2.3.1 A Perspective on Robotics

Although the field of robotics has an inherently multidisciplinary nature with many research directions, we maintain that three key factors are adequate to appreciate these directions. We use the term, robotic system, as a general term that encompasses the full range of systems that researchers view as robotic.

The first factor is the type of robotic system that is determined for the most part by its physical construction and operational characteristics, which are usually interdependent. A key operational attribute is the degree of autonomy of a robotic system which can vary along a continuum from teleoperated to autonomous. Most robotic systems can be classified into one of two types based upon their primary area of competence. That is, most robotic systems are either manipulators or navigators. A robotic manipulator can skillfully manipulate parts in its environment to accomplish a variety of tasks, while a robotic navigator can skillfully navigate through an environment to accomplish a variety of goals. Only a handful of researchers have considered robotic systems that can both skillfully manipulate and navigate.

The second factor that distinguishes between robotic research directions is the particular subject matters emphasized by the researcher. Is the focus on sensing, control, programming, planning, navigation, manipulation, learning, intelligence, architectures, or some combination?

The third critical factor that aids in the understanding of robotic research is the environment(s) that the robotic system can operate in. On Earth, environments can be classified

into air, ground, water, and space and each of these classifications has a number of different specializations. Some descriptors that researchers have used in reference to environments are indoor, outdoor, structured, unstructured, static, dynamic, etc. In addition, the breadth of applications for which robotic systems are considered potentially useful has often led to a novel environment that changes the approach and the type of questions that researchers explore. The following list of applications:

- patrolling a prison
- mining precious metals on the ocean bed
- mechanical "guide dogs" for the blind
- domestic servant
- fast food work
- agriculture
- construction

demonstrates this point by giving applications whose environment and associated tasks are significantly different from the majority of applications. For example, for fast food work, researchers have to address the question of the hygiene of the robot. The dependence between the environment of application and the type of robotic system that is constructed and used is important to note.

2.3.2 Assembly and Robotics

With respect to assembly, the type of robotic system is an industrial manipulator, the environment is a manufacturing plant, and all the previously mentioned subject matters have been studied to different degrees by researchers.

This subsection commences by outlining seminal assembly research that was described in [Nevins and Whitney, 1977, 1978, 1980]. These two authors studied both the theoretical and experimental aspects of part mating. Part mating is the study of how parts interact during assembly. Four types of assembly were identified by [Nevins and Whitney, 1980]. The first type of assembly is the manual labour of people and this type has the highest flexibility but the lowest precise repeatability. The second type of assembly identified is done by special purpose machines

and it is designed for mass production with low flexibility and high repeatability. The third type of assembly is done by programmable assembly machines, typically industrial manipulators, and this type has both medium flexibility and repeatability. The last type of assembly is simply any combination of the three previously identified types [Nevins and Whitney, 1980].

Rathmill compiled a collection of papers of the trends in robotic assembly in 1985. Hoska [1988] gave a good overview of assembly robots and according to this article, assembly applications comprised only about 11% of robotic applications in the United States in 1985. For a more recent state of the art review of robotic assembly, one can refer to [Rampersad, 1995a]. The field of assembly and task planning concerns itself with the automatic synthesis of a detailed plan to assemble a product with a robot, given only a high level product description [Gottschlich et al., 1994]. During assembly planning, the order of the mechanical instructions to build a product from parts are determined, while during task planning these instructions are converted into actions executable by a robotic system [Gottschlich et al., 1994].

Having provided a basic introduction to each of the three major fields of relevance to this dissertation, the following four sections examine research that overlaps between these fields. In particular, we examine the research that overlaps between these fields in the following order: manufacturing and AI, manufacturing and robotics, AI and robotics, and lastly all three. Note that the classifying of a reference according to this scheme of overlap was not always straightforward; however, the best effort was made to most appropriately classify each reference. The constraints that were given in the introduction of this chapter will continue to guide the presentation.

2.4 Manufacturing and AI

This section examines the research that overlaps between the field of manufacturing and the field of AI. This section comments on the relationship between manufacturing and traditional AI, introduces the overlap between manufacturing and MASs, and then presents specific applications of MASs to the field of manufacturing. Researchers have applied the field of MASs to the field of

manufacturing at all levels: factory or enterprise, shop or line, cell, and component. This section discusses applications of MASs to manufacturing that do not emphasize robotic assembly.

2.4.1 Manufacturing and Traditional AI

The field of manufacturing has been and continues to be a common application area for the methods and techniques developed by researchers from the field of AI. Before focusing on applications of DAI to the field of manufacturing, we comment on the applications of AI to this field. In short, researchers have investigated applications of AI to all levels of a manufacturing system and these applications are too numerous to discuss here; however, several articles have examined different aspects of this relationship between manufacturing and AI including [Cook and Emrich, 1987; Bratko, 1988; Kempf, 1988; Meyer et al., 1988; Rogers and Williams, 1988; Swyt, 1989; Tzafestas, 1990; Shapiro, 1992; Walker et al., 1994; Benjamin et al., 1995; Toguyeni et al., 1995; Tzafestas and Verbruggen, 1995; Wozny and Regli, 1996].

The specific topic of scheduling of a FMS using AI techniques has received considerable attention [Ben-Arieh et al., 1988; Charalambous and Hindi, 1991; Fischer, 1993; Hindi and Singh, 1995; Huang et al., 1997]. Knowledge-based scheduling approaches were an attempt to improve upon the deficiencies of the more traditional approaches to industrial scheduling that used the techniques of operational research [Hindi and Singh, 1995]. The bulk of these scheduling systems described centralized approaches to knowledge-based scheduling.

2.4.2 Manufacturing and MASs - Introduction

One of the first articles that discussed the potential of distributing the intelligence of a manufacturing system among its elements and also of using a heterarchic architecture was [Hatvany, 1985]. O'Hare [1990] described the difficulties that were encountered during the development of a CIM system and how DAI has the potential to solve some of the inherent complexities in implementing a CIM system. Albayrak and Krallman [1995] gave a state of the art review of research projects that have used DAI to control a manufacturing system. They also provided an extensive introduction to blackboard systems and the description of a specific

implementation of a blackboard system. Kokkinaki and Valavanis [1995] compared single agent or AI planning with multi-agent planning or DAI planning as applied to automated manufacturing systems.

2.4.3 Manufacturing and MASs - Factory or Enterprise Level

Three major current initiatives are actively investigating the general idea of agent-based manufacturing [AARIA, 1997; AIMS, 1997; SIAM, 1997]. The first initiative is the Sandia Intelligent Agents for Manufacturing project [SIAM, 1997]. This project has an agent-based architecture that uses a public domain, Java agent template. This Java agent communicates using the KQML (Knowledge Query and Manipulation Language) protocol [Finin et al., 1994]. The Sandia group have implemented their framework on a computer integrated machining cell [Panceralla et al., 1995]. The second initiative is the AARIA (Autonomous Agents for Rock Island Arsenal) project [AARIA, 1997; Parunak et al., 1997]. The specific goal of this project is to develop an industrial-strength agent-based factory scheduling and simulation system for an army manufacturing facility [Parunak, 1997]. This project also aims to integrate the dialog with suppliers and customers into an overall agent-based manufacturing enterprise [AARIA, 1997]. The third initiative is the AIMS (Agile Infrastructure for Manufacturing Systems) project [Sriram and Candadai, 1996; AIMS, 1997]. This project aspires to develop the information infrastructure needed to implement the strategy of agile manufacturing [Sriram and Candadai, 1996]. The current form of this infrastructure is called AIMSNet [Crocker et al., 1996]. AIMSNet uses an agent-based architecture to support the concept of an agile virtual enterprise [Sriram and Candadai, 1996].

Several researchers have investigated the topic of using DAI techniques to control the scheduling of a complete FMS. In this scheduling research, the application of the contract net [Smith, 1979] was a common approach.

We present three early applications of the contract net to this scheduling problem. Shaw and Whinston [1985] made each cell of a FMS an agent and these group of agents would perform

the tasks of the FMS through contract net based cooperation. They modelled the bidding scheme using augmented Petri nets [Shaw and Whinston, 1985; Shaw, 1987]. Parunak [1985] used the hierarchical structure of the levels of a manufacturing system to apply the contract net. That is, each node of this hierarchy was made into an agent that could communicate with parent, children, and sibling nodes. Depending upon the situation, nodes could be accepting to do a task or granting a task to another node. The method of audience restriction was used to reduce the communication overhead [Parunak, 1985]. Parunak [1990] evaluated the current state of manufacturing applications of the contract net. Duffie et al. [1988] described their ideas on a heterarchical control architecture for a fault-tolerant manufacturing system. They decomposed the operation of a FMS consisting of a machining cell and an assembly cell with four types of entities: pallet, part processing, material handling, and human. Part processing entities included machine tools, assembly robots, and input and output stations. These entities cooperated with each other using the contract net method of coordination.

The remainder of the contract net applications to scheduling of a FMS are presented. Baker [1992] proposed a contract net model for the production planning and control system of a small-batch or job-shop manufacturer. This model allowed for the direct connection of customers to suppliers. That is, a customer's order would get fulfilled by cost-based negotiation among the workstations [Baker, 1992]. Rabelo and Camarinha-Matos [1994] proposed a multi-agent architecture for dynamic scheduling of a CIM system. They decomposed the architecture with four types of agents, scheduling supervisor, local spreading center, enterprise activity, and consortium, that cooperated using the negotiation metaphor [Rabelo and Camarinha-Matos, 1994]. Tchako et al. [1994] modelled and simulated a contract-net approach to structure a manufacturing system. The entities of the manufacturing system, such as workstation, cell, production line, and handling system, were considered as autonomous agents that cooperated to achieve a production program. They simulated their approach on a manufacturing system consisting of a machining cell, an assembly cell, and a pallet transportation network of conveyors [Tchako et al., 1994]. Tsukada and Shin [1994, 1996] addressed the problem of rescheduling a decentralized manufacturing

system in the case of unexpected scheduling disruptions. In their approach, the cell with a disrupted schedule politely negotiates a new schedule with other cells [Tsukada and Shin, 1996]. Saad et al. [1995] discussed a heterarchical scheduling approach for FMSs where workcells and the products to be constructed are modelled as autonomous agents. In this paper, they detailed the negotiation process that occurs between these agents. Ramos [1995] discussed a deadline-based approach for dynamic scheduling of manufacturing systems. This system was decomposed into task agents and resource agents. Task agents were responsible for carrying out tasks by negotiating with resource agents. Resource agents were responsible for managing what we referred to as the components of a FMS in section 2.2.1. Ramos [1995] also proposed an approach to renegotiation in the case of exceptions.

We continue by presenting other research that has studied the application of MASs to the higher levels of a manufacturing system. Parthasarathy and Kim [1990] proposed and studied an actor-based model for an autonomous manufacturing system. They used this model to study the capabilities of autonomous production systems in a manufacturing environment. Pan and Tenenbaum [1991] presented an agent framework for complete integration of a manufacturing enterprise. Although their approach evolved independently from the field of DAI, it clearly has strong similarities with this field [Pan and Tenenbaum, 1991]. Researchers [Cutkosky et al., 1993; Kuokka and Harada, 1995] investigated the application of MASs to the problem of integrating the process of concurrent engineering design with physically distributed team members. Sikora and Shaw [1994] discussed a multi-agent solution to the problem of integrating the manufacturing information associated with the production processes of making printed circuit boards. Barbuceanu and Fox [1995] presented a multi-agent solution to integrate the supply chain of a manufacturing enterprise.

2.4.4 Manufacturing and MASs - Shop Level

Researchers have also investigated the idea of agent-based control of the shop floor. Lin and Solberg [1992] proposed and simulated a generic framework for controlling a shop floor using

autonomous agents. This framework was intended for a highly uncertain and changing manufacturing environment. They decomposed the control architecture into the following agents: part agents, resource agents, intelligence agents, monitor agents, communication agents, and database management agents. These agents controlled and scheduled the real-time activities of the shop floor by cooperating through a multi-step negotiation process that was based upon the contract net and that used a price evaluation mechanism [Lin and Solberg, 1992]. Fischer [1994] presented a MAS framework for planning and controlling a FMS with an emphasis placed on the shop floor control system. The planning component had a six layer, hierarchical structure [Fischer, 1994]. The top two layers were the production planning level and the shop floor control level. The remaining four layers made up the internal architecture of each autonomous agent on the shop floor. These lower layers were labelled task coordination, task planning, task execution, and machine control. The specific example used to illustrate their system was the coordination between an industrial robot and a heating cell [Fischer, 1994]. Hahndel and Levi [1994a, 1994b] applied the techniques of MASs to the problem of production planning and scheduling for a FMS consisting of agents that could machine, assemble, and/or paint. Each agent consisted of two basic elements, a knowledge base and a combined communication and planning module. These agents coordinated with each other using the contract net to produce a schedule for the shop floor [Hahndel and Levi, 1994a, 1994b]. Parunak [1994] gave a preliminary report on the status of a collaborative project involving a group of industrial companies to implement agents on the shop floor. This collaboration continues today as the AARIA project [AARIA, 1997]. Shih and Srihari [1995] presented a MAS approach to control the shop floor for a surface mount, printed wiring board, assembly facility. They defined an intelligent agent as a stand alone expert system or a control unit that implemented heuristic reasoning on the shop floor. They decomposed the architecture into six types of agents — central coordination, line balancing, status monitoring, machine configuration, and scheduling — that communicated using a blackboard mechanism [Shih and Srihari, 1995].

2.4.5 Manufacturing and MASs - Cell and Component Level

At the cell level, O'Grady and Lee [1990] described a hybrid method to control a manufacturing cell. In particular, their method of control combined the actor and blackboard approaches. With respect to the application of MASs to the nonrobotic components of a manufacturing cell, Barata and Steiger-Garçao [1994] described a multi-agent approach to the real time monitoring and prognosis of a numerically controlled (NC) machine. They used the specific hardware of transputer-based computing to construct a first prototype of their system [Barata and Steiger-Garçao, 1994]. Lefrancois and Montreuil [1994] presented an agent-based approach to control the workstation components of a manufacturing system. They stressed the reasoning activities of these agents and illustrated their approach by applying it to the modelling of workstations in a rolling-mill facility [Lefrancois and Montreuil, 1994].

2.5 Manufacturing and Robotics

This section examines the overlap of the field of manufacturing and the field of robotics. This section does discuss research that applies traditional AI techniques but it excludes research that applies MASs. We limit this investigation to research of greater interest to this dissertation by concentrating on specific aspects of each research field. In short, the focus is on the cell level of a manufacturing system and on robotic assembly. In addition, this section places a greater emphasis on multi-robot assembly cells (MRACs). The issues that researchers have addressed for robotic assembly cells include the following:

- sensors / multi-sensor integration
- task planning / execution
- automatic error recovery
- assembly planning / sequencing / scheduling
- communication / coordination / control / management
- optimal design

In this list, the items progress from local to more general and related items on a single line are separated by a forward slash. For general discussions on the role of robotic systems in a CIM

factory, one can refer to [Andrews and McCarroll, 1988; Howell, 1988]. In the remainder of this section, we discuss each of these items in their own subsection.

2.5.1 Robotic Assembly Cells - Sensors / Multi-Sensor Integration

A few references on sensors and multi-sensor integration for robotic assembly are mentioned. Selke and Pugh [1986] overviewed the different uses of sensors during each stage of their generic assembly concept of feed-transport-mate. Mochizuki et al. [1987] described placing a connector on a printed circuit board using visual and force sensors. Hutchison and Kak [1989] analyzed how to optimally choose the best sensing strategies in a robotic cell with multiple sensors. Kelley et al. [1990] presented an approach to multiple sensor integration during the insertion of printed circuit boards into slots. Eccles et al. [1991] presented a method to integrate multiple vision sensors for the purpose of controlling a robotic assembly cell. They also stated that richer sensory input provides potential benefits for robotic assembly in the form of robust operation, reduced fixturing, and automatic error recovery. Werling [1991] gave a good overview of the different approaches that were being used to plan sensing strategies for vision sensors in assembly environments. Bagherzadeh et al. [1992] developed a knowledge-based expert system to interpret the data from a vision system and thereby, improve the control of a robotic assembly cell. Zussman et al. [1994] presented a technique to plan the best viewpoint to scan data when using a laser scanner during robotic assembly. Miura and Ikeuchi [1995] developed a sensing strategy for a laser range finder to be used during robotic assembly tasks. Rowland and Nicholls [1995] proposed a virtual sensing implementation to sensor integration for a flexible assembly machine that works at multiple levels of abstraction.

2.5.2 Robotic Assembly Cells - Task Planning / Execution

Recall that, task planning addresses the problem of converting mechanical assembly instructions into actions executable by an assembly robot. Both the task planning and execution of these actions by an actual robotic system are important.

Smithers and Malcolm [1987] outlined their ideas upon a behavioral approach to task

planning that would overcome the limitations of existing robotic assembly systems. Rubin and Dilworth [1992] delineated an approach to task planning that used a second generation expert system with learning capabilities and they presented an example with drilling, milling, screwing, and assembly. Tung and Kak [1994, 1996] detailed an integrated framework for sensing, task planning, and execution that they implemented with an assembly robot. Their task planning module incorporated sophisticated geometric reasoning capabilities into its planning. Najjari and Steiner [1996] described a knowledge-based task planning system for a single robot that had a learning mode and an execution mode.

2.5.3 Robotic Assembly Cells - Automatic Error Recovery

Sensors are vital to the process of automatic error recovery in robotic assembly cells. According to [Abu-Hamdan and El-Gizawy, 1992], the three phases of automatic error recovery are monitoring and detection, diagnosis, and recovery. The majority of this research used a knowledge-based approach and was done for a single assembly robot.

Lee et al. [1987] presented a model-based causal reasoning approach to error diagnosis for a robotic assembly cell. Hardy et al. [1987] described a robotic task error detection and recovery approach that used the frame knowledge representation scheme. Selke et al. [1987, 1991] presented a rule-based approach to error detection during robotic assembly that learned from its mistakes. Hou [1989] proposed a low level, error detection and recovery routine for robotic assembly. Russell et al. [1989] outlined an error recognition, diagnosis, interpretation, and correction method for collaborating robots operating in an assembly cell. Raczkowsky and Seucken [1991] discussed a knowledge-based system for the diagnosis of assembly process errors that they structured as a blackboard system. Abu-Hamdan and El-Gizawy [1992] developed an error diagnosis system for a cell that was demonstrated with a knowledge base of fifteen different assembly errors. This system was decomposed into three expert systems that cooperated with each other using a blackboard structure. Kang and Wenham [1993] developed a framework for error diagnosis and recovery for a multi-robot assembly cell (MRAC) that used concepts from the theory

of fuzzy sets. Lastly, Borchelt and Alptekin [1994] gave a good review of the problem of automated diagnosis and error recovery in a robotic assembly workcell and then proposed a different control structure for this problem.

2.5.4 Robotic Assembly Cells - Assembly Planning / Sequencing / Scheduling

Although assembly planning, sequencing, and scheduling are related, each of them emphasize different issues. Assembly planning has to do with determining and representing all feasible sequences in which a product can be assembled from its parts. The output of the process of assembly planning is typically used during sequencing and scheduling. Many researchers have studied the problem of assembly planning including [Popplestone et al., 1990; Homem de Mello and Sanderson, 1991; Huang and Lee, 1991; Missiaen, 1991; Wolter, 1991; ElMaraghy and Laperrière, 1992; Maimon and Kapitanovsky, 1992; Tönshoff et al., 1992; Wolter et al., 1992; Chen and Pao, 1993; Hara and Nagata, 1993; Kapitanovsky and Maimon, 1993; Werling and Wild, 1994; Ames et al., 1995; Cao and Sanderson, 1995; Caracciolo et al., 1995; Gutsche et al., 1995; Mascle, 1995; Noorhosseini and Malowany, 1995]. In our literature search, Park and Chung [1993] was the only paper to discuss assembly planning that directly considered a multi-robot cell and generated all parallel feasible assembly sequences.

The topic of sequencing is concerned with determining an optimal sequence and assignment of operations, given a specific robotic workcell and criteria of optimization. Freedman and Malowany [1988] described a method to select the time optimal sequence for a robotic cell with repetitive sequences of operations. They illustrated their approach on a two robot cell configured for the inspection and repair of hybrid integrated circuits and printed circuit boards. Sanderson et al. [1990] discussed an interactive tool for selecting the best assembly sequence.

Meanwhile, scheduling focuses on the proper assignment and coordination of operations during the actual execution of the robotic assembly cell. The problem of scheduling a MRAC was characterized well in [Coupez et al., 1989; Glibert et al., 1990]. They distinguished between an opportunistic, on-line approach and a quasi-optimal, off-line approach. They used an enumerative

technique in combination with a branch and bound method to find an optimal schedule. Boneschanscher [1990] proposed a method to determine a minimum time schedule for a small batch MRAC. Gasmi et al. [1990] presented a knowledge-based approach to the on-line planning and scheduling of a MRAC. Researchers [Fu and Hsu, 1993; Hsu and Fu, 1995] optimized the schedule for a two-robot assembly cell by formulating this scheduling problem as a state space search problem and then using the A^* search algorithm.

2.5.5 Robotic Assembly Cells - Communication / Coordination / Control / Management

The topics of communication, coordination, control, and management all have to do with various aspects of the proper operation of a robotic assembly cell. Each of these topics can be studied at different levels of abstraction. In general, the more abstract levels are of greater interest to this dissertation.

In the case of a robotic assembly cell, the topic of communication has to do with the sending and receiving of data, information, or knowledge between its components. Communication provides a foundation for coordination, control, and management and it occurs at different levels of abstraction. Low level communication, which is concerned with how information is physically sent and how to synchronize events, often relies on operating system constructs and overlaps with research on distributed computing environments. High level communication is more interested in the content and meaning of the information or knowledge sent and how to coordinate the operations of a robotic assembly cell at a higher level of abstraction. Gauthier et al. [1987] gave an excellent review of low level communication in the context of distributed robotics for manufacturing cells. They discussed both the various interprocess methods of communication and the hardware choices available for setting up communication between cell components [Gauthier et al., 1987]. Temmes et al. [1986] described the hardware of a two robot assembly cell and the low level communications setup for this cell. Akcadag et al. [1992] detailed the hardware and software for the low level communication and computer control of a commercial robot as a component of a flexible robotic cell.

Although the study of coordination has many facets, in this case we focus on the coordination of multiple robots in an assembly cell. Loosely coupled coordination refers to multiple manipulators working independently but having to coordinate with each other to share the common resource(s) of their environment [Shin and Epstein, 1987]. Several researchers have studied this type of coordination for the sharing of physical space [Mayer and Wood, 1986; Roach and Boaz, 1987; Duffy et al., 1989; Shih et al., 1991; Manivannan, 1993]. Tightly coupled coordination refers to multiple manipulators, usually two, physically cooperating to carry out a single task [Shin and Epstein, 1987]. The dissertation of [Maimon, 1984] was the first significant attempt to address the problem of loosely and tightly coupled coordination for a MRAC. He used an algorithm based method to optimize the control of the manipulators and operating system constructs to coordinate the resource conflicts. The dissertation of [Schneider, 1989] studied thoroughly the tightly coupled coordination of a semi-autonomous robotic system with two manipulators. The system used a four-level hierarchy to investigate the dynamic and strategic control of these manipulators. Yamada et al. [1995] discussed the applications of multiple robots in loosely and tightly coupled configurations to the problem of automobile assembly.

The topics of control and management of a robotic assembly cell have to do with directing the overall operation of the cell. The study of control architectures is closely related to these topics since such architectures define a structure for directing the operation of a robotic assembly cell. We view the topic of management as being more general or abstract than the topic of control. In either case, the topics of control and management often incorporate issues studied in the previously mentioned topics into their solution.

We begin by examining knowledge-based methods to control and manage robotic assembly cells. Chochon and Alami [1986] presented a knowledge-based system to cope with execution monitoring, action scheduling, and failure diagnosis and recovery for a MRAC. Kak et al. [1986] described an experimental, single robot, assembly cell that used a knowledge-based approach to insert a peg in a hole with the aid of a 3D vision camera. Picardat [1986] used an expert system for

planning, plan execution monitoring, and supervising a single robot of an assembly cell. Mina [1987] presented ideas on a knowledge-based reasoning module for an industrial manipulator to be used in real-time cell applications. Rembold and Soetadji [1987] outlined the early progress made towards an autonomous assembly robot with two manipulators. They indicated that an AI-based system for assembly planning, implicit programming, obstacle avoidance, assembly monitoring, and error recovery was under development. Researchers [Malowany and Malowany, 1988; Chung and Malowany, 1988] simulated a rule-based framework to control a multi-robot workcell for circuit board assembly and repair. Kim et al. [1989, 1991] proposed and simulated a knowledge-based approach to manage the operation of a MRAC. Delchambre et al. [1991] described a knowledge-based approach to process planning, scheduling, error recovery, and supervising and diagnosis for a two robot assembly cell. Ferullo and Billatos [1992] presented the conceptual design for a sensor network and knowledge based expert system to control and monitor a single robot assembly cell. Chutima et al. [1992] discussed a knowledge-based, decision support system that could react in real-time to disturbances in the control of an automated, industrial robot, assembly line. Lastly, López-Mellado [1995] presented a knowledge-based approach, which uses frames and production rules, to synthesize a real-time controller for an assembly system.

Researchers have also proposed control architectures for robotic assembly cells. Several of these architectures have been object-oriented [Booch, 1986]. Stevenson [1987] introduced a distributed architecture for an assembly cell where the individual sensing and manipulation systems ran as separate processes in parallel under the supervision of a cell controller. Maimon and Fisher [1988] described an approach to control where the components of the single robot assembly cell were represented as objects that communicated with each other with messages. Alander et al. [1990] outlined a framework for a robotic assembly cell that would use high-level, object-oriented control and that would also have the properties of modular, multipurpose, and multiarm. Esteban and Courvoisier [1991] described a multi-level, hierarchical control architecture for a MRAC. Fabian and Lennartson [1992] proposed and simulated an object-oriented method to control a workcell that contained one robot, two machines, and two buffers. Farsi [1993, 1994] detailed the

lower level aspects of an open system architecture whose goal was to allow for the flexible and modular combination of the components of a manufacturing cell. Culbreth et al. [1994] presented an object-oriented architecture for the intelligent control of a prototype manufacturing cell for the production of furniture components. Lin et al. [1994] discussed an object-oriented approach to the control of a robotic flexible manufacturing cell. In their approach, the cell controller handled coordination and synchronization issues while the individual objects, representing components, were responsible for their own activities. Finally, researchers [Kim et al., 1997; Merat et al., 1997] specified and implemented an object-oriented software architecture to control an agile manufacturing cell for light mechanical assembly. Their cell included multiple Adept robots, a Bosch conveyor system, multiple flexible parts feeders, and cameras for parts feeding and hardware registration [Merat et al., 1997].

2.5.6 Robotic Assembly Cells - Optimal Design

This subsection explores the issues that are related to the optimal design of a MRAC. He et al. [1990] developed a procedure to design a MRAC for optimal productivity. Their design considered workspaces, kinematic and dynamic manipulability of the robots, and trajectories of the end-effectors [He et al., 1990]. Rajan [1994] demonstrated that the design of assembly products and of cooperating MRACs is affected by the interactions between fixed assembly constraints and variable robot constraints. Researchers [Cardarelli et al., 1995; Pelagagge et al., 1995, 1996] presented criteria to aid in the design of assembly cells with cooperating robots. Their design criteria considered the following four points regarding such assembly cells: cell layout, assembly task characterization, precedence relationships, and no-collision planning [Pelagagge et al., 1995, 1996]. Rampersad [1995b, 1996] introduced a total productivity model for a MRAC that considered both technical and economic issues. This model, which was simulated as a discrete system, offered the opportunity to compare the total productivity of alternative system designs [Rampersad, 1996].

2.6 AI and Robotics

This section examines the relationship between the field of AI and the field of robotics. To begin with, the cross-pollination between these two fields has been substantial. For an exploration of the relationship between traditional AI and robotics, one can refer to [Basran and Petriu, 1993]. This section begins by giving a perspective on MASs and robotics and then uses this perspective to more closely explore the overlap between MASs and robotics.

2.6.1 A Perspective on MASs and Robotics

Consider the following definition for the field of robotics:

> "Robotics is the *intelligent connection of perception to action.*" [Brady, 1985, p. 79, emphasis in original]

There is a striking similarity between this definition for the field of robotics and two of the definitions for an agent listed in section 2.2.2. In essence, it is the robotic system or the robotic agent that must somehow intelligently connect perception to action. Based in part on this observation, we make four points. First, developments in robotics and MASs have occurred in parallel and continue to occur in parallel. Second, robotics is a common application area for MASs. Third, the main difference between a robotic agent and the other types of agents displayed in figure 2.3 is that a robotic agent should have a physical implementation as well as a computational implementation. The majority of nonrobotic agents only have computational implementations. Fourth, except for the property of social ability or the ability to communicate with other agents, the majority of autonomous robotic systems would pass the weak notion of agency [Wooldridge and Jennings, 1995].

We present a framework for viewing research in the field of robotics from a MAS perspective. Recall that, research in MASs was split into micro aspects and macro aspects in section 2.2.1. Micro aspects have to do with the issues internal to the operation of an agent or intra-agent issues. Macro aspects have to do with the issues between a group of agents or inter-agent issues. Given that most robotic systems are either manipulators or navigators as described in

section 2.3.1, this leads to four research groupings:

- manipulators and intra-agent issues
- manipulators and inter-agent issues
- navigators and intra-agent issues
- navigators and inter-agent issues

The following four subsections summarizes research on manipulators and navigators that fits into these groupings but does not stress robotic assembly.

2.6.2 Manipulators and Intra-Agent Issues

A few researchers that examined control architectures for robotic manipulators are discussed. Taylor and Grossman [1983] described the low-level aspects of implementing a system architecture for a manipulator. They also detailed AML (A Manufacturing Language), a high-level language for programming a manipulator. A complete control system architecture for a manipulator was described in [Albus et al., 1987; Lumia et al., 1989]. Although this architecture was intended for telerobot control, it provided a rich framework. This framework was composed of three multi-level hierarchies: task decomposition, world modelling, and sensory processing [Albus et al., 1987; Lumia et al., 1989].

Various intra-agent aspects of the implementation of control architectures for manipulators are presented. Harmon et al. [1984, 1986] presented the use of a distributed blackboard mechanism to coordinate the subsystems of an advanced welding robot in an automated cell and a robot vehicle for ground surveillance. They used the blackboard to store and maintain consistency in the world model that was divided into model of self and model of environment [Harmon et al., 1986]. Sakane et al. [1993] developed an agent-based approach for a cooperative multi-sensor system. They applied this system to the robot vision task of inspecting for water leaks on the sleeve of a valve [Sakane et al., 1993]. Bohner [1994, 1995] presented an agent-based method to control the joints of a single manipulator or multiple manipulators. This method used a distributed fuzzy rule base and the joint agents communicated with each other in order to achieve global suboptimal control [Bohner, 1994, 1995]. Zanichelli et al. [1994] discussed a multi-agent

framework to control a manipulator. They tested their framework by simulating the manipulator task of exploring an unknown workspace and manipulating blocks. Suehiro and Kitagaki [1995] described an agent-based implementation of the control of a direct drive manipulator. Kao et al. [1996] proposed an agent-based approach to collision avoidance for a manipulator. They simulated their approach for a six, DOF (degree of freedom), Puma robot that was performing a welding task.

2.6.3 Manipulators and Inter-Agent Issues

The question of how to coordinate the physical cooperation of multiple manipulator agents has been addressed by the robotics community but is not of direct concern to the field of MASs. Typically, physical cooperation studies the manipulation of a single object with two such agents. This type of coordination is also referred to as tightly coupled coordination [Shin and Epstein, 1987]. The methods for physical coordination occur at a lower level and on a shorter time scale than the methods of coordination of MASs listed in figure 2.2. Shin and Epstein [1987] proposed low-level communication primitives to support a distributed modular architecture for an integrated multi-robot system with a cooperative capability. Many researchers have investigated the difficult problem of physical cooperation including [Maimon, 1984; Maimon and Nof, 1985; Classe and Feldman, 1986; Hawker et al., 1986; Nof and Hanna, 1989; Schneider, 1990; Kountouris and Stephanou, 1990; Yun, 1993; Yamada et al., 1995].

Researchers have also studied approaches to multi-agent planning where the agents are robotic manipulators. Multi-agent planning of robotic manipulators often takes a global or centralized approach and this planning is related to the study of loosely coupled coordination [Shin and Epstein, 1987]. An example of loosely coupled coordination is the specific problem of manipulators sharing a workspace [Shin and Epstein, 1987]. Just a note that this problem of sharing a workspace was already discussed in subsection 2.5.5.

Georgeff [1983] proposed a method to synthesize a multi-agent plan by inserting communication acts into single agent plans to synchronize activities and avoid harmful interactions.

This method was illustrated for the case where two manipulators had to place metal stock in a single lathe [Georgeff, 1983]. Amori [1988] discussed a multi-agent planner that used a hierarchical approach to construct a static plan to be carried out by a set of robotic agents. Kamel and Syed [1988] presented an object-oriented approach to the multi-agent planning problem that was suited for agents with similar capabilities. They applied their approach to the problem of metal cutting a part from raw material using three robotic agents [Kamel and Syed, 1988]. Kamel and Syed [1994] presented a method of multi-agent planning for agents with disparate operational capabilities. They used interagent constraints during the planning process to avoid interagent conflicts and they applied their approach to the problem of metal cutting a part from raw material using three robotic agents with disparate capabilities [Kamel and Syed, 1994].

Amamiya et al. [1993] proposed an agent language to help in the modelling of the control of a multi-robotic-agent system. One of the examples that they demonstrated with their language was two robotic agents physically cooperating to translate a bar.

2.6.4 Navigators and Intra-Agent Issues

The overlap between mobile robotic agents and the field of MASs is much more substantive than between manipulative robotic agents and the field of MASs. This subsection examines this overlap between mobile robots and the field of MASs.

The study of control architectures is one significant common research area. We give only a very brief introduction to this topic for extensive surveys refer to [Ferguson, 1992; Müller, 1996]. Müller [1996] identified four types of control architectures: deliberative, reactive, interacting, and hybrid. In a deliberative architecture, the agent plans what to do before acting. In a reactive architecture, the agent reacts to situations that arise in its environment. In an interacting architecture, the agent knows how to cooperate and coordinate with other distributed intelligent agents [Müller, 1996]. A hybrid architecture attempts to optimally integrate the deliberative, reactive, and interacting architectures into a single architecture [Müller, 1996].

We mention a few references where robotic agents were physically implemented with

various control architectures. The mobile robot, SHAKEY, that was described as an application of AI techniques in the development of an integrated robot system [Nilsson, 1969] was implemented with a deliberative control architecture. Meier [1987] implemented a deliberative architecture on a mobile robot using an expert system. Brooks [1986a, 1986b, 1990, 1991a, 1991b] proposed, developed, and advanced the subsumption architecture for mobile robot control. This architecture was essentially a layered, reactive control architecture. Connell [1992] implemented a hybrid architecture on a navigator that mapped indoor office environments. Bonasso et al. [1995] described their experiences implementing hybrid architectures on several different types of robotic systems. For further discussions on robot architectures, one can refer to [Malcolm et al., 1989; Neves and Gray, 1995].

Another interesting development is the implementation of the control architecture of a mobile robot agent using a MAS approach. Elfes [1986] described a distributed control architecture for an autonomous mobile robot that was implemented using a blackboard structure. Researchers [Thomas et al., 1993; Tigli and Thomas, 1994] proposed a method to design mobile robot controllers using a multi-agent approach. Bozinovski, [1994] implemented a MAS to control a mobile robot using a multitasking system. Berge-Cherfaoui and Vachon [1994] integrated reactive behavior into the control architecture of their mobile robot by implementing the system using an agent-based approach with a blackboard structure. Researchers [Hu et al., 1995; Hu and Brady, 1996] developed a modular hybrid architecture for control of a mobile robot that they implemented using multiple locally intelligent control agents on a parallel processing system.

In closing, we mention a reference on programming of navigators. Lim [1994] presented a programming language for mobile robots that represented the program as a graph structure of agents.

2.6.4 Navigators and Inter-Agent Issues

Researchers have studied approaches to multi-agent planning and scheduling where the agents are robotic navigators. With respect to multi-agent planning and scheduling, we observed

that on the whole centralized approaches were more common for manipulators while decentralized approaches were more common for navigators. Pape [1990] gave a good summary of the differences between centralized and decentralized multi-agent planning and scheduling approaches. More specifically, this article developed a mixed strategy, both centralized and decentralized, for the problem of task planning and scheduling for manipulators and navigators. This strategy was simulated for the case of three heterogeneous robots carrying out user requests for copies of microfilm in an indoor office environment [Pape, 1990]. Premvuti and Yuta [1992] presented a decentralized solution to the problem of moving without colliding and sharing a pathway for a multiple navigator system. Shibata and Fukuda [1993] proposed a novel strategy for motion planning of multiple navigators. In their approach, each robot planned its motion based upon its own situation and the motion planning was optimized by applying a genetic algorithm [Shibata and Fukuda, 1993]. Rausch et al. [1995] described a technique for multiple robots to coordinate with each other and avoid collisions during the crossing of a traffic intersection.

Several researchers have investigated various issues regarding the cooperative pushing, transporting, or carrying of an object from one location to another with mobile robots. Caloud et al. [1990] described a project to control the operation of dozens of mobile robots in an indoor environment. The majority of tasks involved transporting of relatively small objects between locations. They presented their current solutions to the multi-robot research issues of motion planning, task planning, and task scheduling [Caloud et al., 1990]. Researchers [Ishida et al., 1991; Asama et al., 1992] described the communication and cooperation of a system of multiple mobile robots that was based upon the actor formalism. They used the contract net for high level negotiation between mobile robotic agents and they demonstrated their solution for the task of multiple navigators pushing an object [Ishida et al., 1991]. Kawauchi et al. [1993] presented an approach to distributed decision making where each unit or mobile robotic agent should obey some simple control laws. They implemented this approach in the context of the Cellular Robotic System (CEBOT) [Fukuda et al., 1989], a reconfigurable or self-organizing robotic system. They demonstrated their decision making approach for the task of multiple units carrying a single object

[Kawauchi et al., 1993]. Kube et al. [1993] described a mechanism to control the group behavior of multiple mobile robots executing a collective task. They simulated their mechanism for the task of pushing a box. Noreils [1993] described an architecture for cooperative, mobile robotic agents that was demonstrated with two actual examples using two indoor mobile robots. The first example of cooperation was where one robot led the second robot to a target destination, while the second example was cooperative box pushing [Noreils, 1993]. Aylett et al. [1995] discussed the architectural issues associated with two mobile robots cooperating to relocate an object to a specified destination. Brown and Jennings [1995] presented and implemented a cooperative model for two mobile robots pushing a box. In their model, one robot steered while the other robot pushed the box between them [Brown and Jennings, 1995]. Ota et al. [1995] proposed a layered hierarchical structure for the motion and task planning of multiple mobile robots. They simulated their structure for the task of transporting many objects between two points [Ota et al., 1995].

We briefly summarize a few references that have explored other cooperative tasks with multiple mobile robots. Balch and Arkin [1994] compared three different types of communication for reactive, multi-navigator systems. They evaluated these communication schemes by simulating and experimenting with mobile robots that were carrying out the three tasks of foraging, consuming, and grazing [Balch and Arkin, 1994]. Parker [1994] experimented with the cooperation of real mobile robots for the task of toxic waste cleanup. Dubreuil [1995] proposed a simple behavioral and communication architecture for the task of foraging and grazing with multiple mobile robots. Lin and Hsu [1995a, 1995b, 1996a, 1996b] proposed and simulated two distinct coordination protocols for multiple, homogeneous mobile robots working together on an object-sorting task. These articles detailed the algorithms for these two protocols. The object-sorting task was essentially a foraging task with different destination locations for each object of a different type.

We end this subsection by discussing two general articles on the topic of navigators and inter-agent issues. Dudek et al. [1993] presented a taxonomy of the different ways that a collection

of autonomous robotic agents could be structured. They used the term swarm to denote a large number of smaller, simpler robots. They identified seven dimensions along which robotic swarms could vary: size, communication range, communication topology, communication bandwidth, reconfigurability, processing ability, and composition [Dudek et al., 1993]. Mataric [1995] overviewed the issues and approaches in the group behavior of autonomous agents with a slant towards the mobile robotics community. She divided the research of MASs into the three major strands of distributed artificial intelligence, physical multi-robot systems, and artificial life. Their group demonstrated the group behaviors of avoidance, following, aggregation, dispersion, flocking, wandering, foraging, docking, and learning to forage with approximately 20 mobile robots [Mataric, 1995]. Artificial life studies, interrelates, and intermingles the group behavior of intelligent agents from a biological and a robotic perspective. For more information on artificial life from a robotics perspective, one can refer to the following references [Deneubourg et al., 1991; Arkin and Hobbs, 1993; Drogoul and Ferber, 1993; Kube and Zhang, 1993; Mataric, 1993; Numaoka and Takeuchi, 1993; Parker, 1993; Yanco and Stein, 1993; Arkin and Ali, 1994; Beckers et al., 1994; Mataric, 1994; McFarland, 1994; Nolfi et al., 1994; Smithers, 1994; Steels, 1994a, 1994b, 1995].

2.7 Manufacturing, AI, and Robotics

This section concentrates on research that takes place at the intersection of the field of manufacturing, the field of AI, and the field of robotics. For a short but general commentary on the subject of robotics in manufacturing from the point of view of the field of AI, one can refer to [Bekey, 1996].

In particular, this section discusses applications of agent-based concepts to robotic assembly cells. We categorize these applications into agent-based control architectures for assembly manipulators, multi-agent planning for assembly, and agent-based assembly systems.

2.7.1 *Agent-based Control Architectures for Assembly Manipulators*

Hörmann et al. [1989] implemented the control architecture of an autonomous assembly robot with two manipulators using the blackboard structure. They decomposed the operation of this robot into the following modules: plan interpreter, resource scheduler, operation coder, communication system, world model, and failure monitor. This blackboard structure was later modified in order to decentralize the responsibility for coordination and scheduling to all system components [Laengle and Lueth, 1994].

Researchers [Herd et al., 1990; Duffy and Herd, 1991; Duffy et al., 1991] proposed and developed an agent-based control architecture for a multi-sensor, MRAC. In their decomposition, the robot control system was broken down into three major subsystems: robot plan execution, sensing, and error detection and correction. Each subsystem was implemented as a group of agents with their own organizational structure. For example, they implemented the robot plan execution system as a hierarchy of agents. They used a global information approach to coordinate these subsystems. They divided the information into a task pool, world model pool, and error pool, and they represented this information using the frame knowledge representation scheme. They applied their architecture to a cell with two different commercial cartesian robots, a wrist/force torque sensor, touch sensors, multiple cameras, and a 3-D laser ranging system. At the time, they were starting work on a parallel processing architecture with a shared memory model to support the real-time operation of the robots [Herd et al., 1990; Duffy and Herd, 1991; Duffy et al., 1991].

The most thoroughly investigated agent-based control architecture for a single manipulator operating in an assembly cell has been developed by the following researchers [Oliveira et al., 1991; Ramos and Oliveira, 1993; Oliveira, 1994; Oliveira and Rocha, 1994; Oliveira and Ramos, 1996]. They decomposed the operation of an assembly robot into six intelligent subsystems or agents: task level planner, execution level planner, vision-based object recognition system, three-dimensional object recognition system, data base representing object features, and world

descriptor. The cooperative layer of each agent consisted of a high level communication module, an agent acquaintance module, and a self model module, while the internal communication and decision making of each agent was implemented using the blackboard structure. The agents cooperated with each other using both client/server and negotiation protocols.

Tzafestas and Tzafestas [1991] outlined a blackboard decomposition for an assembly cell consisting of a stationary manipulator, mobile manipulator with attached platform, pallet, worktable, input stream, output stream, dustbin, toolkit, and sensors. They partitioned the domain knowledge of the assembly cell into four sources: arrival, access pallet, assemble, and user. They also described a control mechanism to coordinate the activity of the blackboard system. Tzafestas and Tzafestas [1995] proposed a layered reactive architecture for assembly robots. In their architecture, each robot had a repertory of behavioral components and a reasoning control system to choose between these behaviors. They evaluated their architecture through simulation with an assembly cell consisting of a transfer robot, an assembly robot, a part storage pallet, and two conveyor belts.

Osato [1993] proposed and implemented an agent-based control architecture for a manipulator that worked on the well-known puzzle, Towers of Hanoi, a simplified assembly task. From their viewpoint, each agent was a logical component of high-level control for a robotic system.

Jagannathan and Evans [1994a, 1994b] proposed a cooperative, hierarchical, agent-based control architecture for a manipulator operating in an assembly line. They decomposed the hierarchical structure into the following four layers: task level supervisor, task planner and organizer, task execution, and path tracking. Each layer was further decomposed into the set of agents necessary to realize an intelligent controller that is one with learning, self-organization, and high-level supervision.

Tsuda and Takahashi [1995] proposed and simulated a multi-agent approach to detect and change the contact state of a manipulator while it carried out the task of single-peg and dual-peg

insertion. Their approach used sensors and local models to determine the contact state of the robotic assembly [Tsuda and Takahashi, 1995].

2.7.2 Multi-Agent Planning of Assembly Manipulators

This subsection discusses multi-agent planning for manipulators that do assembly. Note that the topic of scheduling of a MRAC is related and this topic was already discussed in subsection 2.5.4.

Nagata et al. [1988] developed a system for planning of multiple manipulator agents. This system was decomposed into a fundamental planning subsystem for multiple manipulators and a subsystem for detecting and avoiding collisions of cylindrical-type robots. They demonstrated their multi-agent planning system for two manipulators operating in the blocks world and for three manipulators cooperating on a vertical assembly with three main parts and two screws [Nagata et al., 1988].

Hörmann [1992] described the on-line task planning and execution system for a two manipulator assembly robot. This system was implemented using a centralized knowledge-based approach.

Chu and ElMaraghy [1993] presented an integrated task planning and execution system for a MRAC. Their solution integrated the system with a two-phase, five-layer structure. The two phases were off-line planning and on-line execution. They divided the multi-agent planning problem into the two subproblems of mono-agent plan generation and manipulator task allocation. They used a finite state machine to synchronize the robots and prevent collisions in the shared workspace. They demonstrated their system on a two robot, assembly cell that vertically assembled a dishwasher power module.

Nof and Rajan [1993] described an approach to task planning for a MRAC where physical cooperation was emphasized. They divided the planning process into two steps. Given the assembly and cell description, the output of the first step was a matrix representing the capabilities

of robot sets for the assembly tasks and the robot and cell constraints. Given the output of the first step, the second step generated a consistent and coordinated global execution plan.

2.7.3 Agent-based Assembly Systems

Hasegawa et al. [1994] presented a holonic architecture for planning and scheduling a manufacturing system. A holon is an agent that has an information processing component and may or may not have a physical processing component. They explained their architecture using a simplified robotic assembly testbed. They decomposed the operation of this testbed into six types of holons or agents: product, part, machine, machine-type, operation, and coordinator. Given a product holon generated by a customer, these holons negotiate with each other using the contract net during the planning process in order to determine an assembly sequence and a list of machine-types needed for the assembly. They proposed and simulated a near optimal scheduling approach for their holonic architecture that used a Lagrangian relaxation technique [Hasegawa et al., 1994].

Rizzi et al. [1997] presented their initial thoughts on an agile assembly architecture for modular precision assembly systems. They described an agent-based approach for the distributed design, modelling, simulation, and control of a minifactory for precision assembly. In their minifactory, the focus was on vertical assembly with four degrees of freedom (DOF). This assembly problem was uniquely structured as the cooperative execution of a 2 DOF, stationary manipulator agent with a courier agent, a 2 DOF planar manipulator. The courier agent had the dual role of both transporting the subassembly and of transiently cooperating with the manipulator agent to form a four DOF manipulator for precision assembly. In their system, the individual agents were made both capable of and responsible for negotiating for the use of shared resources. In particular, they proposed a distributed reservation scheme to resolve space conflicts. They foresee this distributed approach offering a number of advantages including: modularity, robustness, scalability, and ease of use [Rizzi et al., 1997].

2.8 Synopsis

This chapter delineated the research that defines the larger context within which our research occurs. This delineation began by providing a brief overview of the three major research fields that provide the foundation of this dissertation. These overviews were used to establish the most relevant aspects of each research field for this dissertation. The remainder of this chapter studied the interdisciplinary relationships between these research fields by examining research that overlapped among all combinations of these fields.

The sections and subsections of this chapter will be referred to as necessary in order to help indicate how our research differs from or relates to the existing research.

Chapter 3

Framework for Agent-based Management

This chapter details our framework for the agent-based management of a task-level multi-robot assembly cell (MRAC). This chapter is broken down into six sections. The first section addresses issues that are fundamental to the framework described in this dissertation. The second section introduces the reader to this framework. The third section presents the internal structure that each agent has in common. The fourth section discusses the aspects where each of the agents differ from one another. The fifth section describes the content of the messages that are being communicated among the agents in order to cooperate with each other. The last section examines how this framework is related to and distinguishes itself from the existing research.

3.1 Fundamental Issues

This section clarifies specific characteristics of the problem under study in this dissertation and discusses motivations for proposing this framework. Although our framework supports the various strategies of automated manufacturing systems discussed in section 2.1, this chapter presents it in relation to the strategy of FMSs.

3.1.1 Specific Characteristics of the Problem

An important observation from the previous chapter is that similar problems are being addressed with different scopes. For instance, researchers have addressed the question of a control architecture for an entire manufacturing system and for a single assembly robot. We make

this point to reiterate that the scope of this dissertation is on assembly cells. We further clarify by noting that assembly cells are different from assembly lines [Lotter, 1986; Makino and Arai, 1994]. Given that ranges of production volume vary from low to medium to high or mass, then flexible assembly cells and lines are used for medium production volumes, that is, between the lower volumes of manual assembly and the higher volumes of dedicated automatic assembly [Lotter, 1986; Makino and Arai, 1994].

Hara and Azuma [1988] discussed the differences between robotic assembly lines and robotic assembly cells. In general, robotic assembly lines are larger in scale and have higher productivity but lower flexibility than robotic assembly cells. Typically, robotic assembly lines have a single flow of products through the system and these lines are intended to realize mixed production of several product types with similar structures and a fixed assembly sequence [Hara and Azuma, 1988]. Meanwhile, robotic assembly cells can be organized into a cell production system with multiple possible product flows through the system. Such a cell production system makes it easier to adapt to frequent product changes, allows for variability in the assembly sequence, and creates a system with high extensibility [Hara and Azuma, 1988]. Because of these positive characteristics of a cell production system and since robotic assembly cells can be organized into a line structure, we concentrate on robotic assembly cells in this dissertation.

The second issue is concerned with the coordination of the components of the robotic assembly cell. We investigate the loosely coupled coordination [Shin and Epstein, 1987; Hern, 1988] of the components of this assembly cell. With respect to the robotic manipulators, this means that the manipulators do share a common workspace but parts are only handled by a single manipulator at a time. The topic of tightly coupled coordination of manipulators was already discussed in subsections 2.5.5 and 2.6.3.

Lastly, although the concept of a manufacturing cell is integral to FMSs, actual cells vary widely [Watt, 1985; O'Grady et al., 1987]. We identify four dimensions along which cells can vary:

- type of cell
- granularity of components
- scale
- level of abstraction

The type of cell refers to the manufacturing purpose for which the cell is constructed, such as machining, inspection, assembly, painting, welding, etc. The granularity of the components refers to the size and degree of complexity of the individual components. This dimension may vary from fine to medium to coarse. For instance, a robot with external sensors and error detection routines is larger grained than a robot that operates without such sensors and routines. The scale dimension is a function of the actual number of components that make up the cell. The scale of a cell may vary from small to medium to large. For example, a small scaled cell could consist of a single machining center plus a loading/unloading robot while a large scaled cell could manufacture an entire product family [O'Grady et al., 1987]. The level of abstraction refers to the type of instructions that the cell and its components can accept. Our research is directed towards robotic assembly cells with coarse grained components, on a medium scale, and task-level instructions as the basic level of abstraction.

3.1.2 Motivations

A MRAC is a complex, spatially and temporally distributed, physical part processing system. The basic question is how to best organize and operate such a system. Ideally, one would like to design this system so that it has as many of the following properties as possible:

- modular
- extensible
- reliable
- efficient
- robust
- maintainable
- adaptable

One of our general motivations is to work towards a MRAC that has as many of these systemic properties as possible.

Broadly speaking, two specific but related approaches to developing systems with many of these properties have evolved. The *object*-based/oriented approach developed out of the field of software engineering [Booch, 1986; Korson and McGregor, 1990]. Researchers that have studied object-based approaches to managing robotic assembly cells were discussed in section 2.5.5. The second approach is the *agent*-based approach that we thoroughly discussed in the previous chapter. The agent-based approach was chosen for two main reasons. First, the agents of a MAS and the components of a MRAC share the characteristics of being autonomous, loosely coupled, heterogeneous, asynchronous, and distributed. Second, although these two approaches share common architectural issues [Aarsten and Brugali, 1997], the agent-based approach organizes and structures a system at a higher level of abstraction.

To operate a MRAC at a higher level of abstraction is a key motivation for proposing the agent-based management of this assembly cell. There is a strong link between the level of automation of a manufacturing system and the level of abstraction at which the instruction(s) are input to this system. For example, specifying the instruction as simply build product is obviously at a higher level of automation and abstraction than specifying a sequence of mechanical instructions to assemble a product. Such a sequence of mechanical instructions is the output of the process of assembly planning [Gottschlich et al., 1994] and it is the input that we assume for our cell. In particular, we expect this sequence to be specified as a directed graph that respects the precedence relations of the product assembly. A primary goal of our agent-based management approach is to provide a framework to flexibly support this assembly planning level of abstraction for a MRAC.

Another motivating aspect of our framework, which also relates to the level of abstraction, is to provide a MRAC techniques for task-level scheduling and programming. In this case, the tasks are assembly operations that are specified as mechanical instructions as discussed in the previous paragraph. The assumption is that assembly operations specified as task-level instructions are in the current manufacturing environment the most natural level of abstraction to

encapsulate the functionality of the agents of an assembly cell.

In closing, we just want to add that one of the obvious motivations for using an assembly cell with multiple robotic manipulators is to reduce the cycle time of the cell [Chu et al., 1991].

3.2 Introduction to the Framework

This section introduces our framework for agent-based management of a MRAC. This section deliberates on the problem of how to partition such an assembly cell into an agent-based system before providing a basic description of the framework. A condensed description of this framework that used the acronym FABRIC (Flexible Agent-Based RobotIc assembly Cell) was provided in [Basran et al., 1997b].

3.2.1 How to Partition?

One of the basic questions is how to most appropriately partition a MRAC into an agent-based system. Through the process of partitioning, researchers decide how to distribute the functionality and intelligence of the overall cell into specific agents. Tzafestas [1994] gave a general discussion on the problem of how to decompose a modern manufacturing system into a MAS system. The three types of decomposition described were task-based, robot-based, and hybrid. Examples of these types of decompositions were given for assembly cells. According to [Tzafestas, 1994], task-based decompositions are scalable on the number of tasks and work best for a static physical setup, robot-based decompositions are scalable on the number of robots and are more useful for assessing the performance of a variety of multi-robot cells, and hybrid decompositions are scalable on both task agents and robot agents. This article also argued that the central issue of decomposition was the granularity level of the individual agents. The last relevant point from [Tzafestas, 1994] is the observation that there is a coupling between how to decompose a system and the method of coordination being used by the group of agents.

The term robot-based decomposition is too specific and needs to be generalized to equipment-based decomposition since this latter decomposition was used by many of the agent-

based decompositions discussed in subsections 2.4.3 and 2.4.4. With this slight change in terms, we have decomposed our MRAC using a hybrid approach.

Several factors influenced our decomposition. The first factor was the decision to use the contract net [Smith, 1979] as the cooperative method to coordinate the behaviour of the majority of agents. Second, equipment-based decompositions support extensibility of the physical components of an assembly cell and could potentially reduce the physical setup time. Third, the question of how to manage the shared space of an assembly cell among the agents needed to be addressed. Fourth, the support for task-level scheduling and programming was another key factor in determining how to best partition the system.

3.2.2 Basic Description of the Framework

This subsection commences by summarizing the basic operation of the contract net [Smith, 1979, 1980; Smith and Davis, 1981; Davis and Smith, 1983]. The contract net is a high-level communication protocol for dynamically allocating or scheduling tasks between managers and contractors and it is particularly suited for assembly since task execution time is quite variable. An agent has a task that it needs done but that it cannot perform itself. This agent, the manager or contractee, broadcasts a task announcement message with a deadline for receiving bids to its acquaintance agents. The acquaintance agents that can perform the task specified in this announcement message respond to the manager with a message containing a bid. After waiting until the deadline, the manager selects one of the bidding agents and then sends a message announcing which one of them was awarded or granted the task. If the manager did not receive any bids by the deadline, it reannounces the task. The successful agent becomes the contractor and assumes responsibility for carrying out the announced task. When the contractor completes its task, it sends a report message to the manager. Although that completes this synopsis of the contract net, one should note that other types of high-level communication messages including directed awards, termination, availability, request, and information were also discussed in [Smith, 1979].

Having addressed the background issues, we now describe the agents that cooperate to manage the operation of the MRAC. We partitioned this assembly cell with four types of agents. Figure 3.1 depicts the communication messages that are exchanged among all of these agents. An arrow points from an agent that sends a message to the agent that receives this message. Just a reminder that we decompose our assembly operations into the two distinct phases of part assembling and part fetching.

The task scheduling agent (TSA) is responsible for scheduling the assembly operations specified as task-level instructions so that the precedence constraints of the current assembly are satisfied. In other words, the input to the TSA is the output of the process of assembly planning or more precisely a directed graph specifying the precedence relations of the assembly and the associated task-level mechanical instructions. The TSA is a manager agent that schedules these tasks using the contract net. In figure 3.1, the arrows connected to the TSA depict the messages exchanged during contract net negotiation.

Each assembling agent (ASA) is a self-contained, sensor-based, robotic system that is capable of physically manipulating parts to execute the part assembling phase of an assembly operation. Figure 3.1 shows the general case with n ASAs. Each ASA can and does respond to the TSA as a contractor agent that can execute assembly operations specified as task-level instructions. These assembly operations are carried out by the ASA by coordinating with the part presentation agent (PPA) and the shared space agent (SSA). That is, each ASA uses the contract net method as a manager to find a PPA to fetch parts for assembly and it cooperates with the SSA in order to avoid collisions when it must enter a shared region of space. All of these messages exchanged between an ASA and the other agents are also depicted in figure 3.1.

Each part presentation agent (PPA) is a mechanical piece of equipment that can obtain and deliver parts to one or more of the ASAs to a reasonable degree of spatial accuracy. A PPA could be an automated guided vehicle (AGV), an integrated hopper/feeder/conveyor belt system, etc. Figure 3.1 displays the general case with m PPAs. The concept of a part presentation system or a

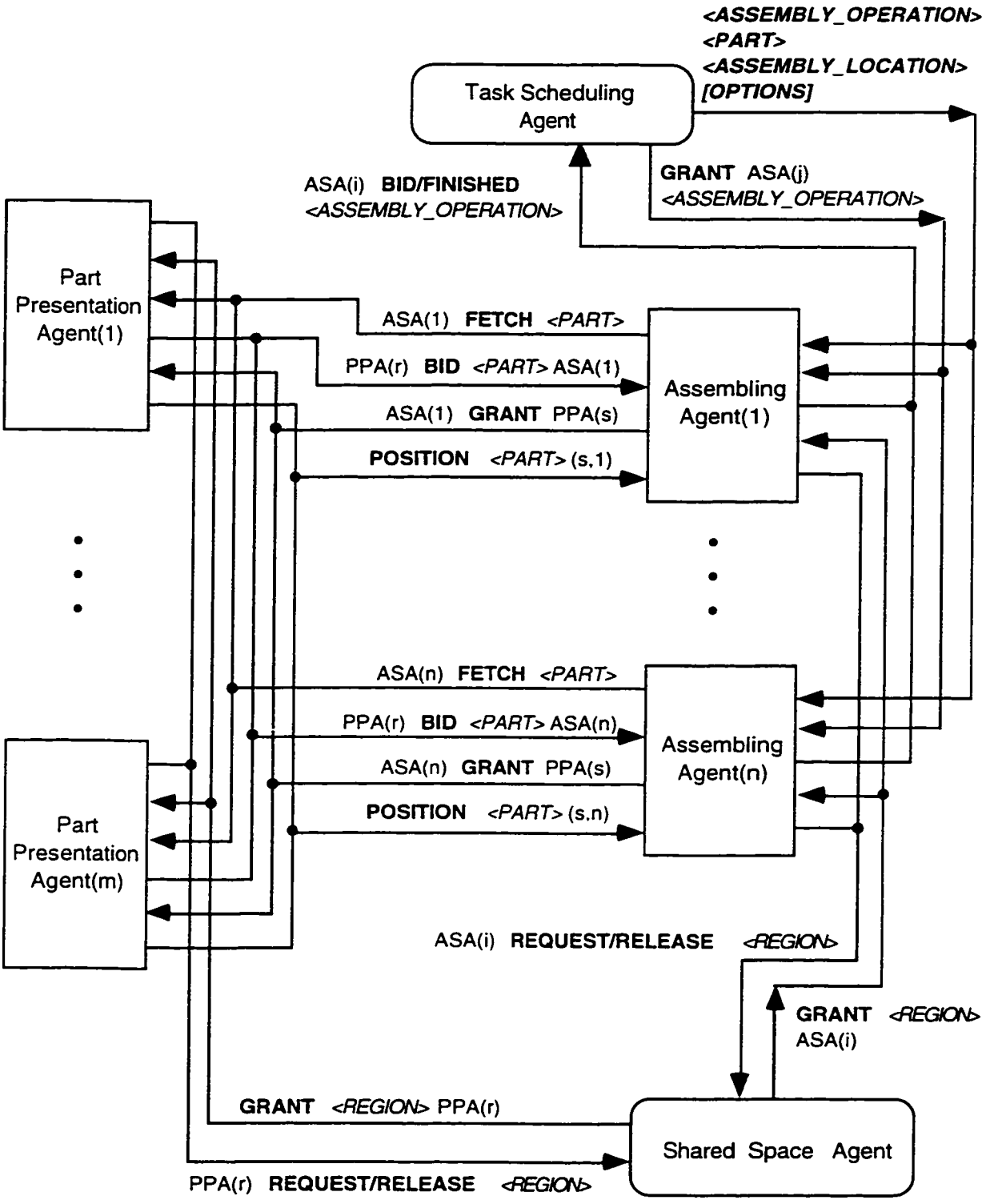


Figure 3.1: Communication messages exchanged between the agents of a MRAC.

flexible parts feeder is well accepted [Shaum, 1986; ElMaraghy and Knoll, 1989; Mazouz and Han, 1989; Ross, 1994; Rao et al., 1996; Causey and Quinn, 1997]. Each PPA responds as a contractor agent to any ASA that it can deliver parts to, if it is available. If necessary, a PPA cooperates with the SSA to prevent collisions in the assembly cell.

The shared space agent (SSA) is responsible for ensuring collision-free performance of the ASAs and PPAs of the cell by properly managing the common resource of physical space. We presume that the physical space of the assembly cell is divided into regions and each region is shared by at least two physical agents. The physical agents of the assembly cell are aware of which regions they need access to and they cooperate with the SSA to guarantee mutually exclusive access to all shared regions of the cell.

With respect to our framework for the agent-based management of a MRAC, we assume that there is a transportation system to move subassemblies both into and out of this cell. In other words, we assume that this cell is part of a larger cell production system [Hara and Azuma, 1988].

3.3 Generic Internal Structure of Each Agent

The generic internal structure of our agents combines the essential aspects of agents from the field of MASs described in section 2.2 with our previous research in multi-robot assembly systems [Petriu et al., 1996]. This section introduces this internal structure of each agent.

Figure 3.2 shows the three basic elements that define the internal structure of each agent. The three elements are a messaging system, a knowledge base, and a communicating state-machine. The messaging system can both send and receive messages from other agents. The messaging system contains a message queue that stores all of the messages received by an agent. Each agent has the option to deal with the messages on a first come, first serve basis or on a priority basis.

Although there are several, different knowledge representation schemes [Barr et al., 1982; Brachman and Levesque, 1985; Davis et al., 1993], in our case, the knowledge base of each agent

uses a frame knowledge representation scheme [Minsky, 1985]. A frame is a data structure that typically represents a single object, a class of related objects, or a general concept [Karp, 1993]. Each frame has the property of being named and consists of a set of slots. Each slot describes a single attribute of what its frame represents. The properties of slots are called facets. Typical slot facets are name, value, datatype, and value restriction [Karp, 1993]. Frames are typically organized into hierarchies and therefore, benefit from the concept of inheritance. A collection of frames in one or more inheritance hierarchies is a knowledge base [Karp, 1993].

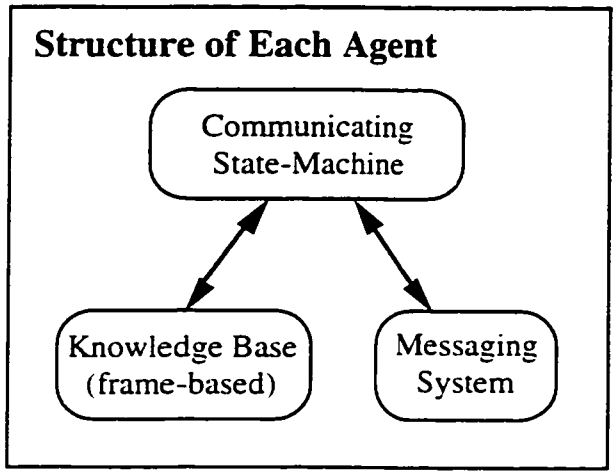


Figure 3.2: Basic internal structure of each agent.

A more crucial question regarding this knowledge base is what does each agent have in it. In the general case, each agent should store a model of self, model of its environment, and model of its community in this knowledge base. The model of self would contain both declarative and procedural knowledge of the capabilities and limitations of an agent. The model of its environment would contain a sensor constructed world model that the agent would use in order to act sensibly in its environment. The model of its community would store models of acquaintance agents and different methods to coordinate with other agents. The question of what each type of agent actually stores in their knowledge base is deferred until the next section.

The third element of the internal structure of each agent is the communicating state-machine (CSM). Communicating state-machines (CSMs) define functional and temporal models of a

distributed system consisting of multiple asynchronous components [Yourdon, 1989; Casavant and Kuhl, 1990]. In our case, this model is finite and nonchanging. A CSM consists of a set of states connected together as a directed graph with labelled transitions. Typically, a CSM is represented pictorially. Our agents cope with certain basic errors by encoding contingencies as specific transitions in their CSM. The states and transitions of this CSM together with the knowledge in the knowledge base and the messages both sent and received by the agent determine the overall behavior of each agent.

3.4 Specific Aspects of Each Agent

This section describes aspects specific to the four types of agents. Each type of agent has its own unique CSM and the knowledge in the knowledge base varies from agent to agent. This section begins by explaining the notation that we use to present the CSMs of the agents before discussing each of the four agents in turn.

The states of a CSM are represented by ovals and the transitions between states are represented by solid directed arcs. The state with a double border is the initial state of a CSM. The transitions are labelled with a *condition/action* notation adopted with minor modifications from [Yourdon, 1989]. The condition part is above the horizontal line while the action part is below the horizontal line. Messages sent by a *sender agent* to the current agent take the form, *<sender agent>* » *<message>*. Messages sent from the current agent to a *destination agent* take the form, *<destination agent>* ! *<message>*. Multiple actions on a transition label are separated by the semi-colon character. The dashed directed arcs between an oval and a rectangle indicate dependencies between a state and the message queue of an agent or information specific to it.

3.4.1 Task Scheduling Agent

In the case of the TSA, we suppose that the directed graph specifying the precedence relations of the assembly and the associated task-level, assembly instructions are stored in the knowledge base before the TSA starts executing its CSM. The knowledge base is also used to keep track of which assembly tasks have and have not been completed. The TSA does not store a

model of its environment; however, its model of self simply stores the current state of its CSM and the model of its community stores the ASAs with whom it is acquainted.

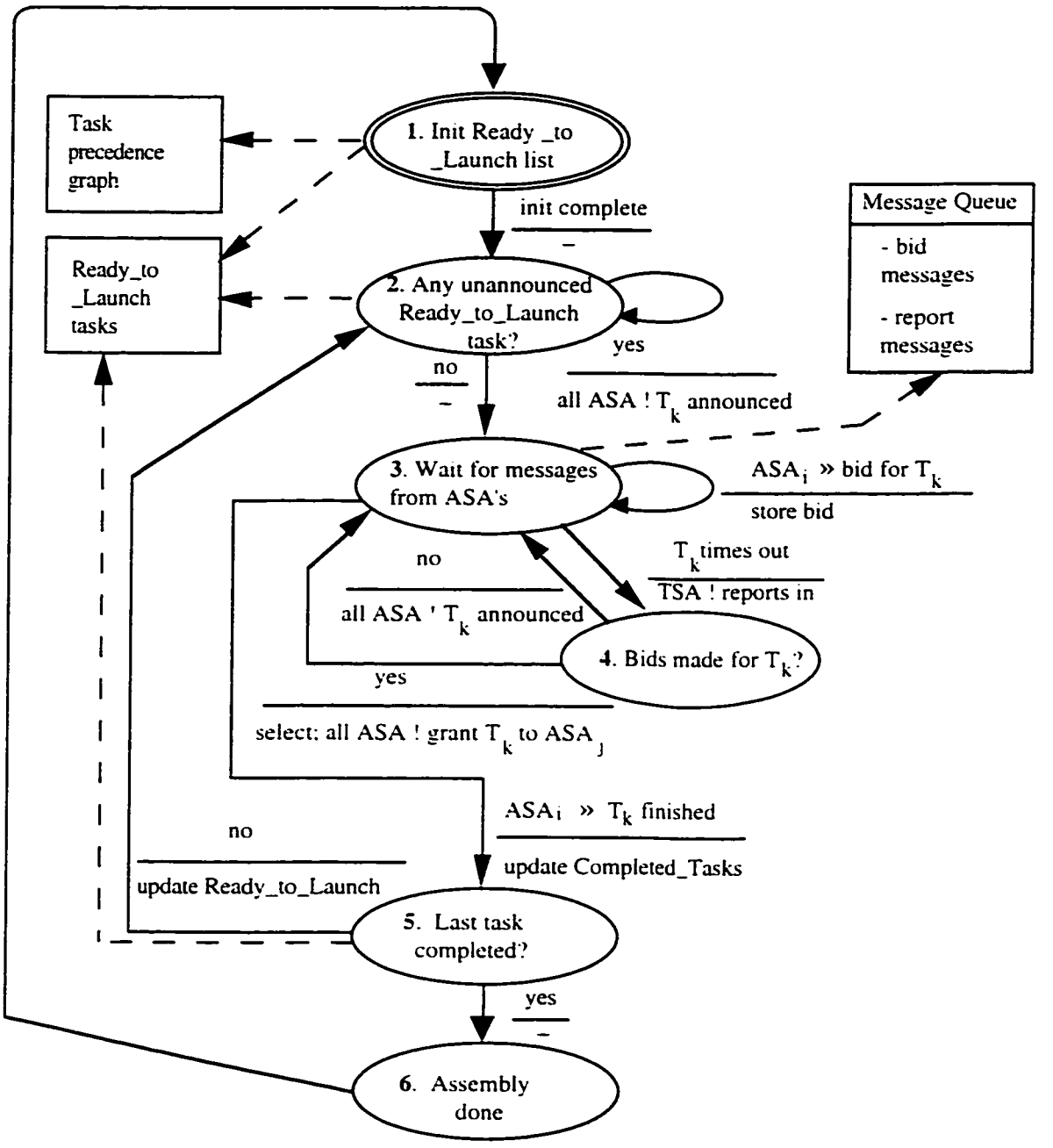


Figure 3.3: Communicating state-machine of the task scheduling agent.

Figure 3.3 depicts the CSM for the TSA. Based upon the task precedence graph, the TSA sets up the list of assembly tasks that are ready to be launched or announced to the ASAs in state

one. Having completed this initial setup, the TSA makes the transition to state two, where it loops until all assembly tasks in the ready to launch list have been announced before switching to state three. State three, wait for messages, defines the core functionality of the TSA. If a bid is received for any announced task from any ASA, the TSA simply stores this bid for later processing. When a deadline for submitting bids for a specific task passes, the TSA makes the transition to state four. If bids were submitted for this task, the TSA selects an ASA, sends a task granted announcement to all ASAs that bid for this task, and returns to state three. If bids were not submitted for this task, the TSA reannounces this task to all ASAs and returns to state three. If the TSA receives a task finished report from an ASA for a specific assembly operation, then it updates the list of completed tasks and makes the transition from state three to state five. If all the assembly tasks of the current precedence graph are not completed, then the TSA updates the ready to launch list and shifts back to state two. If all the assembly tasks of the current precedence graph were completed, then the assembly cell has completed a single cycle of production and can return to state one to start another cycle.

3.4.2 Assembling Agent

Figure 3.4 displays the CSM for an ASA. The ASA starts in state one, the idle state. If the ASA is in the idle state and it receives an assembly task announcement from the TSA that it is qualified to perform, then it bids for this task and makes the transition to state two, wait for grant. If the ASA is granted this assembly task then it announces the task, fetch part, to all acquaintance PPAs and makes the transition to state three; otherwise, it returns to the idle state. In state three, the ASA waits for and accepts bids from the PPAs and it makes the transition to state four when the deadline for submitting bids for the task, fetch part, is passed. If no bids were submitted, the ASA reannounces the task, fetch part, and returns to state three. If bids were submitted, then the ASA selects a PPA, sends a task granted announcement to the bidding PPAs, and makes the transition to state five, wait for part. When the selected PPA returns with the part and its pose or location, the ASA attempts to grasp the part during its transition to state six. If the part was not

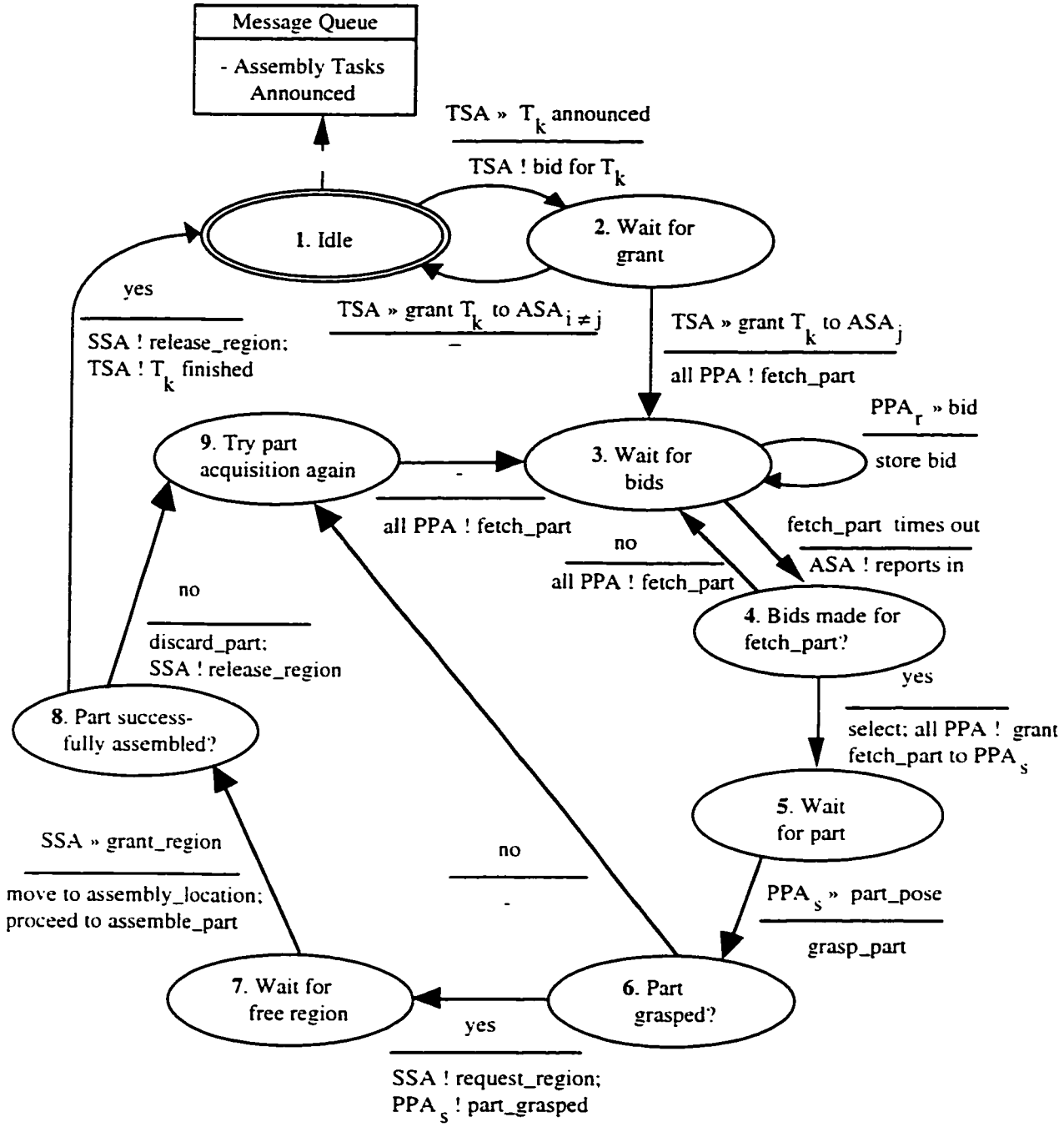


Figure 3.4: Communicating state-machine of an assembling agent.

successfully grasped, the ASA switches to state nine and tries to acquire the part again. If the part was successfully grasped, the ASA requests the region of space needed for the assembly from the SSA, informs the delivering PPA, if necessary, that the grasp was successful, and then shifts to

state seven, wait for free region. When the ASA receives exclusive access to this region of space from the SSA, it moves to the assembly location and proceeds to assemble the part during the transition to state eight. If the part was successfully assembled, the ASA releases this region of space by sending a message to the SSA, informs the TSA with a report message that the assigned task was finished, and returns to the idle state. If the part was not successfully assembled, then the ASA discards the part, releases this region of space, and tries again by switching to state nine. An important observation regarding this CSM is that states one and two encode the ASA acting as a contractor agent while states three and four encode the ASA acting as a manager agent in the contract net method.

We discuss the type of knowledge stored in the knowledge base of an ASA. Its model of self stores the declarative and procedural knowledge regarding its capabilities with respect to assembly operations. This model of self also stores the current state of its CSM and the current assembly task that is being performed. Regarding the model of its environment, we use a local model of the environment in order to carry out the actions that make up an assembly operation. The model of its community simply stores the PPAs with whom it is acquainted and a reference to the SSA.

3.4.3 Part Presentation Agent

Figure 3.5 portrays the CSM for a PPA. The PPA starts in the idle state, state one. When the PPA is in the idle state and it receives a fetch part task from an ASA that it can do, it submits a bid for this task and makes the transition to state two, wait for grant. If the PPA is not granted this task, it returns to the idle state. If the PPA is granted this task, it fetches the part and makes the transition to state three. State three is included to allow for a variation in the types of PPAs. If the PPA is a conveyor belt that does not need a pickup message, then it simply sends a message with the pose of the part to the manager ASA and returns to the idle state. If the PPA is an AGV that needs a pickup message, it sends a message with the pose of the part to the manager ASA and then waits for a pickup message in state four. Such an AGV makes the transition to the idle state only

when it receives a pickup message from the manager ASA. In this CSM, we presume that the PPA does not share any regions of space with other agents of the assembly cell.

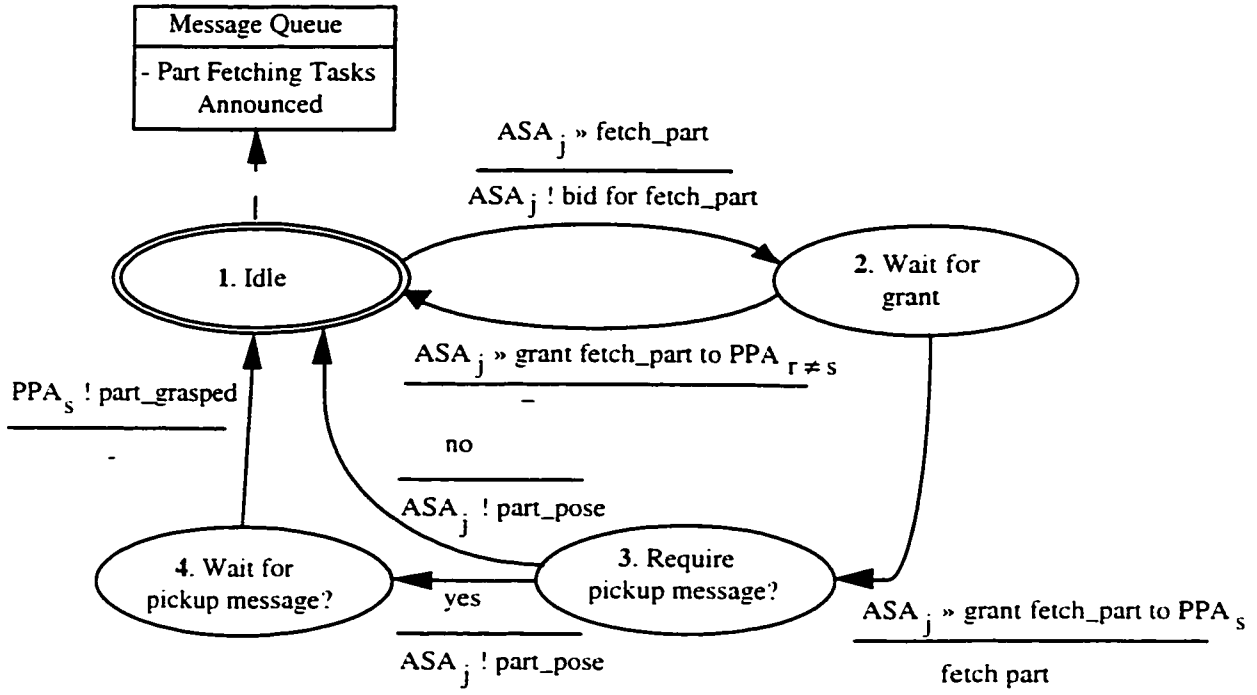


Figure 3.5: Communicating state-machine of a part presentation agent.

With respect to the knowledge stored in the knowledge base of a PPA, its model of self stores the declarative and procedural knowledge regarding its capabilities in delivering parts to the ASA(s). This model of self also stores the current state of its CSM and whether it requires a pickup message from an ASA after it has delivered a part. The model of its community simply stores a reference to the SSA.

3.4.4 Shared Space Agent

Figure 3.6 depicts the CSM for the SSA. Like the ASA and the PPA, the SSA starts in the idle state, state one. If the SSA receives a message requesting a specific region of space from an ASA or a PPA, it checks the availability of this region during its transition from state one to state two. If this region is not available, the SSA adds this request to the list of unsatisfied requests and returns to the idle state. If this region is available, the SSA grants the requesting ASA or PPA

exclusive access to this region, updates the availability of this region, and returns to the idle state. If the SSA receives a message releasing a specific region of space from an ASA or PPA, it updates the availability of this region and shifts to state three. If the list of unsatisfied requests is empty, then the SSA simply switches back to the idle state. If there are unsatisfied requests for regions of space, the SSA gets the next request from the list and makes the transition to state four. If the region of space for this former request is now available, then the SSA grants the requesting agent exclusive access, updates the availability of this region, and returns to state three; otherwise, it just directly returns to state three. The SSA alternates between state three and state four until all former unsatisfied requests have been checked before returning to the idle state.

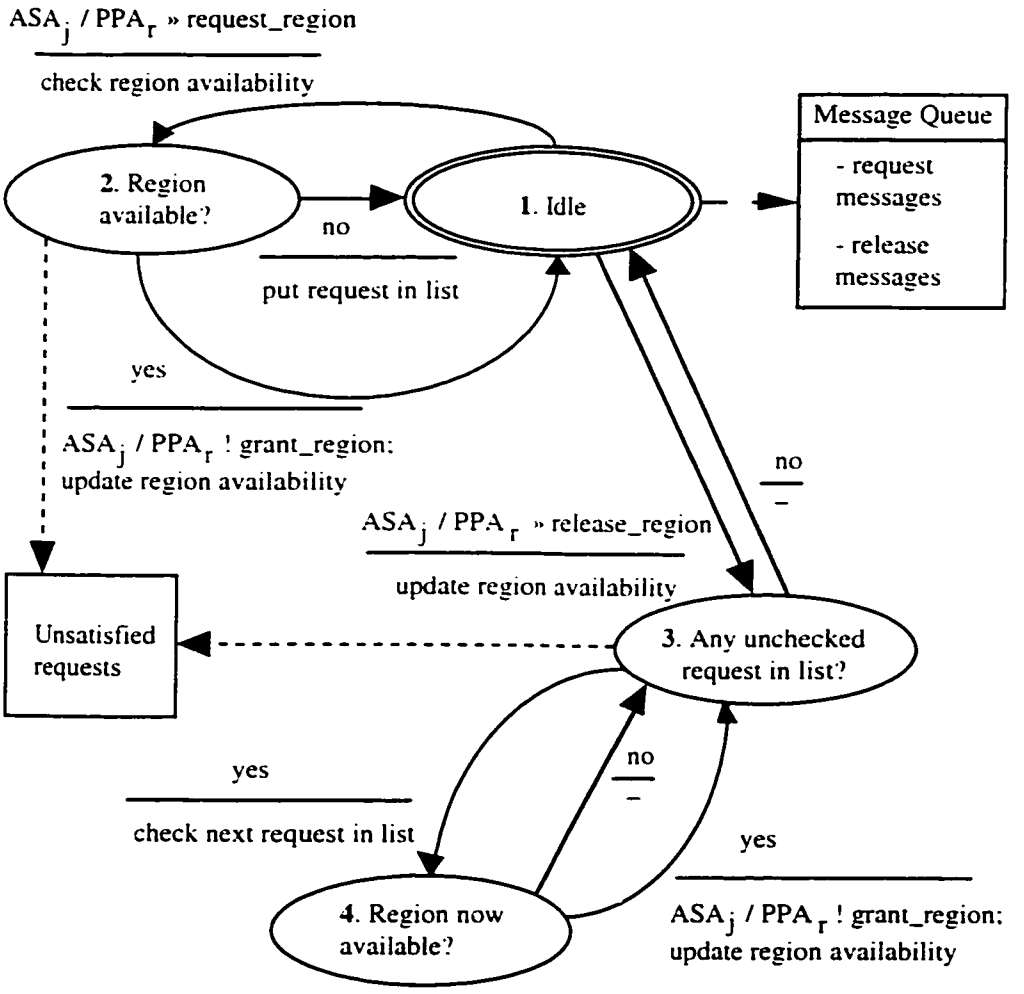


Figure 3.6: Communicating state-machine of the shared space agent.

The SSA uses the knowledge base to keep track of which regions of space are available and which are occupied. The SSA stores this information of these regions of space in the model of its environment. Its model of self simply stores the current state of its CSM and the model of its community stores references to all of the ASAs and PPAs that could make requests for regions of space.

3.5 Contents of Communication Messages

Although the types of messages exchanged between the four types of agents of our framework are determined for the most part by the contract net [Smith, 1979], the contents of these communication messages depends upon the application domain of robotic assembly. Before detailing the specific contents of these messages, this section discusses the preliminary issues of the format of the task-level, assembly instructions, the types of messages exchanged, and the standard contents of all messages.

3.5.1 Preliminary Issues

Recall that, the basic input to our MRAC is a directed graph of assembly operations specified as task-level instructions. The format of these task-level instructions provides the basic structure for the messages exchanged between the agents of our framework. We have adopted a format for these instructions from the field of robot programming [Lozano-Pérez, 1983]. Specifically, we define a task-level instruction as follows:

<ASSEMBLY_OPERATION> <PART> <ASSEMBLY_LOCATION> [OPTIONS]

where:

<ASSEMBLY_OPERATION>	refers to an assembly operation such as peg in hole, force fit, push and twist, crimp shut, etc. [Nevins and Whitney, 1978]
<PART>	the part with which to carry out the assembly operation
<ASSEMBLY_LOCATION>	the assembly location, often a fixture, to perform the assembly operation

[OPTIONS] options specific to this assembly operation

and this instruction was already used in figure 3.1 to indicate the primary contents of the message that the TSA sends to all of the ASAs.

Five types of messages are exchanged between the agents of the MRAC. Four of these types are directly related to the description of the contract net provided in subsection 3.2.2. These types of messages are identified as task announcement, bid, grant, and report. We use the report type of message to inform in three ways. First, a report message is used in the traditional sense of a contractor agent reporting to its manager agent. The second and third way have to do with the sending of information between the SSA and physical agents regarding the shared regions of space of the assembly cell. The next section describes the second and third ways in greater detail. The last type of message is a request for exclusive access to a specific region of space of the assembly cell.

Regarding the standard contents of all messages, we include four items that relate to the higher levels of communication. First, a symbolic reference to the sender of the message. Second, a symbolic reference to the recipient of the message. Third, a symbolic reference to the type of message. Fourth, a time stamp indicating the point at which this message was sent.

3.5.2 Specific Message Contents

According to [Smith, 1979], a task announcement message contains a task abstraction, an eligibility specification, a bid specification, and an expiration time. The task abstraction is a brief description of the task that needs to be done. The eligibility specification contains a list of criteria that the contractor agent must meet in order to bid for the task. The bid specification lists the information that the manager agent wants in the bid. The expiration time is the deadline for receiving bids for this task. The bid message from a contractor agent to the manager agent must contain the information requested in this bid specification. In the grant message, the manager agent grants the task to one of the bidding contracter agents and it also provides a complete task specification. The selected contractor agent typically uses a message of type report to inform the

manager agent of the final outcome of the task execution [Smith. 1979].

The messages exchanged between the TSA and the ASAs are described. Figure 3.7 displays the generic contents of a task announcement message sent from the TSA to the ASAs. The task abstraction is simply the task-level instruction described in the previous section without the options specific to this assembly operation. The eligibility specification lists the basic criteria that the ASA must meet in order to submit a bid. Each of these criteria is concerned with the basic capability of an ASA to perform the assembly operation. Through the bid specification, the TSA requests that any ASA submitting a bid must specify in their bid, the average time in seconds it takes them to execute this particular task. The expiration time stipulates the global time in seconds after which bids will not be accepted by the TSA for this task. The bid message from a qualified ASA must contain this average execution time. When the TSA grants the assembly operation to one of the bidding ASAs, it includes a task specification or in our case the complete task-level instruction. When an ASA finishes an assembly operation, it sends a report indicating which assembly operation was successfully completed.

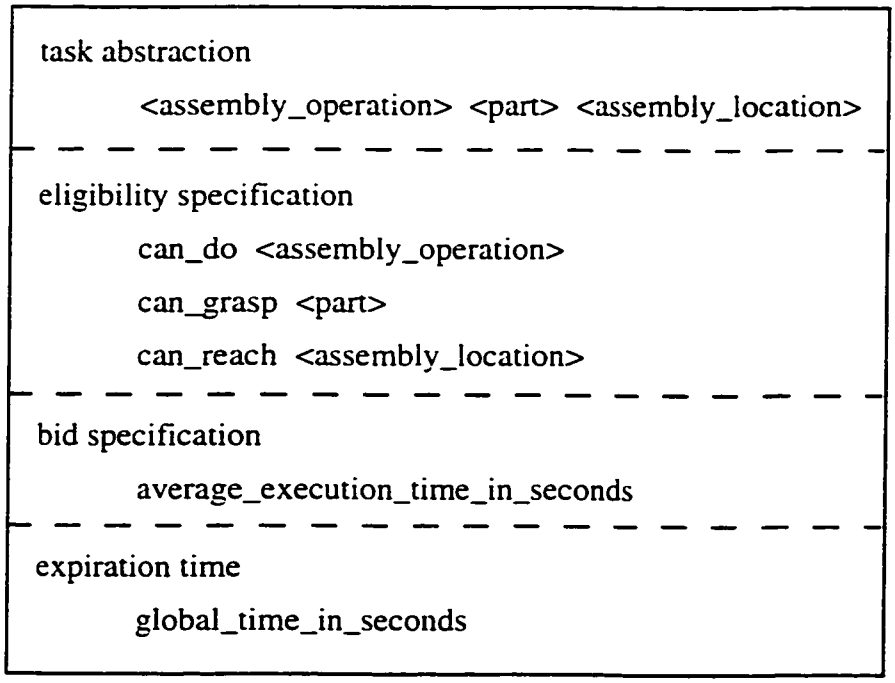


Figure 3.7: Task announcement message sent from the TSA to ASAs.

The messages exchanged between an ASA and the PPAs are described. Figure 3.8 shows the generic contents of a task announcement message sent from an ASA to the PPAs. In this case, the task abstraction is specified as *<fetch_part> <part>*. The part requested by an ASA is obviously the one needed to perform the assembly operation that was granted from the TSA. The two criteria in the eligibility specification ensure that the PPA meets the basic capabilities to deliver the part to the manager ASA. The bid specification of an ASA, like the bid specification of the TSA, compels a PPA to include in their bid the average time needed to execute the task, which in this case is delivering the required part. Since the task abstraction and task specification are the same for this task, the ASA sends a grant message with the task abstraction included to all of the bidding PPAs. When a PPA has fetched the required part, it sends a message of type report to its manager ASA with the pose, position and orientation, of the part included.

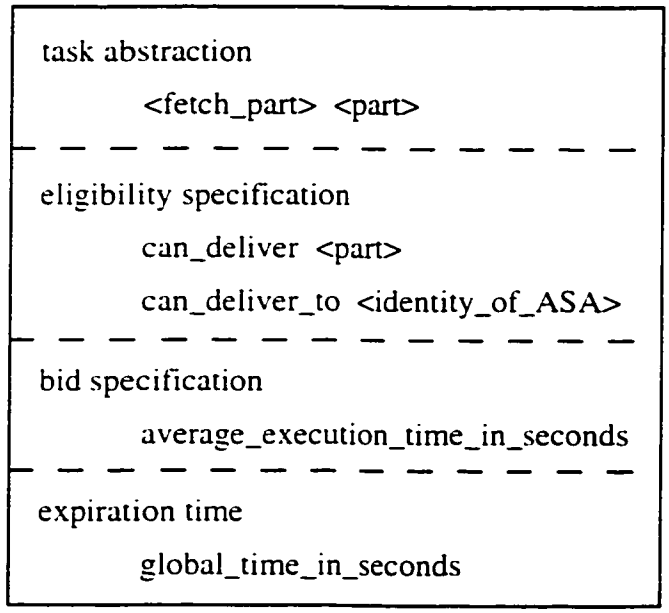


Figure 3.8: Task announcement message sent from an ASA to PPAs.

The exchange of messages between the SSA and an ASA or PPA that needs exclusive access to a shared region of space of the assembly cell is discussed. An ASA or PPA sends a request message to the SSA that indicates the region of space for which exclusive access is needed. When this region of space is available, the SSA sends a message of type report indicating to the

requesting agent that it may now safely physically proceed into this region of space. When an ASA or PPA is done with a shared region of space, it releases its hold on a shared region of space by sending a message of type report to the SSA indicating which region is now free.

3.6 Relationship with Existing Research

Figure 3.9 visually represents the relationship between the framework detailed in this chapter and existing research. We make several comments about this figure. First, the diagram is bounded by three sides in order to reiterate the point that our research takes place at the intersection of manufacturing, MASs, and robotic assembly. Second, the vertical axis of this diagram indicates the scope of the manufacturing system under consideration. The four levels or scopes are factory or enterprise, shop or line, cell, and component. Third, the horizontal axis of this diagram indicates the year that the article was published and it varies from 1984 to 1997. The scale of the horizontal axis is approximately linear. Fourth, the lines between the manufacturing levels are dashed to emphasize that the borders between levels are not precise or rigid. Research at the higher manufacturing levels often considers aspects of the lower levels and vice-versa. Fifth, the references in this diagram were placed at the manufacturing level that was deemed the most appropriate based upon our assessment. Sixth, due to space constraints, only a limited number of references are included in the diagram. These references were chosen for relevance, quality, and breadth.

We give an overview of the application of MASs to the factory or enterprise level. The bulk of this research was described in subsection 2.4.3. Many researchers have considered the application of the contract net [Smith, 1979] to the scheduling and control of a factory. Early applications of the contract net included [Shaw and Whinston, 1985; Parunak, 1985; Duffie et al., 1988]. More recent applications of the contract net included [Baker, 1992; Tchako et al., 1994; Tsukada and Shin, 1994]. An agent framework for complete integration of a manufacturing enterprise was presented in [Pan and Tenebaum, 1991]. Rizzi et al. [1997] presented an agile,

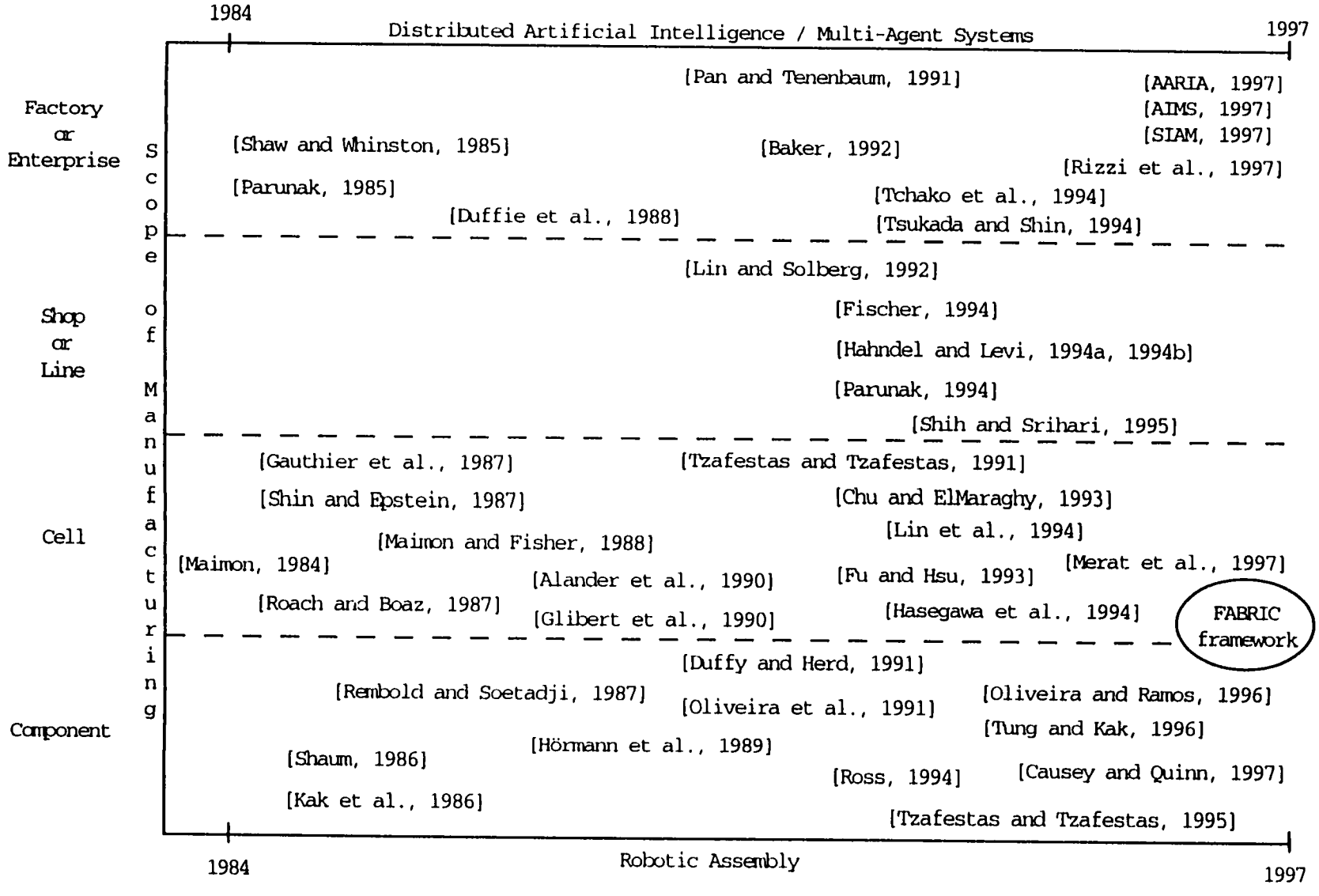


Figure 3.9: Visual representation of the relationship between our framework and existing research.

agent-based architecture for a minifactory that executed precise vertical assembly with four degrees of freedom. Current investigations into the application of agent-based systems to a manufacturing enterprise include [AARIA, 1997; AIMS, 1997; SIAM, 1997].

The application of MASs to the shop level, which were described in subsection 2.4.4, included the following researchers [Lin and Solberg, 1992; Fischer, 1994; Hahndel and Levi, 1994a, 1994b; Shih and Srihari, 1995]. A contract net based method was used to control the activities of the shop floor in two of these research projects [Lin and Solberg, 1992; Hahndel and Levi, 1994a, 1994b].

At the cell level, the diagram emphasizes issues of MRACs. Maimon [1984] proposed and studied a centralized approach to control the activities of a MRAC with manipulators operating in both loosely and tightly coupled configurations [Shin and Epstein, 1987]. Basic communication and coordination issues of multiple manipulators were described in [Gauthier et al., 1987; Shin and Epstein, 1987; Roach and Boaz, 1987]. Centralized approaches to schedule or plan the operation of a MRAC were also investigated [Glibert et al., 1990; Chu and ElMaraghy, 1993; Fu and Hsu, 1993]. Object-oriented control architectures for assembly cells were explored by several researchers [Maimon and Fisher, 1988; Alander et al., 1990; Lin et al., 1994; Merat et al., 1997]. A blackboard structure for an assembly cell was also studied [Tzafestas and Tzafestas, 1991]. Hasegawa et al. [1994] presented a holonic or agent-based architecture for planning and scheduling a manufacturing system that they illustrated with a simplified robotic assembly testbed. They used a contract net approach to determine the assembly sequence and the machine-types needed for the assembly. They also simulated a centralized, near optimal approach to scheduling [Hasegawa et al., 1994]. Research at the cell level summarized here was discussed in various subsections including 2.5.4, 2.5.5, and 2.7.3.

At the component level, two types of components are of interest. The first is the assembly robot. Several researchers have proposed and developed agent-based control architectures for assembly robots [Hörmann, 1989; Duffy and Herd, 1991; Oliveira et al., 1991; Tzafestas and

Tzafestas, 1995; Oliveira and Ramos, 1996]. Each of these architectures were already described in subsection 2.7.1. The research of [Kak et al., 1986; Tung and Kak, 1996] presented and implemented a sophisticated and integrated framework for synthesizing a manipulation plan for a single assembly robot. The reference [Rembold and Soetadji, 1987] was included in the diagram because it was an early reference about an actual assembly robot with two physical manipulators that has received considerable study in the last decade. The second component is a part presentation system [Shaum, 1986] or a flexible parts feeder [Ross, 1994] and this component continues to be studied and implemented [Causey and Quinn, 1997].

As shown in the diagram, the FABRIC framework is an investigation at the cell and component levels with a greater emphasis on the cell level. This framework or scheme distinguishes itself from the existing research in several ways. First, in the general sense, FABRIC is an original cooperative, heterarchic, agent-based architecture for managing MRACs. Second, we distinguish ourselves by focusing on the inter-agent communication and macro-level aspects of this agent-based MRAC. Third, although many applications of the contract net to the factory and shop levels of a manufacturing system have been reported, FABRIC uniquely applies the contract net to the operation of a MRAC. In particular, our application focuses on the task-level scheduling of the assembly operations of a MRAC. Fourth, the internal structure of our agents is distinct with respect to the communicating state-machine. Fifth, we uniquely decompose a MRAC into agents. One result of our decomposition is the encapsulation of the capabilities of an assembly robot at the level of task-level instructions within the concept of an ASA. This encapsulation has implications regarding the task-level programming of a robotic manipulator that will be discussed in chapter five of this dissertation. Moreover, the making of a part presentation system into an agent is also distinct.

In closing, we mention that the state of the art of industrial robotic assembly [Rampersad, 1995a] has not yet considered modern agent-based solutions as a means to integrate, operate, and maintain assembly systems.

Chapter 4

Software Implementation of the Agents

This chapter describes how the agents described in the previous chapter were implemented in software. In this software implementation, both the generic aspects of each agent and the specific aspects of the four types of agents that cooperate in a FABRIC system were programmed.

This chapter is partitioned into five sections. The first section discusses issues about the software technology that was used in this implementation. The second section gives an overview of the software implementation of our generic concept of an agent. The third section details the implementation of a frame-based knowledge base. The fourth section elaborates on the software implementation of the four specific types of agents. The last section summarizes the contents of this chapter.

4.1 Software Technology

We chose to implement the agents of FABRIC using object-oriented software technology. This section gives a very brief introduction to this technology and also summarizes the object-oriented notation that we will use to detail our implementation.

4.1.1 Object-Oriented Software Technology

One of the key questions with respect to using an object-oriented software technology is selecting an object-oriented programming language. Although there are many ways to categorize object-oriented programming languages, we group them into two broad categories. This dichotomy emphasizes general characteristics of these languages and historically can be traced to

differences in the perspectives of software engineering (SE) and artificial intelligence (AI) [Masini et al., 1991].

Software Engineering Perspective	Artificial Intelligence Perspective
compiled environment	interpreted environment
static typing	dynamic typing
manual memory management	automatic garbage collection
efficient	flexible
slow prototyping	fast prototyping
industrial use	research use

Table 4.1: Typical characteristics of the two broad categories of object-oriented languages.

Table 4.1 displays the typical characteristics of languages from these two categories. This table was constructed by using two sources [Masini et al., 1991; Khoshafian and Abnous, 1995]. Typically, environments where the source code of a program is compiled into machine executable form are more common in SE, while environments where this source code is interpreted and then executed by the computer are more common in AI. Static typing means that the structure and behavior of classes, which are organized in an inheritance graph, are fixed at compile time and therefore cannot change during program execution. Dynamic typing means that this inheritance graph is modifiable during program execution and therefore new classes and new methods can be added at run-time. Typically, memory management is the responsibility of the programmer in SE, whereas the technique of garbage collection automatically manages the memory of the computer system in AI. Generally speaking, an interpreted environment has the properties of being more flexible but less efficient and of enabling faster prototyping than a compiled environment. Languages of SE have been employed more in industrial environments while languages of AI have

received more use in research environments. Examples of SE, object-oriented programming languages include Simula and C++ and examples of AI, object-oriented programming languages include Smalltalk and CLOS (Common Lisp Object System). This dichotomy is by no means rigid. In fact, the introduction of the Java object-oriented programming language [Gosling and McGilton, 1995] has even further blurred the distinction between these two categories.

We selected the object-oriented programming language, C++ [Ellis and Stroustrup, 1990], for three reasons: efficiency, more common in industry, and building upon an existing system programmed in C.

With a slant towards C++, we summarize the basic features of an object-oriented programming language. A class defines both the structure and behaviour of a family of objects. The structure defines the content, variables, or data structures of the class while the behaviour specifies the methods, procedures, or functions that work with the structure. Placing the contents and methods into a single class in order to control both the access to and the operations performed on the contents is known as the concept of encapsulation. An object is a particular instance of a class. Objects communicate with each other by sending messages. A template class defines a family of classes that vary with one or more parameters.

Classes can be organized into inheritance graphs where classes inherit the structure and behaviour of all classes that are traversed from its location to the class at the root of the graph. Specific terms are used to describe two classes that are adjacent or directly linked together in this graph. The class that inherits in this link is known as the derived class, subclass, or child class while the bequeathing class is known as the base class, superclass, or parent class respectively. We use the terms base class, derived class, and parent class in our explanations. The case of single inheritance occurs when each class in an inheritance graph only inherits from a single base class and this graph is also referred to as an inheritance hierarchy.

There are three levels of access control to the variables and methods of a class. We present regular access before discussing an exception. Variables and methods declared private are only

accessible by direct instances of the declaring class. Protected variables and methods are accessible by the declaring class and all derived classes of this class. Public variables and methods are accessible by all. A friend declaration allows the programmer to make exceptions to these levels of access control. If a class declares a function or another class as a friend, then it means that this function or class can access all private and protected, variables and methods of the declaring class.

We introduce the concepts of polymorphism, operator overloading, and run-time type identification. Polymorphism refers to the fact that for a method declared virtual, the actual method that is called depends upon the type of object at run-time. For instance, a base class defines a general approach to a problem with a set of virtual methods that each derived class can override or implement in its own way. If a pointer of this base class refers to an instance of a derived class and it calls a virtual method that the derived class has overridden, then the method of the derived class is executed. Operator overloading permits functions with different arguments to have the same name and allows classes to redefine the meaning of basic operators, such as =, +, *, -, ( ), [ ], or &, within their own context. Run-time type identification means that at run-time an object can return the identity of the class from which it was instantiated. This concept allows the programmer to safely cast or coerce the type of a class during program execution.

4.2.2 Object-Oriented Notation

Object-oriented notations specify visual methods to represent both the static and dynamic aspects of an object-oriented system. These notations are customarily associated with a particular technique for object-oriented analysis and design and many researchers have put forth such techniques [Hutt, 1994a, 1994b]. We decided to use the notation associated with the object-oriented analysis and design technique known as the Unified Modeling Language (UML) [RATIONAL, 1997]. The UML technique unified two well known methods of object-oriented analysis and design: the Booch method [Booch, 1994] and the OMT (Object Modeling Technique) method [Rumbaugh, 1995].

Although these notations can represent both the static and dynamic aspects of an object-

oriented system, this subsection only describes the visual methods provided by UML to represent the static aspects. In other words, the remainder of this chapter presents the static structure and behaviour of the classes that implemented the FABRIC agents. This decision to detail the static aspects of the software implementation was considered justified because the dynamic aspects of our agents are reasonably straightforward and these aspects were already described in subsection 3.4.

Figure 4.1 displays the object-oriented notation that we have adopted from the UML [RATIONAL, 1997] and will use to explain the software implementation of the FABRIC agents. As much as possible, this figure uses the terms that were introduced in the previous subsection.

The complete visual representation of a class consists of a rectangle with three compartments. From top to bottom, the compartments hold the name of the class, the contents or variables of the class, and the methods of the class respectively. The minimal representation of a class is to simply identify its name in a rectangle. The three symbols, +, #, and -, which are shown at the bottom of figure 4.1, are used to indicate the access control for specific variables or methods of a class. A template class is visually represented as a rectangle with a smaller dashed rectangle superimposed on the upper right hand corner. The rectangle holds the name of the template class, while the dashed rectangle contains the formal parameters of the template class. A specific class created from a template class is designated by:

Template Name<specific_parameter(1), specific_parameter(2), ..., specific_parameter(n)>

A package is a grouping of model elements such as a set of classes that implement some particular functionality. A package is visually represented as a rectangle with a small rectangle attached to its upper left corner. If the contents of the package are shown then its name may be indicated in the small rectangle; otherwise, its name is indicated in the large rectangle as shown in figure 4.1. Although stereotypes provide a mechanism to introduce new modeling elements and thereby extend UML, we only use them to express certain predefined stereotypes, whose names are

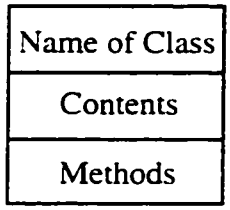
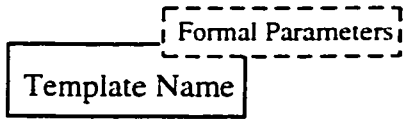
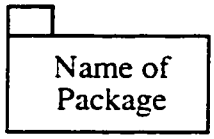
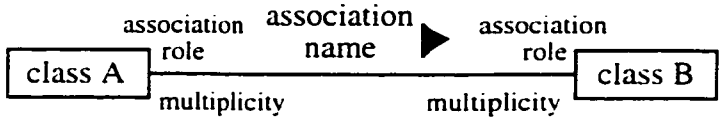
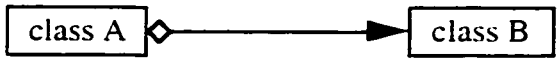
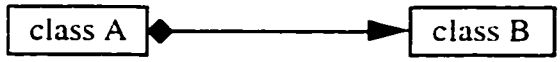
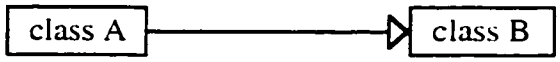
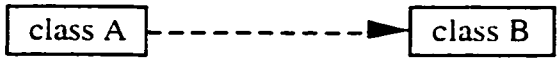
Object Oriented Notation adopted from the Unified Modeling Language	
Visual Representation	Object-Oriented Meaning
	class
	template class
	package
	binary association
	contains association
	composed of association
	generalization
	dependency
<< name of stereotype >>	stereotype
+	public access control
#	protected access control
-	private access control

Figure 4.1: Object-oriented notation adopted from the Unified Modeling Language.

«constructor», «friend», and «binds». A constructor defines how an instance of a class is created. The names of stereotypes are placed between double angle brackets or more precisely guillemets [RATIONAL, 1997].

We now describe different relationships between two generic classes that are indicated as *class A* and *class B* in figure 4.1. In this figure, the visual representation of a binary association is shown as a solid line with complete labelling. The name of the association is optionally indicated above or below the middle of this line and this name may have a black triangle beside it that indicates the direction of the association. In this figure, the binary association reads "class A association name class B". The association role beside each class optionally specifies the role that each class plays in an association. The multiplicity beside each class optionally indicates the range of allowable cardinalities for each class of an association. The * symbol is used to indicate an unlimited cardinality. The next two associations are displayed without any labelling. If an association is shown as a solid line with a hollow diamond at one end and an arrow at the other end, then it indicates a contains association. If the hollow diamond is replaced by a solid black diamond, then it indicates a composed of association. In figure 4.1, without indicating the multiplicity these associations read "class A contains class B" and "class A is composed of class B" respectively. A solid line with a hollow triangle at one end indicates the relationship of generalization. The triangle points from the specific class or derived class to the general class or base class. A dashed line with a single arrow indicates a dependency between two classes. In this figure, the dependency reads "class A depends on class B". The dependency relationship in combination with stereotypes can be used to indicate a friend declaration between two classes and a binding between a template class and a specific class created from this template class with specific parameters.

4.2 Software Implementation of a Generic Agent

Recall that, the three generic elements of all our agents are a messaging system, a frame-based knowledge base, and a communicating state-machine (CSM). The class *Agent* directly

incorporates the first two elements plus it provides methods to support the CSM, to cope with many aspects of the contract net, and to add acquaintance agents.

Figure 4.2 displays the essential structure and behaviour of the *Agent* class using the UML notation. We make a few more points about the notation used in this figure. The notation used for each variable of a class is the symbol of access control, the name of the variable, and an optional colon followed by its specific type. The notation used for each method of a class is as follows: symbol of access control, name of the method, parameters in parentheses, and a colon followed by its return type. If the return type is not specified it is void in C++. The notation that we used for the parameter(s) of a method is either its name or its specific type depending upon which is more informative. The question mark is used to indicate a single parameter of a method that is not specified for the sake of brevity. If the extension *_Ptr* is appended to the name of a basic type, it indicates a pointer to this basic type.

The relationship of the class *Agent* with other classes is first described. The class *Agent* declares the class *Simulator* as a friend. This declaration is necessary in order to make it easy to simulate the behaviour of a group of FABRIC agents. The class *Simulator* and the simulations/experiments generated with it will be discussed further in chapters five and six. The class *Agent* is composed of a frame-based knowledge base that we have represented as a package in this figure. The set of classes that define this package are detailed in the next section. The template class *Basic_Queue* defines a generic simple queue that an agent can operate on with the public methods displayed in figure 4.2. The class *Agent* is also composed of the template class *Basic_Queue* with the parameter *Storage_Type* set to a pointer to a class of type *Message_Frame*. This queue stores the messages that an agent receives and as the name *Message_Frame* implies, the agents exchange messages in the form of frames with each other. These message frames are explained in greater detail in subsection 4.4.1.

Besides the major elements of a knowledge base and a message queue, the class *Agent* also privately stores a unique name for each agent and the name of the direct parent or class from which

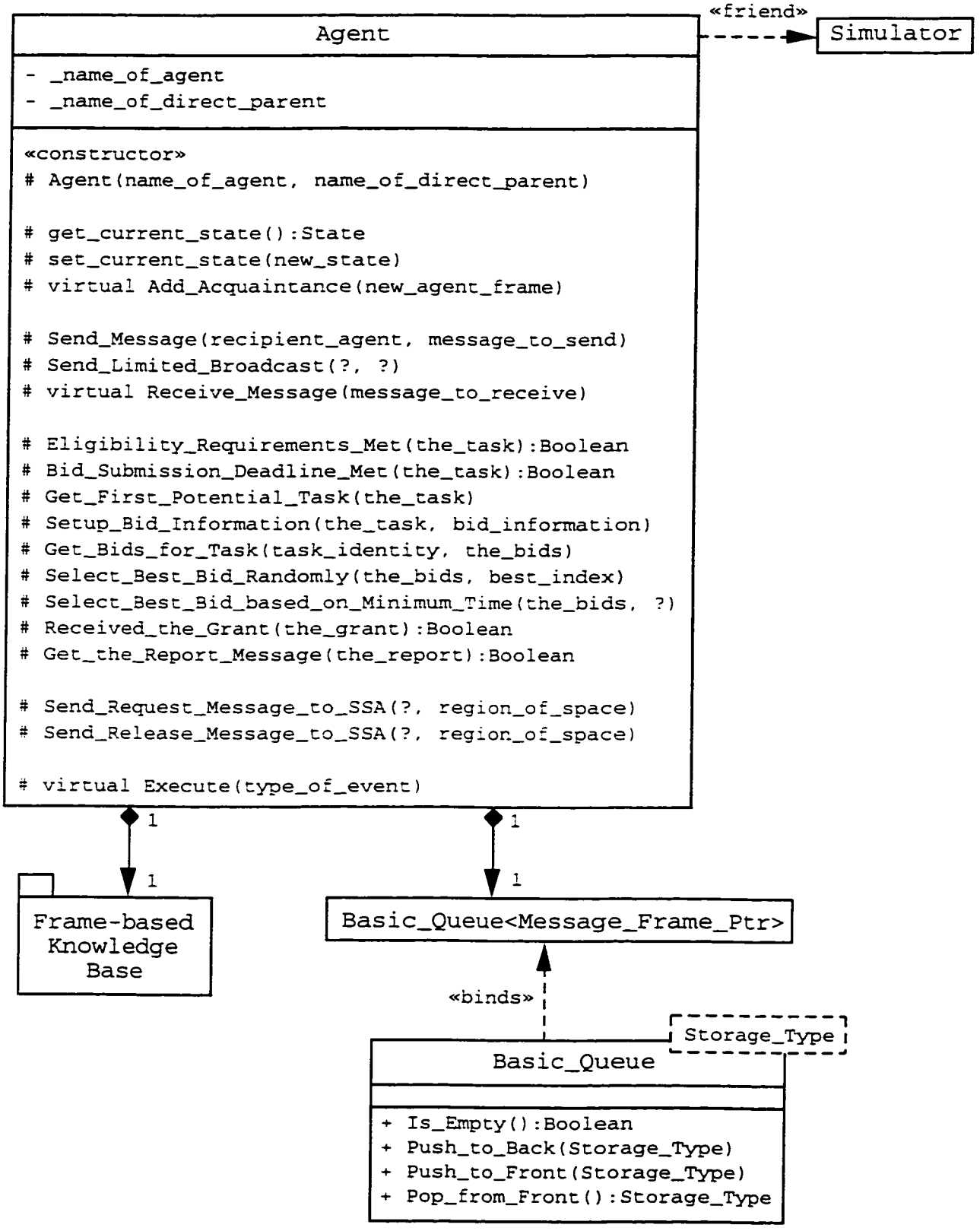


Figure 4.2: UML notation for the class *Agent*.

an agent was instantiated. In our implementation, the convention of beginning all variables of a class with an underscore was adopted.

All constructors and methods presented in figure 4.2 are protected and therefore only accessible by classes derived from the class *Agent*. The protected constructor is used by derived agents to initialize their name and the name of their direct parent. We have divided the remainder of the methods listed in the bottom compartment into five groupings.

The first group of methods make it easier to perform certain actions with the knowledge base of an agent. It is easier to set and get the current state of the CSM of an agent with the methods named: *get_current_state* and *set_current_state*. The method entitled, *Add_Acquaintance*, provides a default method for the derived agents to add an acquaintance agent to their knowledge base. Since this method is declared virtual, the derived agents can override how an acquaintance agent is added to their knowledge base.

The second group of methods defines the functionality available for agents to exchange messages with each other. The method *Send_Message* sends the given message to the specified agent while the method *Send_Limited_Broadcast* sends a given message to all acquaintance agents of a particular type. The virtual method *Receive_Message* implements a default first in, first out queue that a derived agent typically overrides. This virtual method is also important in properly simulating the group behaviour of the FABRIC agents.

The third group of methods implements many of the basic steps involved during contract net negotiation between agents. We only explain two of these methods for exemplary purposes. The method *Bid_Submission_Deadline_Met* takes as input the announced task of interest to an agent, determines whether the deadline for submitting bids has passed, and then returns a *Boolean* indicating its evaluation. Having received bids for a previously announced task, an agent can select one of the bids randomly by calling the method *Select_Best_Bid_Randomly*.

The fourth group of methods make it easy for an ASA or PPA to request or release a region of space from the SSA. The last group declares the single method *Execute* with the single

parameter *type_of_event*. The parameter *type_of_event* refers to a transition in the CSM of an agent. This method specifies how the core execution of an agent in terms of the interaction between events in its CSM and its knowledge base is to be implemented. This method is declared virtual since each derived agent must override this method to define its own independent execution.

This subsection closes by mentioning that Appendix A contains a complete listing of the header file and source file for the class *Agent*.

4.3 Software Implementation of a Frame-based Knowledge Base

According to [Karp 1993], in the last twenty years over fifty frame-based knowledge representation systems have been implemented. The bulk of these systems were implemented using the programming language Lisp or one of its derivatives, Common Lisp, Scheme, CLOS [Karp, 1993; Kantrowitz, 1994]. Our frame-based system is one of the few implemented using the object-oriented programming language, C++ [Ellis and Stroustrup, 1990].

We repeat for the sake of convenience the basic synopsis of a frame-based knowledge base that was provided in section 3.3. A frame is a data structure that typically represents a single object, a class of related objects, or a general concept [Karp, 1993]. Each frame has the property of being named and consists of a set of slots. Each slot describes a single attribute of what its frame represents. The properties of slots are called facets. Typical slot facets are name, value, datatype, and value restriction [Karp, 1993]. Frames are typically organized into hierarchies and therefore, benefit from the concept of inheritance. A collection of frames in one or more inheritance hierarchies is a knowledge base [Karp, 1993].

This section presents the internal details of the software implementation of this frame-based system or package using the UML notation. This section is divided into two subsections. The first subsection provides an overview of the whole system in which the software implementation of the knowledge base and the frames is described. The second subsection details the software implementation of the slots.

4.3.1 Overview of the Implementation

Figure 4.3 provides a visual overview of the object-oriented software implementation of a frame-based knowledge base. In this figure, classes with all three compartments or complete classes only display the essentials of their structure and behaviour, while classes with only a single compartment containing their name are compressed due to limitations of space.

We begin by explaining the template class *Hash_Table*. A hash table defines an efficient mechanism to store and retrieve elements that typically operates in constant time [Sedgewick, 1983; Musser, 1995]. Each element has a unique key of identification. Each entry in the hash table consists of an element plus its key. This key is input to a hash function that returns an integer, which is evenly distributed over a prespecified range of values. Typically, each integer identifies a unique, singly linked list that stores all of the entries whose keys have generated its integer. The template class *Hash_Table* has two parameters, *Key* and *Base_Element*. The parameter *Base_Element* is expected to be a base class that outlines the core structure and behaviour for its derived classes. This base class is required to specify a virtual method *Clone* that allows the hash table to make a copy of it or any derived class. Two basic variables define the contents of the template class *Hash_Table*. The variable *_number_of_buckets* specifies the range of integers starting from zero that the hash function will return. The variable *_the_entries* is an array of vectors that store entries of the hash table. These vectors used the implementation of the Standard Template Library [Stepanov and Lee, 1995]. Based upon the key, the three methods of this template class add, get, and remove elements from the hash table.

As shown in figure 4.3, two classes were created from the template class *Hash_Table*. In both cases, the key of the hash table was the class *CString*, a simple class for manipulating strings. The first of these classes, which has the second parameter set to the class *Base_Frame*, is contained by the class *Knowledge_Base*. The class *Knowledge_Base* uses this hash table class to add, get, and remove frames from a knowledge base. The method *Get_All_Frames* retrieves all frames that were created from a particular parent.

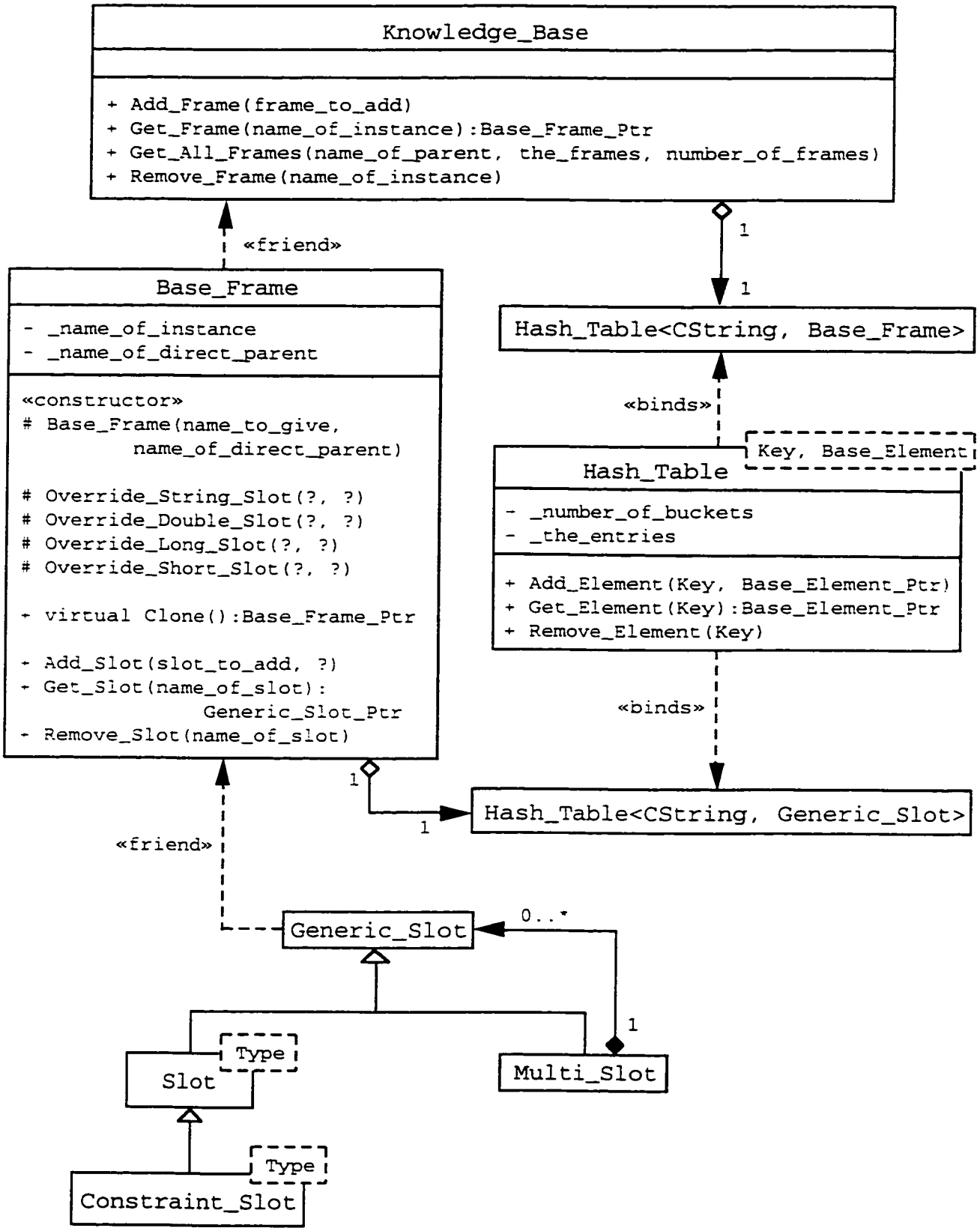


Figure 4.3: The knowledge base package represented in UML notation.

The second of these classes, which has the second parameter set to the class *Generic_Slot*, is contained by the class *Base_Frame*. The class *Base_Frame* defines the core structure and behaviour of the frames of our knowledge base. The class *Base_Frame* and its derived classes use this hash table class to add, get, and remove slots from their representation. Also note that the virtual method *Clone* was defined for this class in order to satisfy the requirements of the template class *Hash_Table*.

The basic variables of the class *Base_Frame* are its name and the name of its direct parent or class from which it was instantiated. The protected constructor expects these two variables as input and it is meant to be used by all derived classes. Four convenience methods are defined in order to make it easy for derived classes or frames to override the value of slots of four basic types. The class *Base_Frame* declares the class *Knowledge_Base* as a friend since the class *Knowledge_Base* needs access to certain private aspects of this class.

The class *Generic_Slot* outlines the core structure and behaviour of an attribute of a frame. The class *Generic_Slot* declares the class *Base_Frame* as a friend in order to allow the class *Base_Frame* complete access to its structure and behaviour. The class *Multi_Slot* is derived from the class *Generic_Slot* and it is recursively composed of zero or more *Generic_Slot(s)*. The template classes *Slot* and *Constraint_Slot* define the structure and behaviour for complete families of classes and slots.

An agent interacts with this frame-based knowledge package through the public methods of the class *Knowledge_Base*, the class *Base_Frame*, and the slot classes.

4.3.2 Implementation of the Slot Classes

This subsection expands on the internal details of the four slot classes. In particular, we use the UML notation to explain the essential structure and behaviour of these classes.

Figure 4.4 displays the essential structure and behaviour of the class *Generic_Slot*. The variables of the class *Generic_Slot* define four basic facets. First, all slots have a name. Second, a

slot can be read only. Third, if a slot is overrideable then a frame that inherits such a slot can change its value. Fourth, if a slot is specified as "no inherit" then this slot is only created for direct instances of a frame and is not inherited by derived frames. The protected constructor expects all four facets as input and it is used by all derived classes. The virtual method *Clone* was defined for this class in order to satisfy the requirements of the template class *Hash_Table*. Each derived class must override this method in order to make a proper copy of itself.

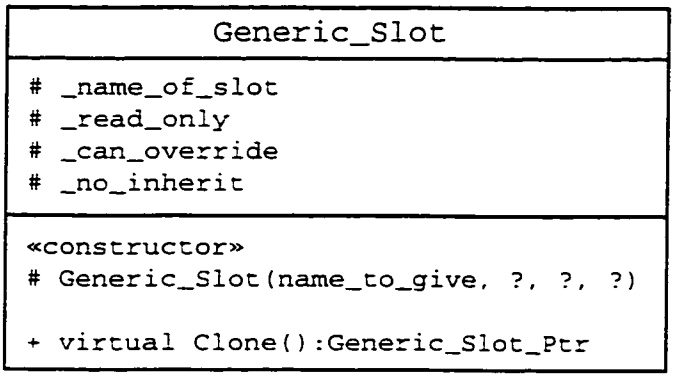


Figure 4.4: Essential structure and behaviour of the class *Generic_Slot*.

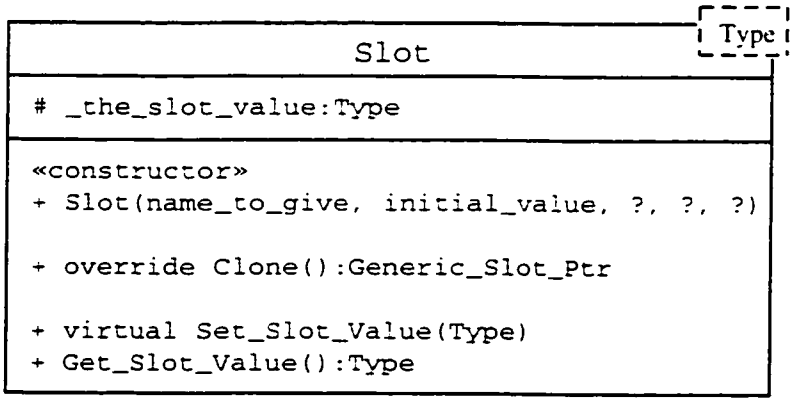


Figure 4.5: Essential structure and behaviour of the template class *Slot*.

Figure 4.5 depicts the essential structure and behaviour of the template class *Slot*. The template class *Slot* is derived from *Generic_Slot* and it adds the facet that a slot only store values of a specific type. With its definition, the programmer can easily create a slot of any type. The variable *_the_slot_value* stores the actual value specified by the parameter *Type* of this template

class. The template class *Slot* places no restrictions on the values of its datatypes. In addition to the information expected by the class *Generic_Slot*, the public constructor of this class also expects an initial value. This template class overrides the method to clone itself and return a pointer to the class *Generic_Slot*. Two access methods are defined by this template class. The method *Set_Slot_Value* allows the user to change the value of a slot and it is declared virtual in order to allow derived classes to override how a slot's value is changed. The method *Get_Slot_Value* simply returns the current value of a slot. We have used several different types of slots including: *Boolean, CSkill_Ptr, CString, double, RPY_Location, short, State*.

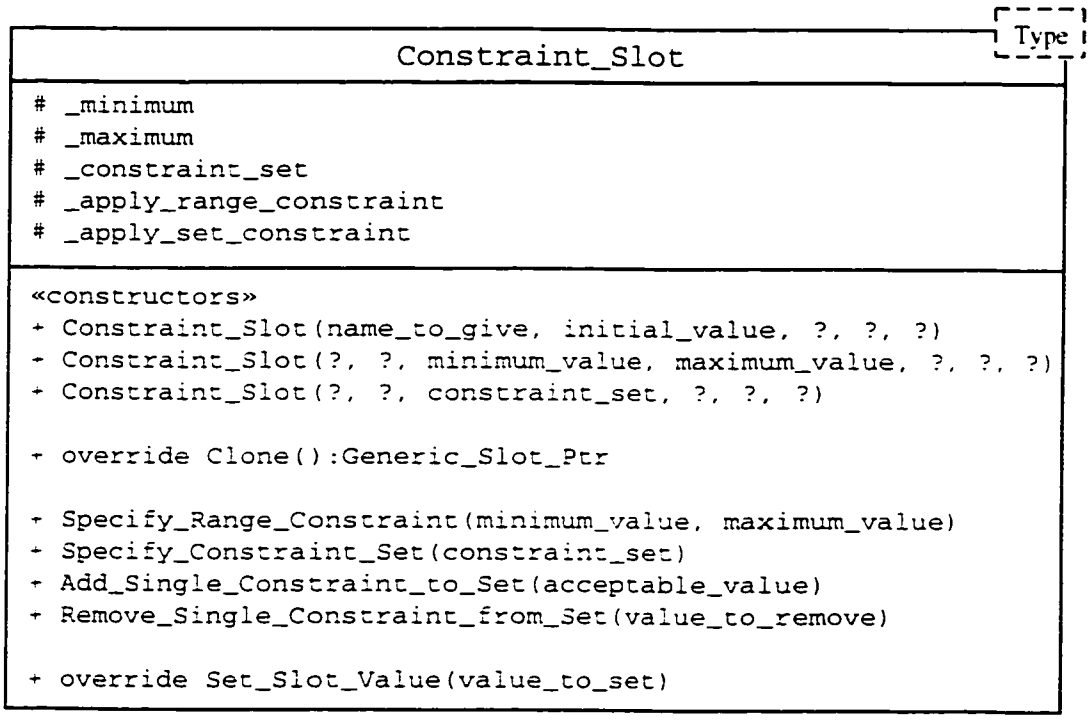


Figure 4.6: Essential structure and behaviour of the template class *Constraint_Slot*.

The template class *Constraint_Slot* simply adds a facet to restrict the values of its datatype inherited from a specific *Slot* class. Figure 4.6 shows the essential structure and behaviour of this template class. Two approaches to restricting the values of a datatype were implemented. The user can restrict the datatype to a range of values from a minimum to a maximum or the user can constrain the datatype to a specific set of values. Only one of these restrictions can be active at a

time and the two variables, *_apply_range_constraint* and *_apply_set_constraint*, reflect this condition. From top to bottom, the three constructors allow the user to set up a slot with an initial value and no initial constraints, a range constraint, or a set constraint. This template class overrides the *Clone* and *Set_Slot_Value* method to ensure that they properly execute in its case. The remainder of the methods implemented in this template class permit the constraints of a slot to be dynamically changed.

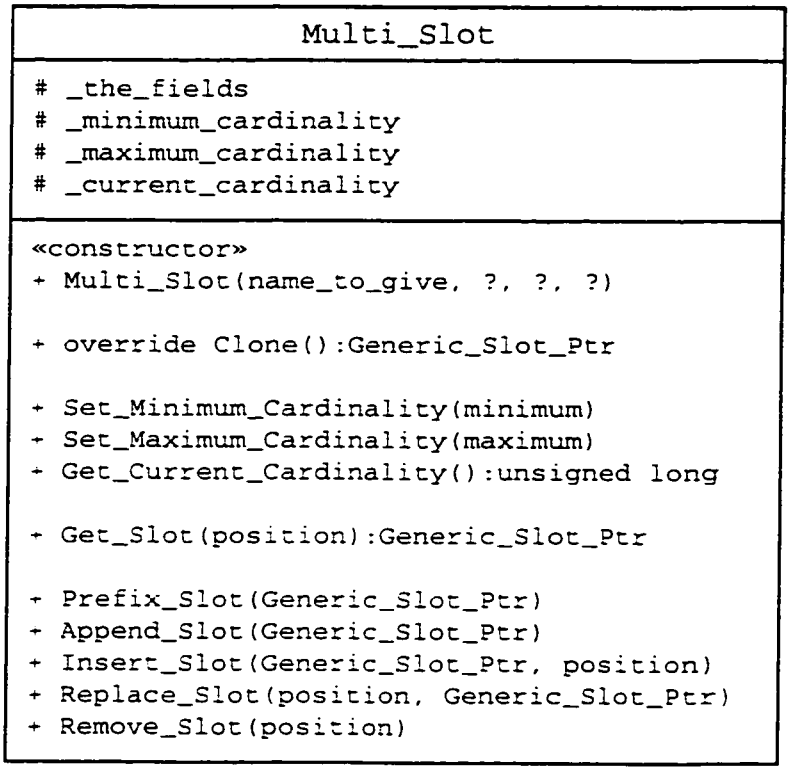


Figure 4.7: Essential structure and behaviour of the class *Multi_Slot*.

Lastly, the class *Multi_Slot*, which is derived from the class *Generic_Slot* and is depicted in figure 4.7, allows one to store, retrieve, and remove slots from a list. This class defines four variables. The variable *_the_fields* stores the list of slots as a vector from the Standard Template Library [Stepanov and Lee, 1995]. The three remaining variables store the cardinality of this list. That is, a user can restrict the minimum and maximum number of items in this list. The public constructor requires the same information as the protected constructor for the class *Generic_Slot*.

The virtual method *Clone* had to be overridden in order that the class *Multi_Slot* properly copied all of its slots and cardinality information. The rest of methods of the class *Multi_Slot* reflect its purpose. In other words, methods to change the cardinality and methods to get, prefix, append, insert, replace, and remove slots from the current list of slots were implemented.

4.4 Software Implementation of the Specific Agents

This section summarizes how the specific agents of FABRIC were implemented in software using the object-oriented programming language, C++. Just a reminder that section 3.4 described how the four types of FABRIC agents differ from each other. Like the previous section, this section is also partitioned into two subsections. The first subsection discusses general and common aspects of the software implementation of these agents while the second subsection details specific aspects of this implementation.

4.4.1 General and Common Aspects

Figure 4.8 shows the class hierarchy of the agent classes. All of the classes are shown in a compressed format. The four classes derived from the class *Agent* implement the four types of FABRIC agents. These derived classes are distinguished from the base class in three key ways. First, the CSM that defines the execution of an agent has a different implementation for each type of agent. Second, the knowledge base of each derived agent regarding itself, its capabilities, its environment, and its acquaintances is unique. Third, how each agent implements its functionality and capabilities is specific to it.

Each of these derived classes implement their respective agents as generically as possible. That is, the classes, *Assembling_Agent*, *Part_Presentation_Agent*, *Shared_Space_Agent*, and *Task_Scheduling_Agent*, implement a generic version of an ASA, PPA, SSA, and TSA respectively. In this implementation, it is the specialized capabilities and knowledge in the knowledge base of each agent that distinguishes one agent from another and demarcates its proper behaviour. Deriving a new agent class from one of these four classes is an obvious way to

completely specialize the capabilities and knowledge of an agent.

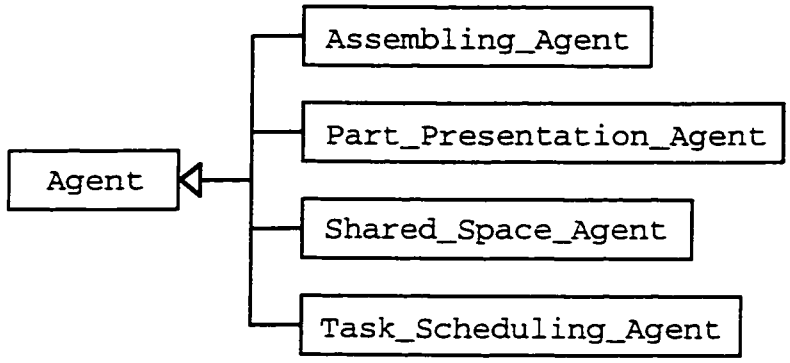


Figure 4.8: Hierarchy of agent classes.

Figure 4.9 displays part of the class hierarchy of the frames of FABRIC. As would be expected, the class *Base_Frame* is at the root of this hierarchy. Three of the frames depicted in this figure are used by each agent to specialize its knowledge. The class *Acquaintance_Agent_Frame* implements a basic frame to store a reference to an agent that is an acquaintance. More specific frames were derived from this class but are not shown in the figure. The class *Model_of_Agent_Frame* represents a basic model of self that at a minimum stores the current state of an agent in a slot and typically also stores slots representing the basic capabilities of an agent. The class *Model_of_Environment_Frame* is available for an agent to represent a basic model of its environment in its slots.

The five classes derived from the class *Message_Frame* implement the five types of messages that are exchanged amongst the FABRIC agents. These messages were described in section 3.5 The base class *Message_Frame* ensures that all messages include the standard contents — sender, recipient, type, and time stamp. The class *Bid_Frame* adds a list of slots or a *Multi_Slot* in order to allow an agent to include any information that the manager agent has requested. The class *Grant_Frame* adds a slot to include the task specification when granting a task to a contractor. The class *Report_Frame* uses the type of a message to identify different types of report messages and it also adds a list of slots to report any information. The class

Request_Frame adds a slot to indicate the type of request and a slot to specify what is being requested. Lastly, the class *Task_Announcement_Frame* extends the class *Message_Frame* by including a slot for the task abstraction, a slot indicating a deadline for receiving bids, and two lists of slots stipulating task eligibility and bid specification.

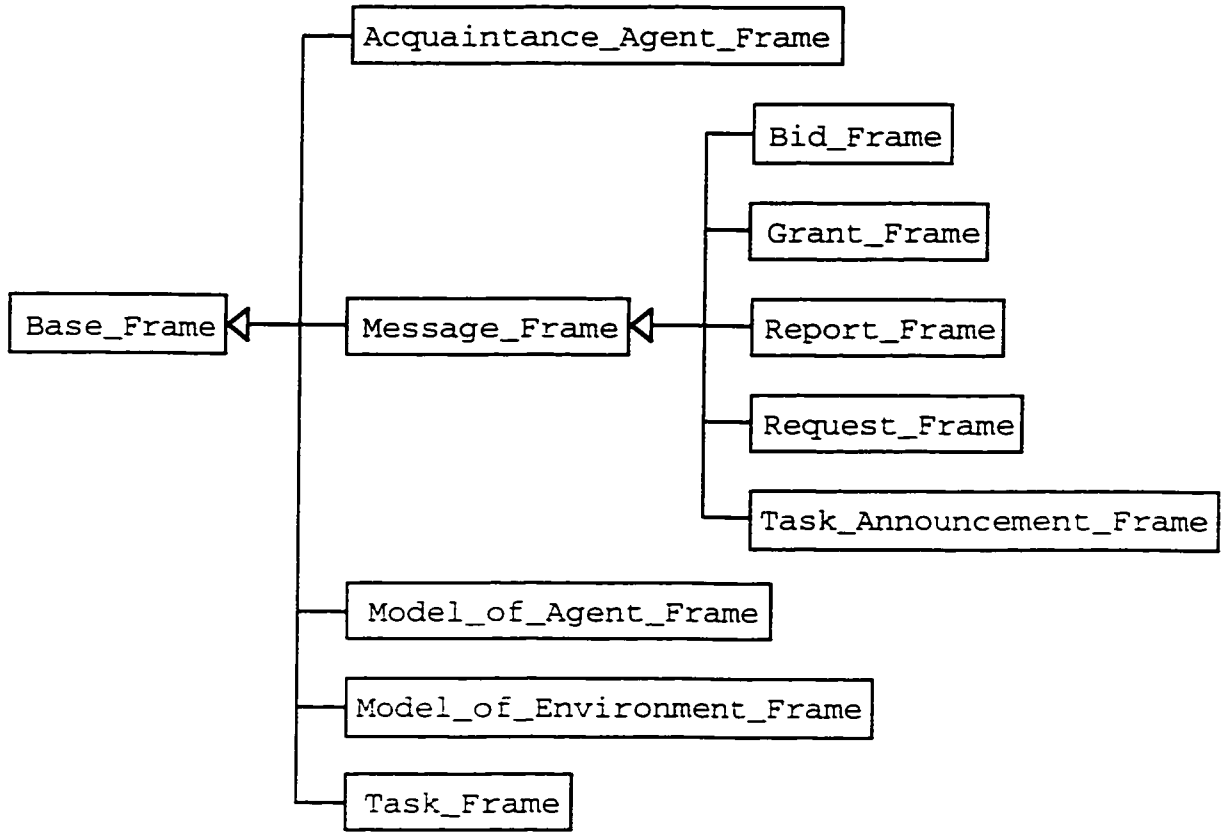


Figure 4.9: A portion of the class hierarchy of the frames.

The class *Task_Frame* is used by the TSA to store all of the information regarding an assembly operation. The slots of this frame represent the different fields of a task-level instruction specifying an assembly operation as described in subsection 3.5.1. In particular, slots representing the fields of an assembly operation, with a specific part, at an assembly location, and with specific options are included. In addition, this frame stores the assembly operations, which must precede an assembly operation, in a list of slots and a slot to store the current state of an assembly operation.

4.4.2 Specific Aspects

This subsection describes the essentials of the internal structure and behaviour of the classes that implement the four types of FABRIC agents. The UML notation is used again in this presentation.

Figure 4.10 displays the core structure and behaviour of the class *Task_Scheduling_Agent*. Three variables define the contents of this class. The variable *_ready_to_launch_tasks* stores the assembly tasks that are ready to be launched or announced using the contract net in a simple queue. The other two variables store the number of assembly tasks left to do in the current production cycle and the number of cycles left to do in the current production run.

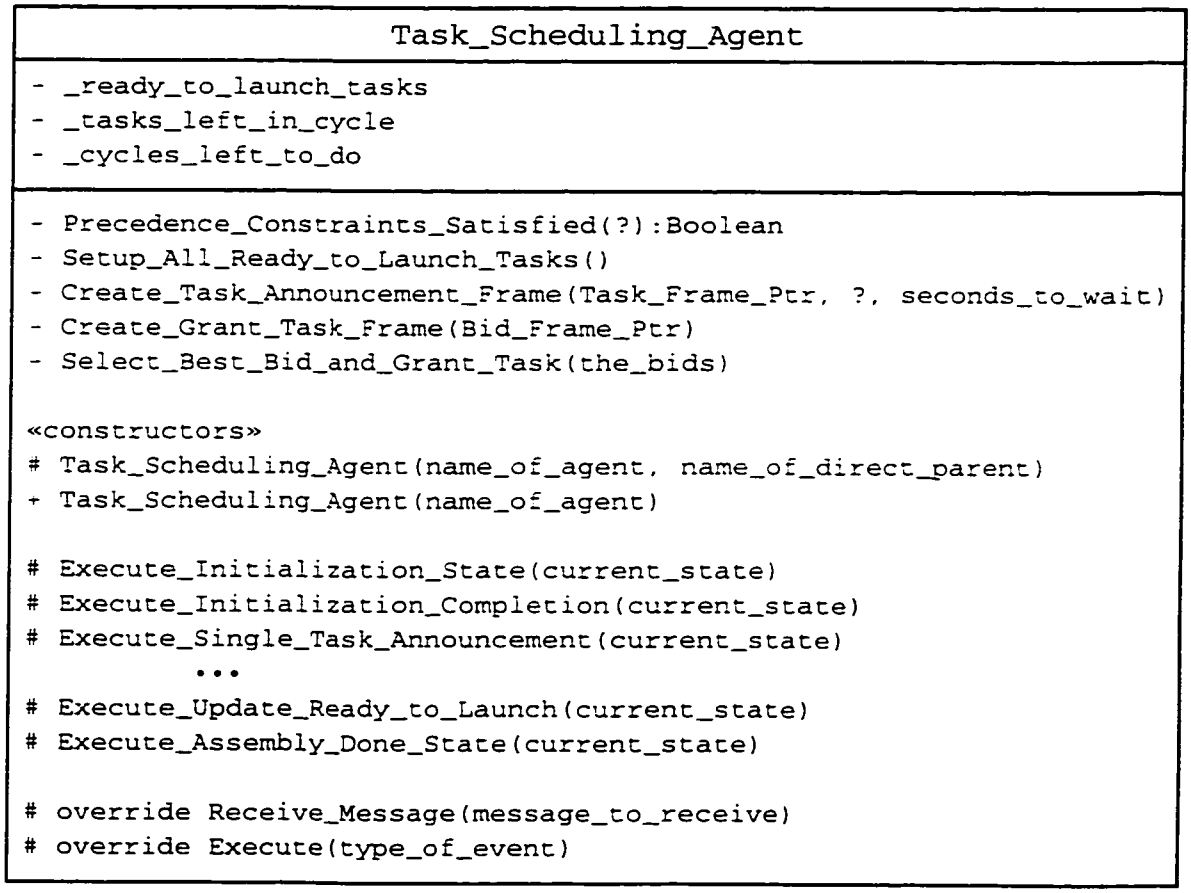


Figure 4.10: Core structure and behaviour of the class *Task_Scheduling_Agent*.

The private methods of the class *Task_Scheduling_Agent* define basic procedures to work

with assembly tasks and be a manager agent in the contract net [Smith, 1979]. Methods to work with assembly tasks include determining if the precedence constraints of an assembly operation are satisfied and setting up assembly operations whose precedence constraints are satisfied into a ready to launch list. Methods implementing the contract net include creating a task announcement frame or message given a pointer to a *Task_Frame* and the number of seconds to wait for bids, creating a grant message given a pointer to the best received bid, and selecting the best bid from a list of bids and granting an assembly task.

The constructors of the four FABRIC agents follow a similar pattern. A protected constructor that requires the unique name of an agent and the name of its direct parent is intended for use by all derived agents or classes. A public constructor that only requires the unique name of an agent is meant to be used to create an agent directly from its corresponding class. There are slight variations but this is the basic pattern.

The next group of methods shown in figure 4.10 also have a similar style for each of the FABRIC agents. Each of these methods begins with the characters *Execute_* and take the current state of the agent as input. An ellipsis displayed in the middle of this group of methods indicates that not all method(s) were included in this list. These methods implement the proper execution of a transition or state in a CSM of an agent. For example, the method *Execute_Initialization_State* sets up all ready to launch assembly operations in state one of the CSM of the TSA, which is shown in figure 3.3. Since the four agents have unique CSMs, these groups of methods vary from agent to agent. These methods are used to implement the overall execution of the CSM of an agent in the virtual method *Execute*.

Recall that, the class *Agent* only defined a basic method to receive a message from another agent and only specified the form of the method in which the execution of an agent is to be implemented. The class *Task_Scheduling_Agent* redefined how it receives a message from another agent and defined its execution by overriding the virtual method *Execute*. With respect to receiving messages, the TSA gives higher priority to report messages than bid messages.

Figure 4.11 displays the core structure and behaviour of the class *Assembling_Agent*. No significant contents were added to this derived class. Private methods implementing the role of a manager in the contract net include creating a task announcement message for the PPA(s) given a pointer to a *Grant_Frame* and the number of seconds to wait for bids, requesting a part from acquaintance PPA(s) given a pointer to a *Grant_Frame*, creating a grant message given a pointer to the best received bid, and selecting the best bid from a list of bids and granting a fetch part task.

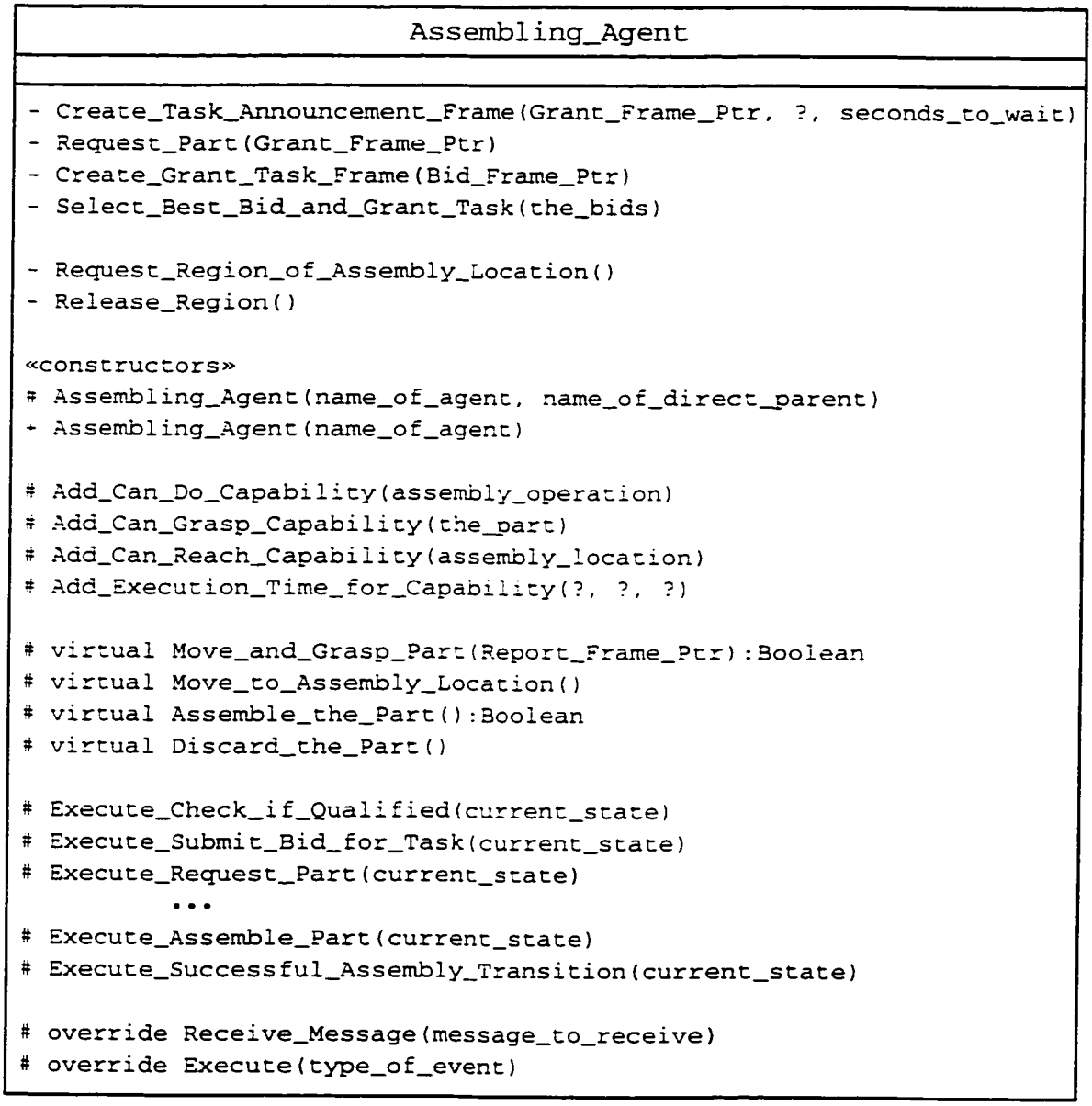


Figure 4.11: Core structure and behaviour of the class *Assembling_Agent*.

Private methods to request and release the region of space of an assembly location from the SSA were also implemented.

The protected and public constructors of this class have the same pattern as the class *Task_Scheduling_Agent*. Four protected methods were defined to make it easier to specialize the self knowledge of an ASA regarding its capabilities. The first method adds the capability to perform a certain assembly operation to the repertoire of an ASA. The second method adds the capability of an ASA to grasp a particular part. The third method adds the capability of an ASA to effectively reach an assembly location. The last method adds the information of the average time that an ASA takes to execute a specific assembly operation to its model of self.

Four virtual methods define the expected physical capabilities of an ASA. These methods are declared virtual so that a specific ASA derived from this class can implement the execution of its physical capabilities in its own way. The first method requires that an ASA will move and grasp a part given a report from a delivering PPA and return a *Boolean* indicating whether the grasp was successful or not. The success or failure of this grasp affects the transition that is made in the CSM of an ASA. The second and third methods require that an ASA moves to the assembly location and then proceeds to assemble the part according to the demands of an assembly operation. The fourth method is only executed if an ASA fails to properly assemble the part as indicated by the third method.

A few of the methods that implemented the execution of the transitions and states of the CSM of an ASA are shown in figure 4.11. For example, the method *Execute_Check_if_Qualified* determines if an ASA is qualified to submit a bid given the conditions of task eligibility specified by the TSA. Finally, the class *Assembling_Agent* overrides the methods to receive a message from another agent and to define its own independent execution.

Figure 4.12 displays the core structure and behaviour of the class *Part_Presentation_Agent*. No significant contents were added to this derived class. A private method implements a procedure to conveniently access its model of self and return whether or not it needs a pickup message upon

delivering a part to an ASA. The protected and public constructors differ from the other agents in that these constructors must have a parameter indicating whether or not a PPA requires a pickup message in addition to the regular parameters.

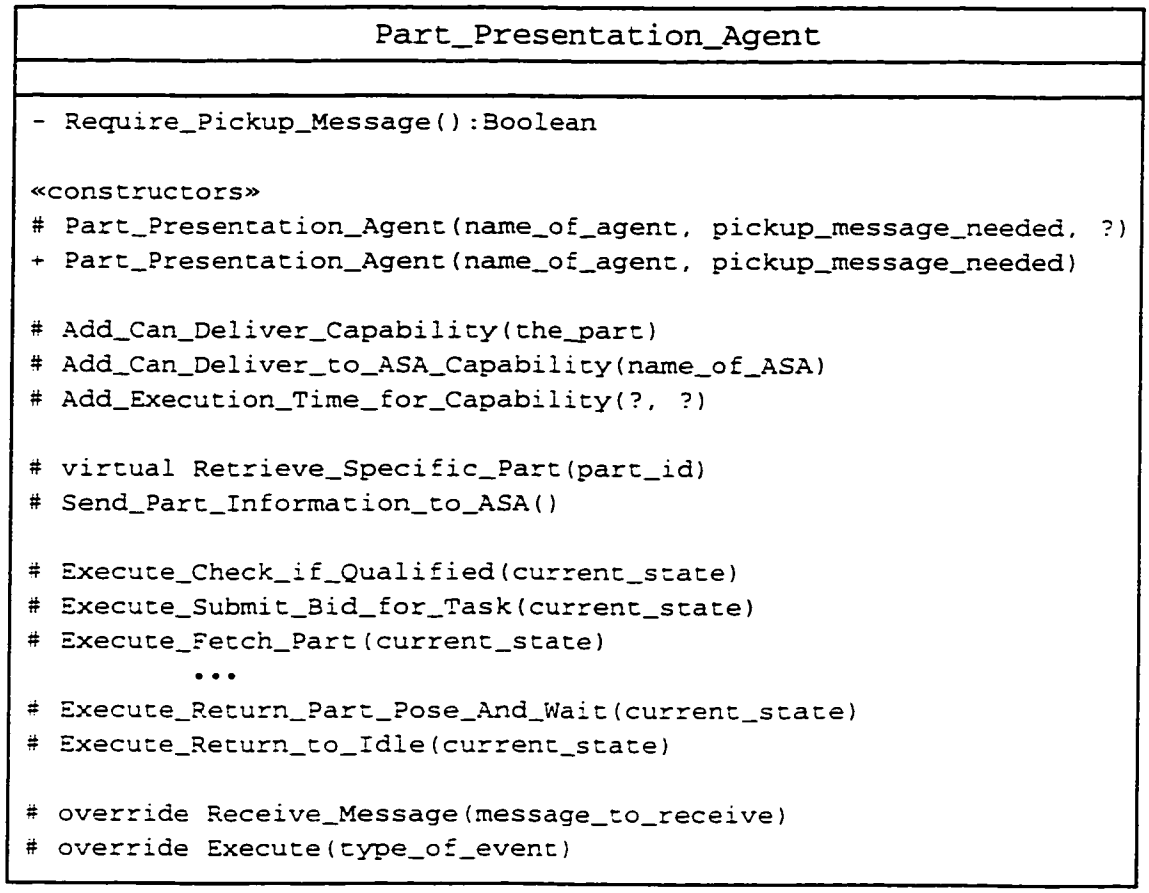


Figure 4.12: Core structure and behaviour of the class *Part_Presentation_Agent.*

Three protected methods were defined to make it easier to specialize the self knowledge of a PPA regarding its capabilities. The first method adds the capability to deliver a specific type of part. The second method adds the capability of delivering to a specific ASA. The third method adds the information of the average time that a PPA takes to fetch a specific part to its model of self.

The expected physical capability of a PPA is expressed as a single virtual method, called *Retrieve_Specific_Part.* This method is declared virtual in order that each PPA derived from this

class can implement its own physical techniques to retrieve or fetch a part. The related protected method, *Send_Part_Information_to_ASA*, makes it easy to send the pose, position and orientation, of the part to a manager ASA.

Some of the methods that implement the execution of a transition or state of the CSM of a PPA are also listed in figure 4.12. For instance, the method *Execute_Fetch_Part* fetches the part using the previously discussed methods and also makes the transition from state two to state three of the CSM of a PPA. The class *Part_Presentation_Agent* also overrides the methods to receive a message from another agent and to define its execution.

Figure 4.13 displays the core structure and behaviour of the class *Shared_Space_Agent*. This class contains a single variable, *_unsatisfied_requests*, which keeps a list of all requests for regions of space from an ASA or PPA that were not satisfied. The private methods of this class define methods to conveniently handle requests for exclusive access to a specific region of space. The first method returns a *Boolean* indicating whether a region of space is currently occupied given a pointer to a *Request_Frame*. The second method sets up a report message to grant a particular region of space to a requesting agent. The third method uses the first two methods to handle a request for a region of space and returns a *Boolean* indicating whether the region was granted or not. This return value changes the transition that a SSA makes in its CSM.

The protected and public constructors of this class have the same pattern as the class *Task_Scheduling_Agent* and the class *Assembling_Agent*. A single protected method was also defined in order to make it easy to specialize the model of its environment by adding different regions of shared space.

The next group of methods implement the proper execution of a transition or state in the CSM of a SSA. A general observation is that for all agents there were more of these execution methods defined in the actual class than listed in their figures. As expected, the class *Shared_Space_Agent* also overrides the methods to receive a message from another agent and to define its autonomous execution.

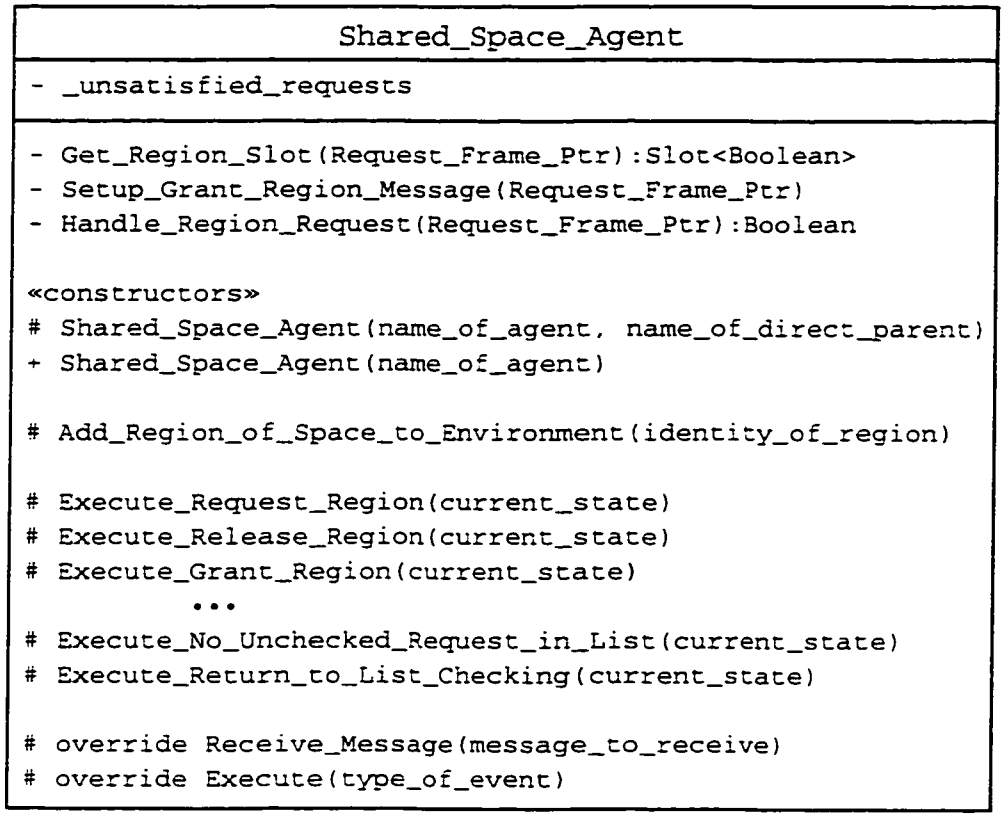


Figure 4.13: Core structure and behaviour of the class *Shared_Space_Agent*.

In closing, appendix B lists the C++ source code for the class *Task_Scheduling_Agent* while appendix C lists the C++ source code for the class *Assembling_Agent*. A very brief account of the software implementation detailed in this chapter was published in [Basran et al., 1997a].

4.5 Synopsis

This chapter provided a description of the software implementation of the agents used to decompose a MRAC into an agent-based system. This implementation was made possible through the use of object-oriented software technology. In particular, we chose the object-oriented programming language C++ for the three reasons of efficiency, that it is more common in industry, and building upon an existing system programmed in C.

Having chosen this object-oriented programming language, it was a nontrivial task to define and implement the classes for the different agents of our framework. One of the basic

difficulties is that the oriented paradigm in general and the C++ programming language in particular are not meant to implement knowledge bases. Programming languages like Lisp or Prolog are better suited to implement such knowledge bases. Although it was a disadvantage to implement a frame-based knowledge base in C++ from scratch, it was also a significant accomplishment.

An advantage of an object-oriented implementation is that it makes it easier to create or instantiate specific instances of each agent and therefore, different configurations of FABRIC systems.

Chapter 5

Physical Level Evaluation of the Framework

Of the four types of agents implemented in software, both ASAs and PPAs should also have an associated physical implementation; however, ASAs are definitely more critical to the potential usefulness of a FABRIC system. For this reason, we investigated the implementation of a proof of concept, actual ASA[†]. This chapter details the implementation of this ASA within the context of a FABRIC system in order to provide a partial evaluation of the physical level, feasibility of our framework. This chapter is based in part on [Basran and Petriu, 1997; Basran et al., 1997c[‡]].

This chapter is partitioned into four sections. The first section introduces the reader to the overall implementation of this physical agent. The second section describes a preexisting robotic programming environment upon which we based our implementation, before detailing how we adapted and extended this environment in order to implement an ASA. In the third section, we demonstrate the operation of this ASA within a FABRIC context for a basic, peg in hole, assembly operation specified as a task-level instruction. The fourth section discusses how this implementation relates to existing research and what this implementation implies.

[†] The implementation of this ASA was carried out while I was a guest worker of the Visual Information Technology group which belongs to the Institute for Information Technology at the National Research Council of Canada in Ottawa.

[‡] Permission to do so was granted by the International Society for Computers and Their Applications.

5.1 Introduction to the Overall Implementation

This section begins by describing the robotic system that we used to physically implement an ASA. This section continues by presenting the software implementation of a class encapsulating the assembly capabilities of this robotic system into an ASA.

Figure 5.1 displays the physical robotic system. This robotic system is a Puma 500, industrial manipulator that has six degrees of freedom. Two sensors are mounted on the wrist of this manipulator. The force/torque sensor measures the forces and torques around three orthogonal axes of its wrist. The range sensor measures the distance from its wrist to the physical surroundings along a single projected line or profile of laser light. A single range profile contains 256 surface measurements.

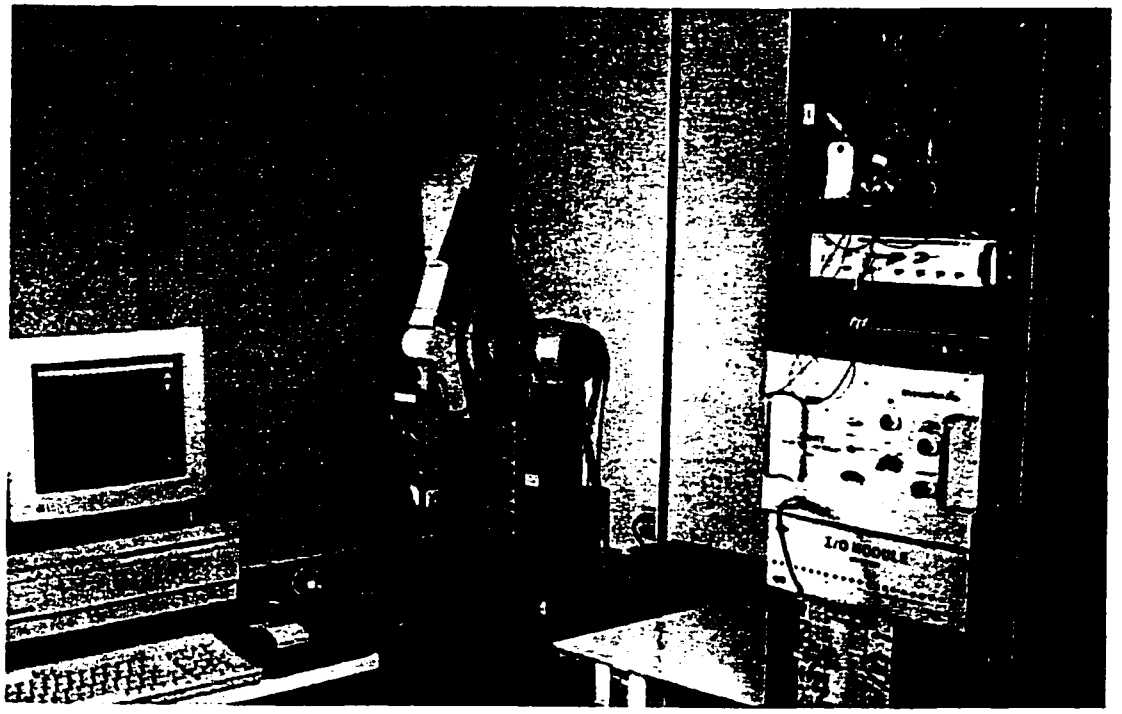


Figure 5.1: Physical implementation of an assembling agent.

The basic question is how to integrate this robotic system into the form required of an ASA. Figure 5.2 displays the class hierarchy of the agents with the class implementing a Puma 500, ASA added with a bold border. The class *Assembling_Agent_Puma500* was derived from

the class *Assembling_Agent*. This method of specializing the capabilities and knowledge of an agent by deriving a new agent class from one of the base agent classes was already noted in subsection 4.4.1.

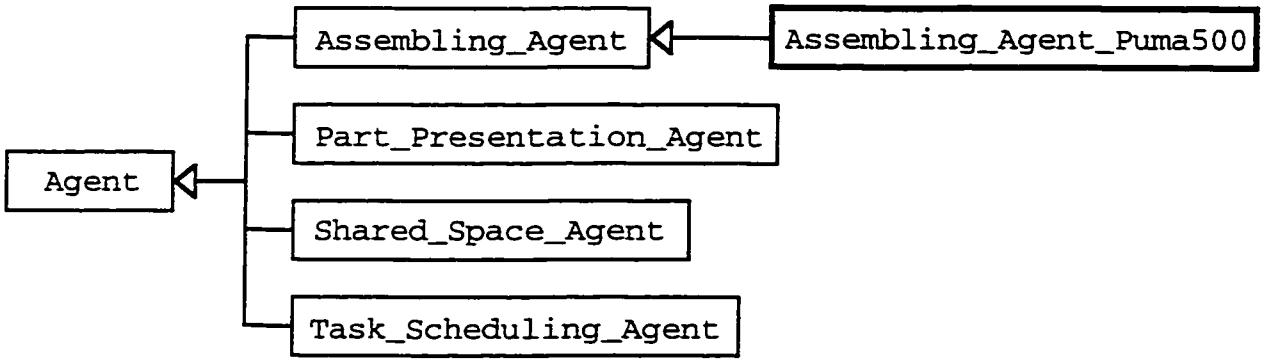


Figure 5.2: Class hierarchy of the agents with a Puma 500, ASA added.

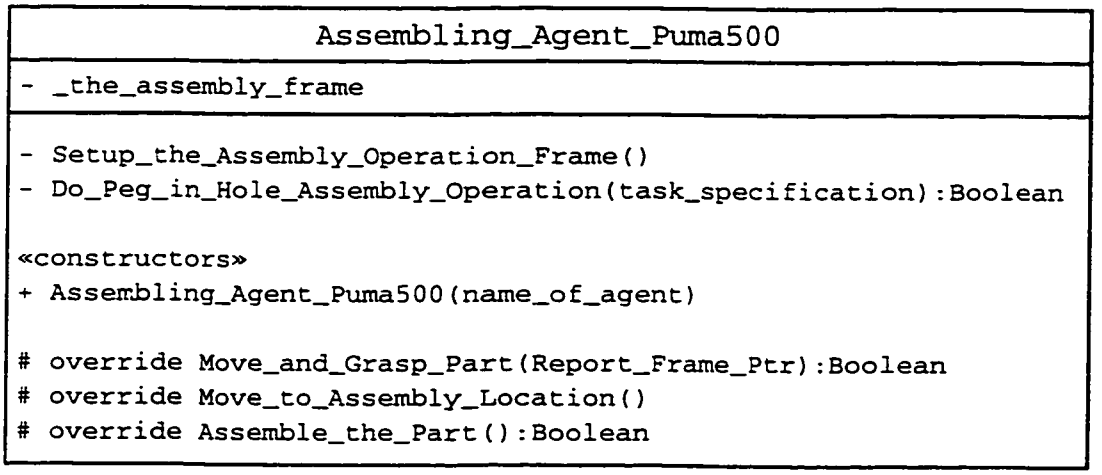


Figure 5.3: Essential structure and behaviour of the class *Assembling_Agent_Puma500*.

Figure 5.3 depicts the essential structure and behaviour of this derived class using the UML notation [RATIONAL, 1997] described in the previous chapter. The single variable, *_the_assembly_frame*, points to a frame that represents both the declarative and procedural knowledge regarding the current assembly operation. Two private methods are indicated in this class diagram. The first method sets up the proper assembly frame for the current assembly task that was granted from a TSA. Given the specification for a task-level instruction, the second

method carries out a peg in hole, assembly operation. Technically, a private method for each assembly operation that a physical ASA can do should be here but in our implementation we only considered the peg in hole. The public constructor of this class simply takes a unique name as input. The Puma 500, ASA class has overridden the execution of three of the four virtual methods outlining the physical capabilities of an ASA as declared by the class *Assembling_Agent*. That is, the Puma 500, robotic system has its own techniques to move and grasp a part, move to an assembly location, and assemble the part according to the demands of an assembly operation. A method to discard a part after an unsuccessful assembly operation was not implemented.

5.2 Details of Implementation

This section expounds upon how we implemented an ASA using a Puma 500, robotic system. Since the answer to this question depends in part upon the preexisting state of this robotic system, this section begins by describing the most relevant aspect of this system.

5.2.1 Preexisting Robotic Programming Environment

Before summarizing the preexisting robotic programming environment [Archibald and Petriu, 1993; Archibald, 1995], we introduce the field of robot programming with a slant towards assembly, task-level programming.

The field of robot programming concerns itself with the efficient computer programming of the operation of a robotic system and many researchers have written expositions on this field including [Bonner and Shin, 1982; Lozano-Pérez, 1983; Blume and Jakob, 1986; Rembold and Hörmann, 1987; Gini and Gini, 1991; Capizzi et al., 1993]. Approaches to robot programming still fall into three major categories [Lozano-Pérez, 1983]. The first category is teach pendant programming or teach by guiding the robot through what it has to do. The second category is robot-level or explicit programming. In this case, the detailed instructions that an industrial robot has to carry out are specified step by step using the constructs of a robot-level programming language. The third category is task-level or implicit programming. In this case, the instructions

to the industrial robot are specified at the level of robot independent, operations on parts. In the majority of cases, the tasks of task-level robot programming are assembly operations. Just a reminder that the process of converting task-level, assembly operation, instructions into actions executable by a robotic system is known as task planning [Gottschlich et al., 1994].

Two attributes help to further distinguish between approaches to creating robotic programming environments. The first attribute has to do with whether the programming is on-line or off-line [Gini and Gini, 1991]. On-line programming means that the physical industrial robot is used during program development and is unavailable to the manufacturing system during this time period. Off-line programming means that the initial program is developed and tested typically on a 3D graphic simulation of the industrial manipulator operating as part of a cell or manufacturing system. The advantage is that the actual industrial robot is not removed from the manufacturing system during program development; however, these programs are less reliable since they were only tested in simulation and usually require on-line, fine tuning. Ravani [1988] published a collection of papers that dealt primarily with off-line programming issues. James and Guptill [1993] described a more recent implementation of an off-line programming system. The second attribute has to do with whether the robot programming interface is textual or visual. The bulk of early and existing programming interfaces were textual; however, more recently researchers have advanced the notion of visual programming interfaces.

A few researchers that have focused on the assembly aspects of task-level programming are presented. One of the earliest languages for describing assemblies and instructing an assembly robot was RAPT [Popplestone et al., 1978, 1980]. Popplestone et al. [1980] stated that they were not concerned with the run-time program to control a manipulator. Collins et al. [1984] described a benchmark for planar assembly that allowed a comparison of the capabilities of assembly robot programming systems. Nnaji [1993] thoroughly described and explored various aspects of assembly, task-level, robot programming from a CAD perspective. In closing, researchers continue to examine the problem of programming assembly tasks [Holm et al., 1993; Deacon and

Malcolm, 1994; Sato and Maciejewski, 1994; Toukal et al., 1995].

We now give a synopsis of the preexisting robotic programming environment whose acronym was SKills-Oriented Robot Programming (SKORP) [Archibald and Petriu, 1993; Archibald, 1995]. SKORP is an on-line, visual, robotic programming environment. The primary goal of SKORP was to reduce the cost of programming robotic systems. The concept of a robotic skill is critical to this environment. A skill is a parameterized, sensing and/or actuation, action that a robot can accomplish repeatably [Archibald and Petriu, 1993; Archibald, 1995]. A skill is represented as an icon and its parameters are interactively modifiable through a dialog window.

According to [Archibald and Petriu, 1993; Archibald, 1995], the robot programmer develops an operation by connecting skill icons together into a desired sequence of robotic execution, determining and entering the required parameters for each skill, and debugging the entire operation by executing it. The debugging process usually entails adjusting the parameters for each skill and may require modifying how the skills connect together to form an operation. Using SKORP, a robot programmer could connect together a sequence of robotic skills and adjust the parameters in order to implement a task-level, assembly operation.

5.2.2 Puma 500, ASA

Having adopted the basic structure of this robotic programming environment, the question becomes how do we use it in order to deal with the task planning problem that is part of the internal operation of an ASA. In the process of addressing this problem, we advanced from SKORP [Archibald and Petriu, 1993; Archibald, 1995] in that the robot programmer did not interactively place, connect, and modify the sequence of robotic skills, but an ASA automatically generated and connected its skills together into an assembly operation with the appropriate parameter values. Also note that providing a solution to the task planning problem strongly supports the task-level approach to robot programming.

Figure 5.4 displays our approach to address the task planning problem for the Puma 500, ASA. The basic approach that we used for task planning was task translation, that is, parse each

task-level instruction and then in conjunction with the frame-based knowledge base decompose each assembly operation into a sequence of robotic skills with all parameter values fully specified. The values of these skill parameters are determined from the task-level instruction and through the knowledge base of this ASA. Clearly, the type and level of abstraction of the basic skills executable by a robotic system constrain the assembly operations that its corresponding ASA can convert. Both the preexisting skills and the new skills developed for this ASA were implemented in the C programming language.

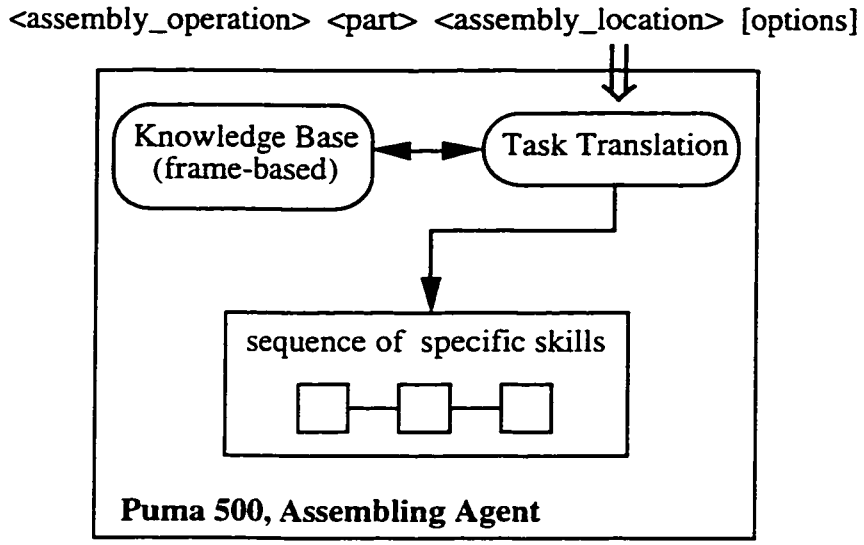


Figure 5.4: Translation of assembly operations into skills by the Puma 500, ASA.

Given this knowledge-based approach to task translation, what frames did the Puma 500, ASA define in order to do this translation. These frames also consequentially store the knowledge about the capabilities of this ASA in its knowledge base. Figure 5.5 displays the frame classes that the Puma 500, ASA added to the class hierarchy of frames. The original class hierarchy of frames was depicted in figure 4.9. In the explanations to follow, we use the concepts of the frame knowledge representation scheme in order to present the basic structure and behaviour of these classes.

The class *Skill_Frame* and its derived classes represent the robotic skills of the Puma 500, ASA as frames. The class *Skill_Frame* defines two basic slots. The first slot stores the name of a

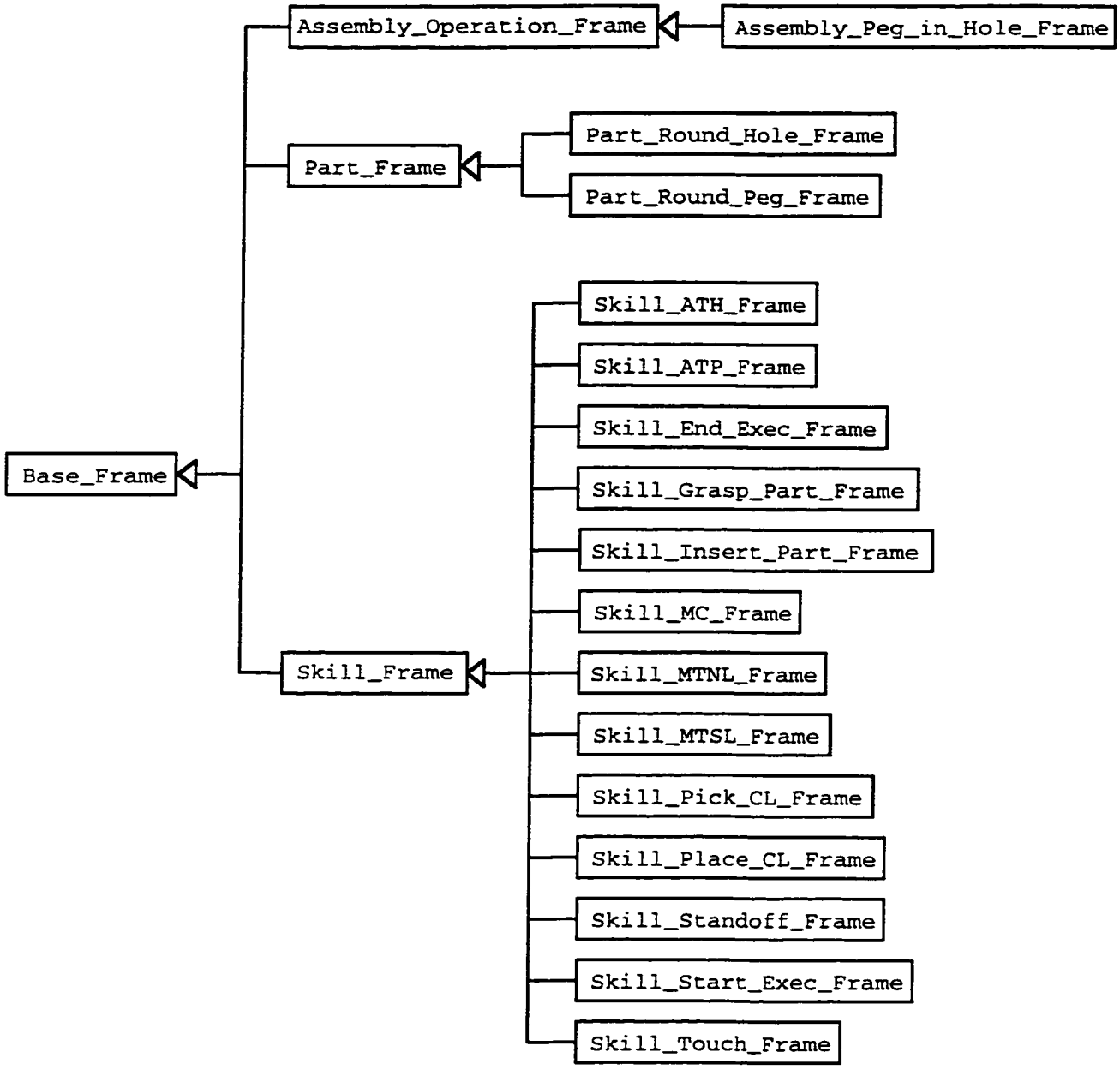


Figure 5.5: Frames added and used by the Puma 500, ASA.

class that implements a skill's visual representation and stores its parameter(s). The second slot actually stores a pointer to an instance of the skill class specified by the first slot. The thirteen derived classes or frames, which begin with the characters *Skill_*, override the values of these two slots plus add slots, if necessary, to store the specific parameter(s) of each skill. Of the thirteen derived skill frames, four represent new skills that we implemented. These four new skills were

align to hole, align to plane, grasp part, and insert part. We discuss three of these four skills in greater detail in subsection 5.3.3. The base class *Skill_Frame* also provides an interface to change the values of the skill parameter slots.

The class *Assembly_Operation_Frame* is used by the Puma 500, ASA to store the common declarative and procedural knowledge on executing an assembly operation specified as a task-level instruction. This class defines a slot to store the type of assembly operation. This class also stores a list of skill frames that is used to specify a sequence of skills for completely decomposing a particular assembly operation. The common knowledge of this class has to do with the fact that during all assembly operations an ASA has to grasp a part and move to an assembly location before performing the actual assembly operation. This class defines methods that our ASA can call in order to execute these steps of an assembly operation. The derived class *Assembly_Peg_in_Hole_Frame* overrides the slot defining the type of assembly operation and appends the skill frames that implement a basic, peg in hole, assembly operation to the inherited list of skill frames. This derived class also defines a method that our ASA can call in order to perform a peg in hole.

The class *Part_Frame* is used to represent information on a part. This class defines a slot to store the type of part. The two classes derived from this base class override the value of this slot and add slots to store the specific parameter(s) of their part type. For instance, the class *Part_Round_Peg_Frame* simply stores two slots, one storing the diameter and the other the height in centimeters.

The specialized knowledge stored in all of these frames about how to perform assembly operations, plus the corresponding knowledge stored in the model of self about what assembly operations are doable, is what differentiates this ASA from any another.

5.3 A Demonstration

This section describes the details behind the demonstration of a basic, peg in hole,

assembly operation carried out by the Puma 500, ASA in a FABRIC context. This section is divided into three subsections. The first subsection provides an overall perspective of this demonstration. The second subsection describes the specific breakdown of this assembly operation into robotic skills. The third subsection discusses certain details of the new skills implemented for this demonstration.

5.3.1 Overall Perspective

In the demonstration that we performed with the Puma 500, ASA, it was treated as one of agents within the overall FABRIC framework. In particular, a simulation/experiment with a TSA, a virtual ASA, an actual ASA, two virtual PPAs, and a SSA was run. The adjective virtual refers to an agent that should have a physical implementation but whose operation is implemented only in software. The agents of this simulation/experiment were instantiated from the software classes described in section 4.4 and section 5.1.

At some point during a simulation run, the TSA, which was implemented as a specific instantiation of the class *Task_Scheduling_Agent*, announces the task-level, assembly operation, instruction: *peg_in_hole round_peg fixture1*. We describe what happens from the point of view of the Puma 500, ASA at the level of its CSM. This ASA was implemented in software as an instance of the class *Assembling_Agent_Puma500*. This ASA is first granted this announced instruction from the TSA through contract-net based negotiation. Then, this ASA requests and receives the *round_peg* and its location from one of the 2 virtual PPAs. Having grasped the *round_peg* and obtained exclusive access to the region of space of the assembly location from the SSA, this ASA proceeds to the location of *fixture1* and then performs a *peg_in_hole* assembly.

5.3.2 Peg in Hole - Skill Breakdown

At an internal level, the Puma 500, ASA breaks down its execution of an assembly operation at the CSM level into a sequence of robotic skills with all parameters specified. This section describes the complete translation of the peg in hole, specified in the previous subsection, into a sequence of skills.

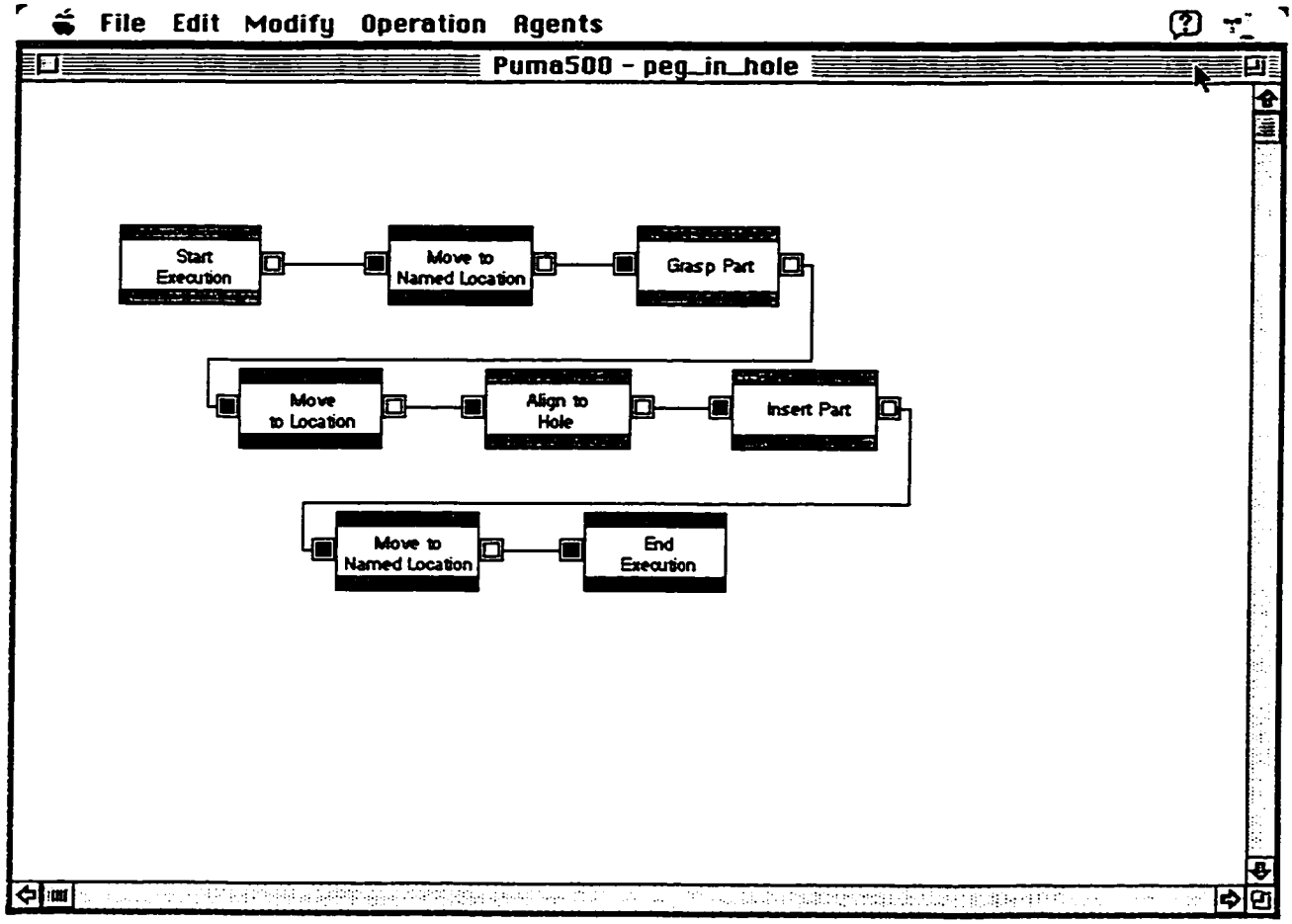


Figure 5.6: The sequence of skills automatically generated for a peg in hole, assembly operation.

Figure 5.6 displays a screen dump of the Macintosh™ computer application that ran the simulation/experiment for our demonstration. This application, which is based upon the research of [Archibald and Petriu, 1993; Archibald, 1995], advances from the original system as described in subsection 5.2.1. In this screen dump, the complete *peg_in_hole* operation was broken down into eight skills by the Puma 500, ASA. Each of these skills were implemented as robustly as possible by including explicit sensor routines that test for proper execution. The first skill simply allows this ASA to prepare for skill execution. The second skill moves the Puma 500 robot to the location of the round peg as specified by one of the virtual PPAs. The third skill, *Grasp Part*, uses the range sensor to grasp the *round_peg*. The fourth skill moves this robot to the assembly location *fixture1*. The fifth skill, *Align to Hole*, uses the range sensor to align the Puma 500 robot

perpendicular to the hole of interest. The sixth skill, *Insert Part*, uses the range sensor and the force sensor to insert the *round_peg* into the hole. The seventh skill simply moves the Puma 500 robot back to its idle location. The eight skill ensures that the execution of this assembly operation is properly terminated by this robot.

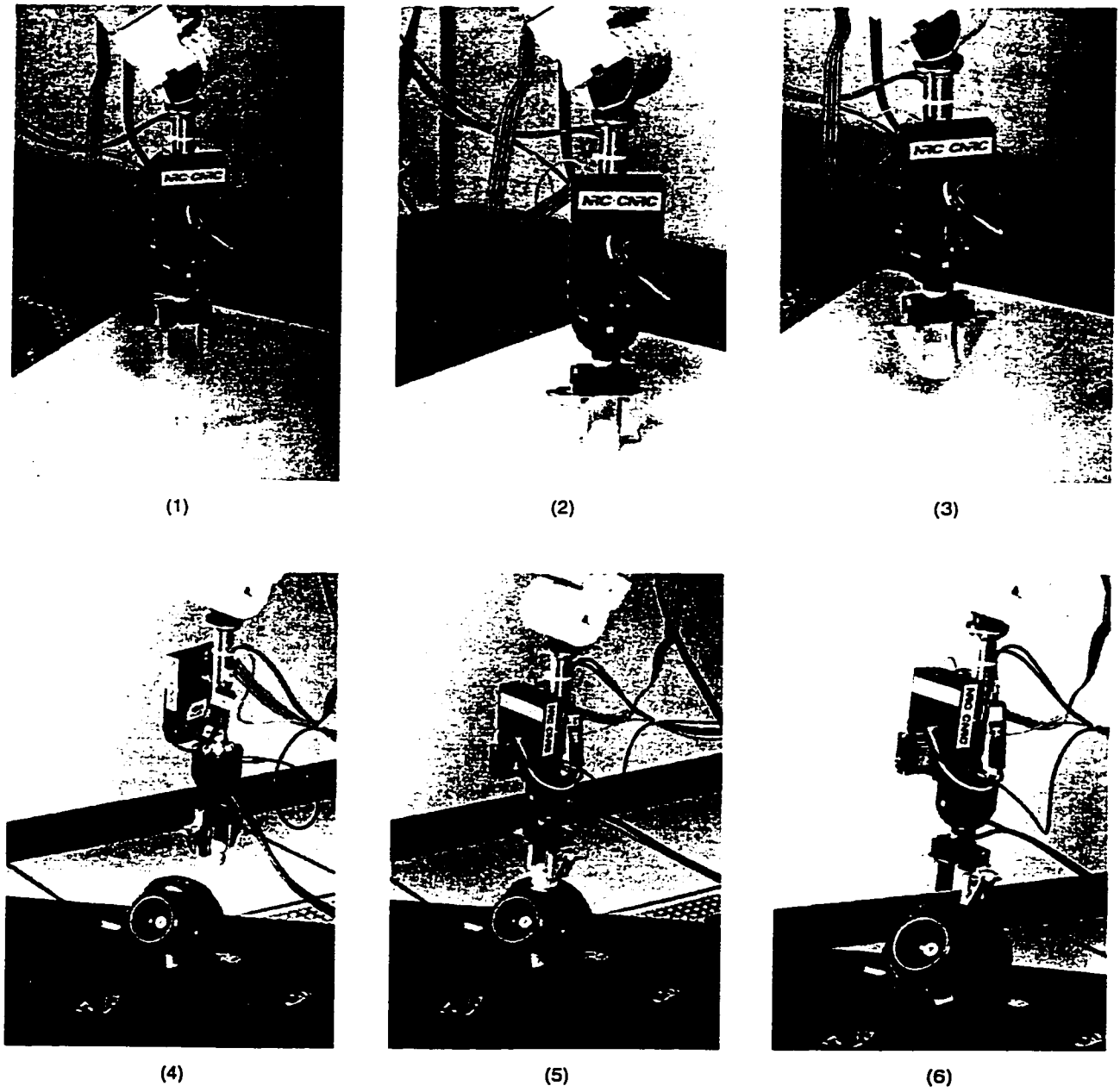


Figure 5.7: Sequence of pictures demonstrating a peg in hole, assembly operation.

Figure 5.7 contains a sequence of pictures that show the Puma 500 robot carrying out this task-level instruction to perform a peg in hole, assembly operation with a round peg. The first three pictures display the robot grasping the round peg. The fourth picture displays the robot aligning itself to the hole using the range sensor. The fifth and sixth pictures display the robot inserting the round peg into the hole.

5.3.3 Details of New Skills

This section presents a few details of the skills, *Grasp Part, Insert Part, and Align to Hole.* Each of these skills uses a local model to execute a parameterized action as robustly as possible.

The skill parameters used for two of these skills are enumerated. The parameters for *Grasp Part* are the type of part, the speed with which to grasp the part, and a list of geometric parameters that for a basic round peg are diameter and height. The parameters for *Insert Part* include the parameters for *Grasp Part* plus a list of geometric parameters of the hole.

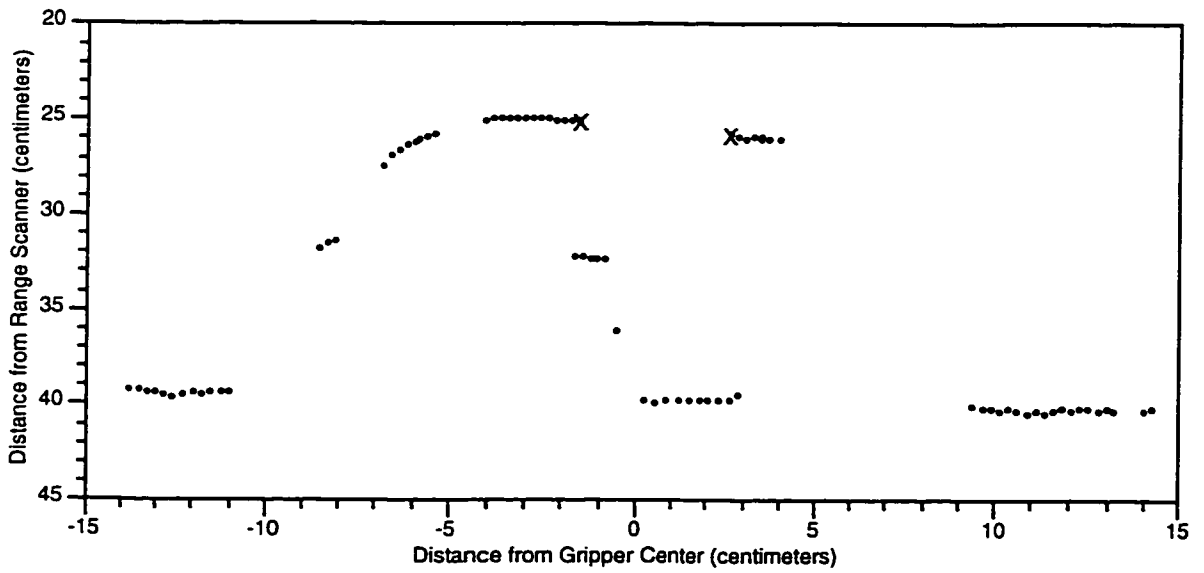


Figure 5.8: Range profile obtained and processed during the *Align to Hole* skill.

Figure 5.8 shows a single profile of typical range data obtained from the range scanner that is mounted on the wrist of the Puma 500 robot. This range data was processed during the execution of the skill, *Align to Hole*. Each dot in this diagram represents a single surface

measurement taken by this range scanner. The horizontal axis represents the distance from the center of the gripper of this robot to a surface measurement. The vertical axis represents the distance from the mount point of this range scanner to a single measurement. In this plot, the edges of the round hole are marked by two x's. These edges were determined by filtering the range data and then searching for an appropriately sized gap in the range data.

5.4 Discussion

This section begins by relating how the internal implementation of our ASA compares with existing research. The primary focus is on the various issues associated with the implementation of task planning [Gottschlich et al., 1994] for an assembly robot in the context of task-level robot programming [Lozano-Pérez, 1983]. As much as possible, we emphasize other research projects that have worked with actual robots.

To begin with, the problem of task planning has been examined by many researchers including [Smithers and Malcolm, 1987; Malcolm and Smithers, 1990; Petropoulakis and Malcolm, 1990], [Segre, 1988, 1991], [Hörmann, 1989, 1992; Hörmann et al., 1989; Hörmann and Rembold, 1991], [Lozano-Pérez et al., 1989, 1992], [Gottschlich and Kak, 1990; Hutchison and Kak, 1990; Tung and Kak, 1994, 1996], [Chen and Trivedi, 1992, 1995], [Ramos, 1992; Ramos and Oliveira, 1992], and [Vijaykumar and Arbib, 1987; Kimbler and Malave, 1989; Ko and Lee, 1991; Rubin and Dilworth, 1992; Chu and ElMaraghy, 1993; Czarnecki, 1995; Bagchi et al., 1996].

Our approach to the task planning problem is unique in three ways [Basran and Petriu, 1997]. First, unlike task planning that focuses on deciding the order and the type of robotic operations to perform in order to complete a task-level instruction, we emphasize task translation — determining and specifying the parameters of the prespecified skill sequence for a task-level instruction. This decision made our approach to this planning problem less sophisticated than certain planners, such as [Lozano-Pérez et al., 1989, 1992; Hutchison and Kak, 1990; Tung and Kak, 1994, 1996], but at the same time our technique is still adequate for a robotic manipulator

operating in an assembly cell. Second, we completely encapsulate our task planning approach within the internal structure of an ASA. Third, our approach occurs within the larger context of the FABRIC framework and this framework supports the loosely coupled coordination of multiple ASAs. These distinctions reflect the fact that our approach merges ideas from three fields of study: robot programming, assembly and task planning, and MASs.

With respect to skills or elementary operations for robotic assembly, several researchers have developed and investigated such skills [Nakamura and Xu, 1988; Cheng et al., 1991; Hasegawa et al., 1991, 1992; Chongstitvatana, 1994; Kaiser et al., 1994; Lee et al., 1994; Morrow and Khosla, 1995, 1997; Pfeiffer, 1996; Suehiro and Kitagaki, 1996; Zhang et al., 1997]. Our implemented skills have an average degree of sophistication when compared with the complexity of skills implemented by these researchers.

Our approach to task planning, where task-level, assembly operation, instructions are decomposed into robotic skills, has two major characteristics in common with previous implementations. First, like some approaches to task planning [Hörmann, 1989; Kimbler and Malave, 1989; Chen and Trivedi, 1992; Ramos, 1992], some approaches to implementing robotic skills [Nakamura and Xu, 1988; Chongstitvatana, 1994; Lee et al., 1994], and some task-level programming systems [Clermont et al., 1986; Rodighiero and Canciani, 1987; Frommherz and Werling, 1989; Benameur et al., 1991; Kelley, 1991], we hierarchically decompose an assembly operation. Second, we use a frame-based knowledge representation scheme [Minsky, 1985] and this scheme has been used by [Kak et al., 1986; Nakamura and Xu, 1988; Hörmann, 1989; Chen and Trivedi, 1992].

We continue by noting that the peg in hole is one of the most thoroughly studied assembly operations [Inoue, 1979; Goto et al., 1980; Mason, 1981; Raibert and Craig, 1981; Whitney, 1987; Gottschlich and Kak, 1989; Strip, 1989; Park and Cho, 1993; Patarinski and Botev, 1993; Qiao et al., 1993; Volpe and Khosla, 1993; Bruyninckx et al., 1995; Gürocak and Lazaro, 1995]. One of the main reasons for our interest in this operation is that it is usually the most common

operation for assemblies.

A few comments on the software implementation of the complete system and of the Puma 500, ASA are made. Including the five program files, which implement the five agent classes described in the previous chapter, but excluding template classes and the C source code implementing the four new robotic skills, the complete software implementation of this demonstration in C++ required 114 program files. Of these files, 63 of them were needed to implement the basic robotic programming environment while the remaining 51 implemented the FABRIC system. The object-oriented techniques provided in the C++ programming language were essential to the practical implementation of this system. The last comment is that an advantage of implementing both the behavioral part of an ASA, the state machine, and its intelligent part, the knowledge base, in the C++ programming language was that the integration of the two parts was seamless.

We address the question of what our implementation of an ASA implies. Our approach to an agent-based MRAC supports task-level programming in that the Puma 500, ASA decomposes an assembly operation into a sequence of skills that are presented to the programmer in a visual debugging environment [Archibald and Petriu, 1993; Archibald, 1995]. This potentially allows a programmer to debug the unsuccessful execution of a task-level, assembly operation.

In closing, the successful demonstration of a basic, peg in hole, assembly operation with the Puma 500, ASA in a FABRIC context supports three key points. First, it indicates that our approach would work in principle for other assembly operations. Second, it demonstrates that our approach to the task translation problem is viable. Third, the integrated simulation/experiment partially validates the feasibility of implementing the physical agents of FABRIC.

Chapter 6

System Level Evaluation of the Framework

This chapter details the computer simulations that we did in order to evaluate the system level operation of the FABRIC framework. This chapter is made up of five sections. The first section introduces the topic of simulation from the point of view taken in this dissertation. The second section describes how the specific concepts of simulation that were discussed in the first section were applied to a FABRIC system. This section also describes the hypothetical assembly cell that our simulations are based upon. The third section discusses certain aspects of the software implementation of a FABRIC simulation. The fourth section presents the results of our simulations. The fifth section uses these results in order to compare FABRIC systems both between different physical configurations and the same configurations under different operating circumstances.

6.1 Introduction to Simulation

The notion of a system and its properties are central to any simulation [Pooch and Wall, 1993]. For our purposes, a system is a collection of interacting and interdependent entities. A continuous system is one whose changes in state occur continuously over time, while a discrete system is one whose changes in state only occur at specific points in time [Pooch and Wall, 1993]. A hybrid system possesses the property of being both continuous and discrete. A system also has the property of being deterministic and/or stochastic. In a deterministic system, changes in state are completely determined by the previous state. In a stochastic system, transitions from one state

to another have an element of randomness [Pooch and Wall, 1993].

In order to simulate a system, researchers must choose a formalism or approach to model and evaluate the actual system. Each of these approaches determines the potential and limitations of a simulation to represent and evaluate a system. The approaches that can be used to model a system depend upon whether the system has continuous and/or discrete properties.

The first subsection of this section discusses different approaches that researchers have used to simulate various aspects of manufacturing systems with an emphasis directed towards robotic assembly cells. The second subsection summarizes the concepts of the primary approach used to simulate a FABRIC system.

6.1.1 Simulations of Manufacturing Systems

Simulations have been extensively used to study the characteristics of manufacturing systems, which are typically hybrid systems. A key reason for using simulations is that the interactions of a manufacturing system are complex and controlled experimentation is an effective technique to study these interactions.

The simulation of the continuous aspects of a manufacturing system usually concerns the safe and proper operation of the physical equipment. Typically, a solid modelling package provides the foundation of such a simulation. These packages allow the user to build geometric models of the physical equipment. A geometric model is just the starting point for such a continuous simulation. For example in the case of a physical robotic manipulator, a kinematic model is required while a dynamic model and sensor model(s) would significantly improve the quality of this simulation. Differential equations are one type of formalism that are used to model these continuous systems. Researchers that have simulated robotic assembly cells using this approach include [Ranky, 1986b, 1991; Liu and Sadler, 1992]. Also note this type of continuous simulation is typically used to test an off-line program written for a robotic manipulator.

Researchers have used several approaches to model and evaluate the discrete aspects of a

manufacturing system. These approaches include queueing networks, Markov processes, discrete event simulation, perturbation analysis, and Petri nets [Viswanadham and Narahari, 1988; Al-Jaar and Desrochers, 1990].

A few references that used two of these approaches to model and evaluate manufacturing cells as discrete systems are presented. Cash and Wilhelm [1986] simulated a network model of an robotic assembly cell by using the computational techniques of discrete event simulation. Researchers have used a Petri net approach to model the real-time control of a machining cell [Boucher et al., 1989] and to model the control of a computer-integrated assembly cell [D'Souza and Khator, 1993]. Zhou et al. [1993] thoroughly described how to synthesize and analyze a Petri net model for a manufacturing cell.

6.1.2 Approach to Simulate a FABRIC System

Although a FABRIC system is a hybrid system, the system level operation of this system was modelled and evaluated as a discrete system. This choice was deemed justified for four reasons. First, the core operation of the four types of FABRIC agents was defined as a CSM, a graphical model for a discrete system. Second, the discrete temporal behaviour of a FABRIC system was considered the most important because of the variations in time needed to carry out the physical tasks of part assembling and part fetching. Third, FABRIC is a MAS that cooperates using the high-level communication protocol of the contract net [Smith, 1979] and this protocol is verifiable as the simulation of a discrete system. Fourth, one of the goals of our simulation was to develop a prototyping tool to compare alternative FABRIC systems. Building simulations of FABRIC systems as a discrete system are less time consuming than as a hybrid system, since these latter simulations require doing the nontrivial task of building a solid model simulation for each alternative physical system.

The approach that we used to model and evaluate a FABRIC system was discrete event simulation [Mitrani, 1982; Banks and Carson, 1984; Pooch and Wall, 1993]. In discrete event simulation (DES), certain computational techniques are used to program the simulation of a discrete

event system on a computer. A discrete event system is one whose changes in state occur in response to events that happen at specific points in time. There are both advantages and disadvantages to having chosen this approach to model and evaluate a FABRIC system [Banks and Carson, 1984; Viswanadham and Narahari, 1988; Al-Jaar and Desrochers, 1990; Pooch and Wall, 1993]. The primary strength of this approach is also its major weakness. In short, the simulation model can be made as accurate as desired to the real system but this accuracy comes at the expense of greater programming time, greater time to initialize the program, and increased computer time to run the simulation model [Viswanadham and Narahari, 1988]. Even with these disadvantages, this approach was chosen for the sake of accuracy and for the fact that there is a natural parallel between DES and MASs that will be clarified in the next section.

The important concepts of a DES are defined and explained using two references [Banks and Carson, 1984; Pooch and Wall, 1993]. Recall that, a system is a collection of interacting and interdependent entities. An entity is an object of interest that belongs to a particular system and properties of entities are called attributes. The state of a system is the minimal information needed to completely specify the state of a system at any time [Pooch and Wall, 1993]. Changes that occur outside the system but that affect the state of the system are said to occur in the environment of the system. In modelling a system, the boundary between a system and its environment should be decided upon.

An activity is defined as an action that takes a specified duration of time and that can be defined as a statistical distribution, an event is an instantaneous occurrence that can change the state of a system, and a delay is an unspecified duration of time [Banks and Carson, 1984]. The term endogenous refers to activities and events that occur within the system. The term exogenous refers to activities and events that occur within the environment of the system. A system whose state changes are only prompted by endogenous activities and events is known as a closed system. An open system has state changes that are prompted by both endogenous and exogenous, activities and events [Pooch and Wall, 1993].

Verification refers to the process of assessing the correctness with which the simulation model has been implemented by a computer program. Validation refers to the process of determining the accuracy with which the simulation model represents the real system. Having completed the process of verification and validation, the question becomes what performance measure(s) to use to assess the results of the simulations.

Two major techniques of simulating a discrete event system are described. In the event-oriented technique, the computer simulation concentrates on events and their effects on the state of a system [Banks and Carson, 1984]. Each event has an associated procedure that handles the effects on the overall system [Mitrani, 1982]. In the process-oriented technique, the computer simulation revolves around a collection of processes. Each process is related to an entity and it represents a time-ordered group of activities, events, and delays.

The key programming concept used to implement a DES is a future event list [Banks and Carson, 1984]. The future event list stores the events of a system that are scheduled to occur at a future time of the simulation. The traditional algorithm [Banks and Carson, 1984] to maintain the future event list, schedule the events of a DES, and advance the time of a DES is as follows:

1. remove the next event to schedule from the future event list
2. advance simulation time to the time of the event removed in step one
3. execute this event; update the system state and the entities
4. generate future events and add them to the future event list
5. update cumulative statistics and counters
6. if an event remains in the future event list, then go to step one

This algorithm is straightforward except for step five, whose purpose is to collect the information needed to calculate the performance measure(s) chosen to assess the results of the DES.

6.2 Simulating FABRIC Systems

This section discuses how the concepts of a DES were applied to the modelling of FABRIC systems and describes the hypothetical assembly cell that the simulations of this chapter are based upon.

6.2.1 *Concepts of DES applied to FABRIC*

Table 6.1 characterizes a generic FABRIC system as a DES. The headings of the columns of this table correspond with the major concepts of a DES. The major concepts identified in this table are system, entities, attributes, and activities, events, & delays.

The system, which is managed by our framework and is to be simulated, is a MRAC. This system has the properties of being stochastic and closed. The state of the system is defined by collecting the states of all the agents that cooperate in a specific physical setup or configuration of a MRAC. The last point in this column has to do with the observation that there is a natural parallel between the process-oriented view of a DES and the agent-based operation of a system. This parallel should be obvious as the points in the remaining columns are explained.

The entities of the DES are the agents of FABRIC. In the general case, this means that the entities include a TSA, n ASAs, m PPAs, and a SSA. The three attributes of these entities are directly related to the internal structure of the FABRIC agents. First, the type of agent is a static attribute that simply identifies which of the four types of FABRIC agents that an agent is. Second, the specialized knowledge in the knowledge base of each agent is a major attribute of an entity. This knowledge is both static and dynamic. For example in our case, an agent's knowledge of its capabilities is static while an agent's knowledge of its current state is dynamic. Third, the messages that an agent has received in its message queue is a dynamic attribute of an entity.

The activities, events, & delays of the entities are defined by the CSM of each of the four types of FABRIC agents. With respect to these CSMs, the specified durations of the activities for part assembling and part fetching must be specified for each ASA and PPA respectively. The durations of these activities are a major determining factor in the results of any DES.

6.2.2 *Multi-Robot Assembly Cell*

This section describes the hypothetical MRAC that was simulated as a DES. To begin with, there are a multitude of applications / case studies of MRACs that could be studied; however,

System Aspects	Entities	Attributes	Activities, Events, & Delays
• multi-robot assembly cell • system state defined by the states of all agents • process-oriented view of simulation • stochastic and closed system	• agents • task scheduling agent • n assembling agent(s) • m part presentation agent(s) • shared space agent	• type of agent • knowledge base • message queue	• defined by the communicating state machine • duration of part assembling and part fetching activities need to be specified

Table 6.1: Characterization of FABRIC as a DES.

a hypothetical one was chosen for two reasons. First, with a hypothetical cell, it was easier to reconfigure the cell so that different physical setups or configurations of this cell could be analyzed and evaluated. Second, with a hypothetical cell, it was possible to choose it to be as protoypical as possible and at the same time allow us to illustrate the system level behaviour of FABRIC.

Figure 6.1 displays one of the five possible configurations that we considered for the physical setup of a hypothetical car assembly cell. In this configuration, there are two PPAs and three ASAs. These physical agents are responsible for cooperating with each other in order to assemble the axles and the wheels to the subassembly of a car. Each of the PPAs consists of a conveyor belt and two part feeders, one for axles and one for wheels. As shown in the diagram, the two PPAs can only deliver these parts to specific ASAs. In particular, *PPA1* can deliver to *ASA1* and *ASA3* while *PPA2* can deliver to *ASA2* and *ASA3*. The capabilities of the ASAs are discussed later in this section. It is assumed that the car subassembly is moved both into and out of this assembly cell by the transportation network of a cell production system.

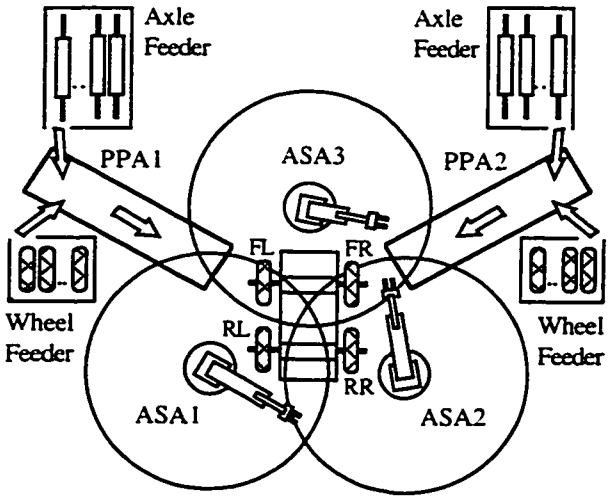
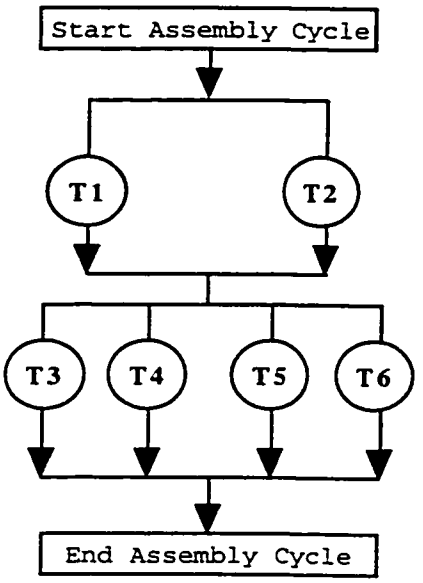


Figure 6.1: A physical setup of the car assembly cell.

Figure 6.2 portrays the task precedence graph for a single assembly cycle of our hypothetical car assembly cell. At the beginning of the assembly cycle, both assembly tasks *T1* and *T2* can be launched. Task *T1* specifies welding an axle to the rear of the car subassembly

while task *T2* specifies welding an axle to the front of the car subassembly. When assembly tasks *T1* and *T2* are completed, then assembly tasks *T3*, *T4*, *T5*, and *T6* can be launched. Tasks *T3* and *T4* specify the assembling of the rear wheels onto the rear axle of a car, while tasks *T5* and *T6* specify the assembling of the front wheels onto the front axle of a car. The basic assembly operation needed to assemble the wheels of a car is a simple peg in hole. When all six tasks are completed, a single cycle of the assembly cell is finished.




```
T1:  weld  axle  rear
T2:  weld  axle  front
T3:  simple_peg_in_hole  wheel  rear_left
T4:  simple_peg_in_hole  wheel  rear_right
T5:  simple_peg_in_hole  wheel  front_left
T6:  simple_peg_in_hole  wheel  front_right
```

Figure 6.2: Task precedence graph of the assembly operations.

Table 6.2 shows the essentials of the five configurations that we simulated and compared for the hypothetical car assembly cell. The first column of this table simply has a number that is used to identify a specific configuration of this assembly cell. For instance, the physical setup that was displayed in figure 6.1 corresponds to configuration number 4 in this table. These identifying numbers are used in the remainder of this chapter. The second column of this table indicates the

number of PPA(s) and ASA(s) in each configuration and specifies which ASA(s) each PPA can deliver to with arrows. For example in configuration 5, *PPA3* can deliver to *ASA1* and *ASA2*. The last column indicates the capabilities of the ASA(s) in a particular configuration to carry out the assembly tasks shown in the precedence graph of figure 6.2. For example in configuration 5, *ASA3* has the capability to perform assembly tasks *T2*, *T5*, and *T6*.

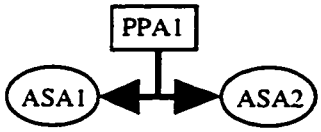
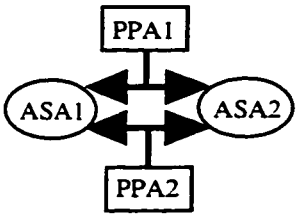
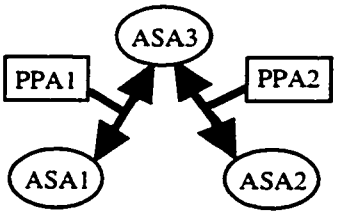
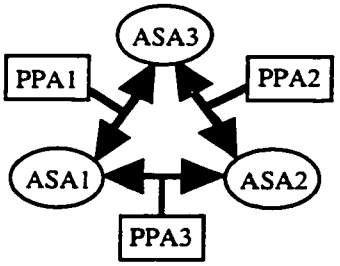
Identifying Number	Configuration of assembly cell	Capabilities of ASAs
1		ASA1: T1-T6
2		ASA1: T2, T5, T6 ASA2: T1, T3, T4
3		ASA1: T2, T5, T6 ASA2: T1, T3, T4
4		ASA1: T1, T3, T5 ASA2: T1, T4, T6 ASA3: T2, T5, T6
5		ASA1: T1, T3, T5 ASA2: T1, T4, T6 ASA3: T2, T5, T6

Table 6.2: Different configurations of the car assembly cell.

6.3 Aspects of the Software Implementation

This section discusses aspects of the software implementation of a DES of FABRIC systems using the context of the hypothetical car assembly cell. This section presents the software implementation of the class *Simulator* and of one configuration of this assembly cell. The UML notation, which was described in subsection 4.2.2, is used to present the details of the object-oriented software implementation. Also note that the implementation described in this section was implemented on a computer system with a single processor.

6.3.1 The Class Simulator

The class *Simulator* and its friend relationship with the class *Agent* was previously indicated in figure 4.2. Figure 6.3 displays the core structure and behaviour of the class *Simulator*. Three private variables define the contents of this class. The variable *_the_agents* is a hash table that stores all of the agents that are simultaneously executing in the current simulation. The variable *_the_event_list* implements the future event list of the DES. This future event list was implemented as a singly linked list that stores the discrete events in the order of time that they will occur. In our case, each event contains the time of the event, the agent to which the event belongs, and the type of event. The variable *_the_simulation_time* simply stores the current time of the DES.

The private methods of the class *Simulator* fall into two groups. The first group consists of three methods that all simulations of FABRIC systems invoke. The first method of this group, called *Add_Agent_to_Simulation*, adds an agent to the hash table of agents maintained by this class. The next two methods of this group define what to do just before and just after the execution of a single simulation run. The second group consists of six methods that can create the different configurations of our car assembly cell. Specifically, the first five methods that begin with *Create_FABRIC_Configuration_* create each of the five configurations of the car assembly cell that were identified in table 6.2. The number at the end of these methods identifies the specific configuration that is created. The sixth method *Create_Agents_of_Simulation* defines a simple

textual interface that allows the user of the DES to select which configuration of the car assembly cell to create.

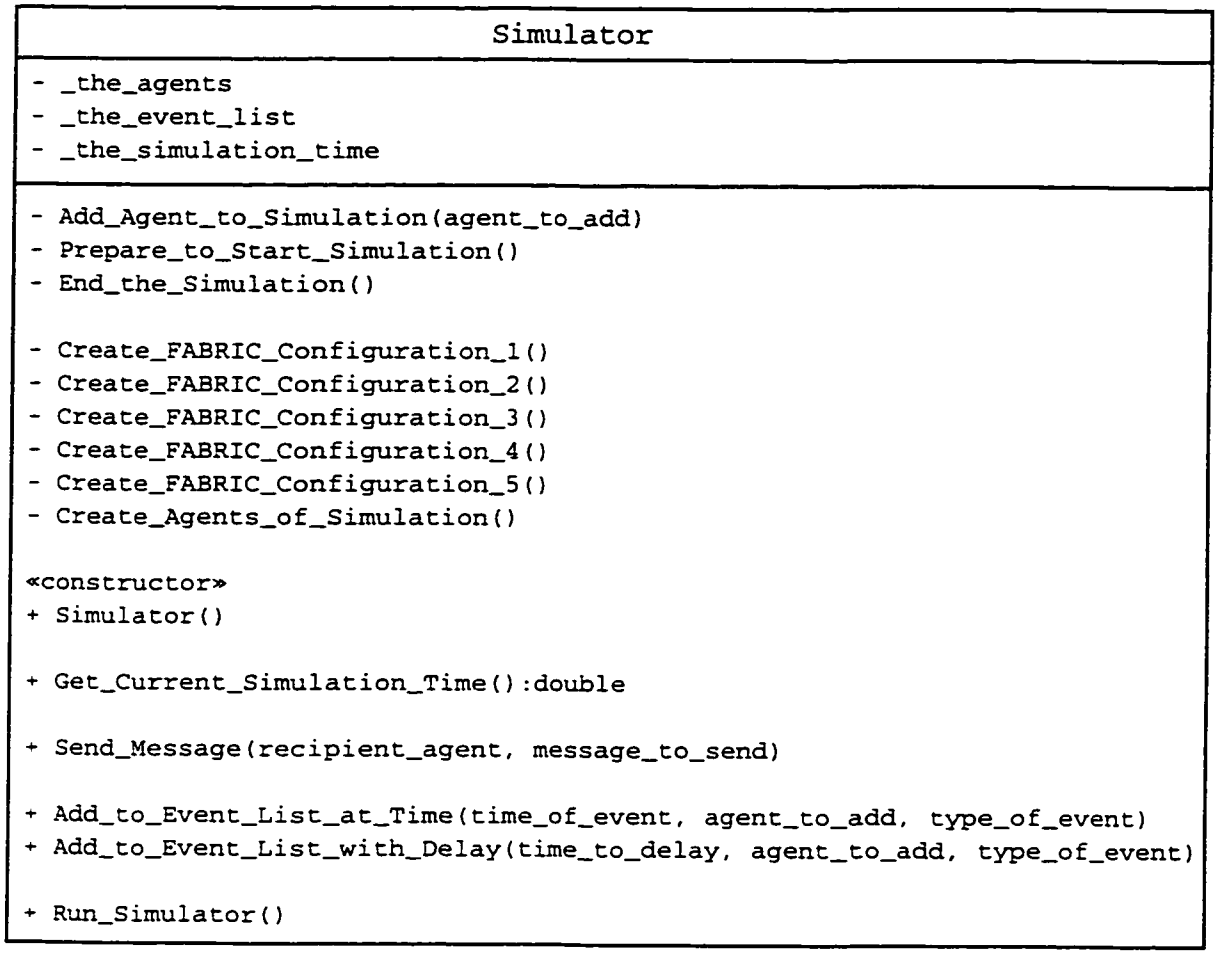


Figure 6.3: Core structure and behaviour of the class *Simulator*.

The public constructor and public methods of the class *Simulator* are described. The constructor of the class *Simulator* creates the agents of a DES by calling the method *Create_Agents_of_Simulation*. The method *Get_Current_Simulation_Time* as its name suggests simply returns the current time of the simulation to the agent that requested it. Given the name of a recipient message and the message to send, the method *Send_Message* is used by the class *Agent* to send a message from one agent to another agent. Since the class *Simulator* stores all of the agents of a simulation run, it only sends a message if the agent referred to is part of the current

simulation. The method *Add_to_Event_List_at_Time* adds an event with the specified parameters to the future event list of a DES. The method *Add_to_Event_List_with_Delay* adds an event delayed the specified amount from the current simulation time to the future event list of a DES.

The public method *Run_Simulator* executes a single run of the simulation. The specific algorithm used in this method is as follows:

1. remove the next event from the future event list
2. advance simulation time to the time of the event removed in step one
3. call the method *Execute* of the agent to which this event belongs with the parameter type of event
4. if an event remains in the future event list, then go to step one

This algorithm differs from the traditional algorithm [Banks and Carson, 1984] described in subsection 6.1.2. The basic difference is that a core part of the operation of this DES has been redistributed to the internal operation of an agent. Each agent of a simulation is responsible for adding its own events to the future event list of the DES by using the public methods of the class *Simulator* to add events to a simulation run. Each agent also assumes responsibility for collecting statistics regarding its behaviour locally. These statistics are used to calculate the performance measure(s) for an agent. The basic approach is that each agent creates a file to store both a trace of its execution and its statistics and/or performance measures(s). With respect to generating different random distribution functions and collecting the statistics for our DES, we ported some C++ program code from a public domain package [C++SIM, 1994].

Just a note that the implementation of a DES using object-oriented technology and its application to the field of manufacturing has been studied previously by researchers such as [Atabakhsh and Chan, 1992; Tchako et al., 1994]. Moreover, Simula, a programming language originally developed for simulation, was one of the first object-oriented programming languages [Masini et al., 1991].

6.3.2 Software Implementation of a Configuration

This subsection gives a brief overview of the software implementation of configuration 4 of

the car assembly cell. Figure 6.4 displays the class hierarchy of the agents with the seven agent classes added to implement this configuration shown in a bold border. The agent classes *TSA_Car_Assembly* and *SSA_Car_Assembly* are used for all configurations of the car assembly cell. The agent class *TSA_Car_Assembly* sets up the task precedence graph for the assembly operations as shown in figure 6.2. The agent class *SSA_Car_Assembly* sets up all of the regions of space that have the potential of being shared amongst the ASAs.

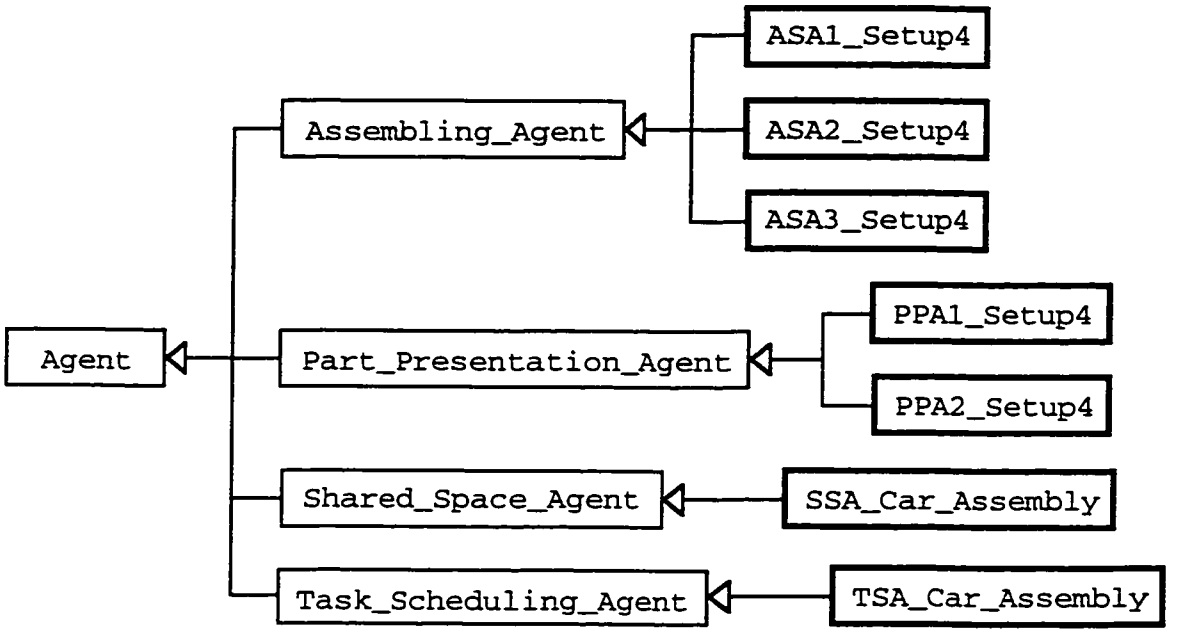


Figure 6.4: Class hierarchy of the agents for configuration 4 of the car assembly cell.

The remaining five agent classes define and implement the capabilities of the five physical agents of this configuration in software. There is a clear parallel between the class hierarchy of agent classes and the physical setup of a MRAC. In particular, the classes, *ASA1_Setup4*, *ASA2_Setup4*, *ASA3_Setup4*, *PPA1_Setup4*, and *PPA2_Setup4*, define and implement the capabilities of the physical agents *ASA1*, *ASA2*, *ASA3*, *PPA1*, and *PPA2* as indicated in figure 6.1 and table 6.1. With these agents, the question of how to behave is inherited from the parent classes, but the question of what behaviour is possible is specialized by the knowledge added to the knowledge base by each of these specific classes.

6.4 Simulation Results

This section presents the results of simulating different configurations of the hypothetical car assembly cell using two basic probability distributions to model the variability in time to execute the physical operations of FABRIC. This section is divided into three subsections. The first subsection discusses background issues of the DES. The next two subsections present the simulation results using these two distributions. The subject of comparing alternative system configurations is deferred until the next section.

6.4.1 Background Issues

This subsection discusses three issues of a DES in relation to our simulation: probability distributions, verification and validation, and performance measures.

The two probability distributions that were used to model the variability in time were the exponential and normal distributions. The exponential distribution is defined by:

$$f(x) = \lambda \bullet e^{-\lambda x},\ x \geq 0 \text{ and } \lambda \geq 0$$

with mean = $1/\lambda$ and variance = $1/\lambda^2$

This distribution was used to model the case where the times to execute the physical operations are highly variable. The normal distribution is defined by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \bullet e^{-(x-\mu)^2/2\sigma^2},\ -\infty < x < \infty$$

with mean = μ and variance = σ^2

The standard deviation of this normal distribution is simply σ. The specific parameters used in these distributions for the simulations are presented in the following two subsections.

The verification that our computer program correctly implemented the DES model of FABRIC was ascertained through two ways. First, the files that traced the execution of the individual agents were examined to verify certain aspects of the proper operation of this system. For instance, the satisfactory exchange of messages during contract net negotiation [Smith, 1979] was verified in this way. These trace files were also used to debug problems encountered during

simulation runs of various system configurations. Second, the simulator of FABRIC ran successfully for several thousands of assembly cycles under various operating circumstances. The validation of the DES program with the real system was not possible because of our decision to simulate a hypothetical assembly cell. However, our results are consistent with previous non-agent-based simulations of the assembly system using Petri nets [Petriu et al., 1993] and CSMs [Petriu et al., 1994]. The differences between our DES program and these prior simulations are discussed in subsection 6.4.3.

Two questions are addressed regarding performance measures. The first is what performance measures are used to evaluate a FABRIC configuration. The second is how to measure them properly in a DES.

Although many factors contribute to the total productivity of a robotic assembly system [Rampersad, 1995b, 1996], two basic measures of performance were calculated for different FABRIC configurations. The first is the time that it takes the MRAC to complete a single cycle of production. In our case, this cycle time is the time between two successive starts of beginning to carry out the six assembly tasks that are shown in the precedence graph of figure 6.2. This cycle time is inversely proportional to the throughput or production rate. The second performance measure is the percentage of time during a production run that the equipment or physical agents are being utilized. In particular, each ASA and PPA is considered busy from the point that they are granted a task until they return to the idle state of their CSM.

The method that we used to estimate these performance measures from simulations of FABRIC is described using the reference [Banks and Carson, 1984]. In particular, this method estimates a point value and a confidence interval for a performance measure, from initialization bias free and statistically independent observations of this measure during a steady-state simulation. To begin with, it is assumed that the simulation program was ran in order to estimate a performance measure n times, Y_1, Y_2, ..., Y_n. Then according to [Banks and Carson, 1984] the following equations can be defined:

$\hat{\theta}$ = the point estimate of a performance measure

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} Y_i$$

S = the standard deviation of the estimates of a performance measure

$$S^2 = \sum_{i=1}^{n} \frac{(Y_i - \hat{\theta})^2}{n-1}$$

$\hat{\sigma}(\hat{\theta})$ = the standard error of a point estimate

$$\hat{\sigma}(\hat{\theta}) = \frac{S}{\sqrt{n}}$$

f = degrees of freedom of a t-test

α = level of significance for a two-sided t-test

$t_{\alpha/2, f}$ = value obtained from a t-test table with the specified parameters

w = half the width of a confidence interval

$$f = n - 1$$

$$w = t_{\alpha/2, f} \bullet \hat{\sigma}(\hat{\theta})$$

θ = the actual performance measure

$$\hat{\theta} - w \leq \theta \leq \hat{\theta} + w$$

The inequality equation at the end of these definitions is the one that specifies the confidence interval of a performance measure. The basic procedure is relatively straightforward. First, calculate the mean, standard deviation, and standard error of the n estimates of a performance measure. Then choose a level of significance, look up the t-test value needed to claim this level of significance, and finally use this value in combination with the standard error of a point estimate in order to calculate the confidence intervals of a performance measure. For explanations of the t-test, refer to [Kennedy and Neville, 1974; Spiegel, 1975].

6.4.2 Results with Exponential Distributions

This subsection begins by introducing the basic parameters used in the exponential distributions, presenting the convergence characteristics of a configuration, and illustrating the

calculation of the performance measures for a single configuration. This subsection continues by presenting the simulation results for all five configurations of the car assembly cell in both tabular and graphical format.

The parameters that all simulations of this dissertation have in common are described. Note that to keep the discussion general the dimensions of parameters that represent time are in generic units of time. One of the key parameters of a simulation run is the number of cycles for which the discrete event operation of an assembly cell is simulated. Each simulation run that was used to estimate the performance measures of a FABRIC configuration consisted of 100 cycles. In contrast to the times for the mechanical operations of a FABRIC simulation, the times for the electronic operations are the same for all simulations. The delay in sending a message from one agent to another agent was modelled as a normal distribution with $\mu = 0.10$ and $\sigma = 0.02$. With respect to the contract net, the time that a manager agent waits for bids from contractor agents is another key parameter. For all manager agents and simulations runs, this waiting time was set to 0.5 units of time.

Description of Mechanical Operation	Mean of Exponential Distribution
a PPA fetches a part for an ASA	4
an ASA picks up a fetched part	1
an ASA moves to an assembly location	1
an ASA executes task T1 or T2	6
an ASA executes task T3, T4, T5, or T6	4
resetting the assembly cell for the next cycle	1

Table 6.3: List of the mechanical operations and the means of their exponential distributions.

Table 6.3 shows the means of the exponential distributions that were used to model the variability in time of the different mechanical operations of our hypothetical car assembly cell. Only the means are indicated since the mean defines the variance and the parameter λ of an exponential distribution. For a PPA, the only mechanical operation is the time taken to fetch a part

for an ASA, which is assumed to be the same for all parts. In all of our simulations, it is assumed that each PPA physically delivers a part to an ASA using a conveyor belt and therefore does not need a pickup message as indicated in its CSM. For an ASA, the minor mechanical operations are picking up a part and moving to an assembly location, while the major mechanical operation is carrying out one of the assembly operations specified in figure 6.2. The last item in this list represents the time taken for the completed car subassembly to be moved out and the next uncompleted car subassembly to be moved into this assembly cell.

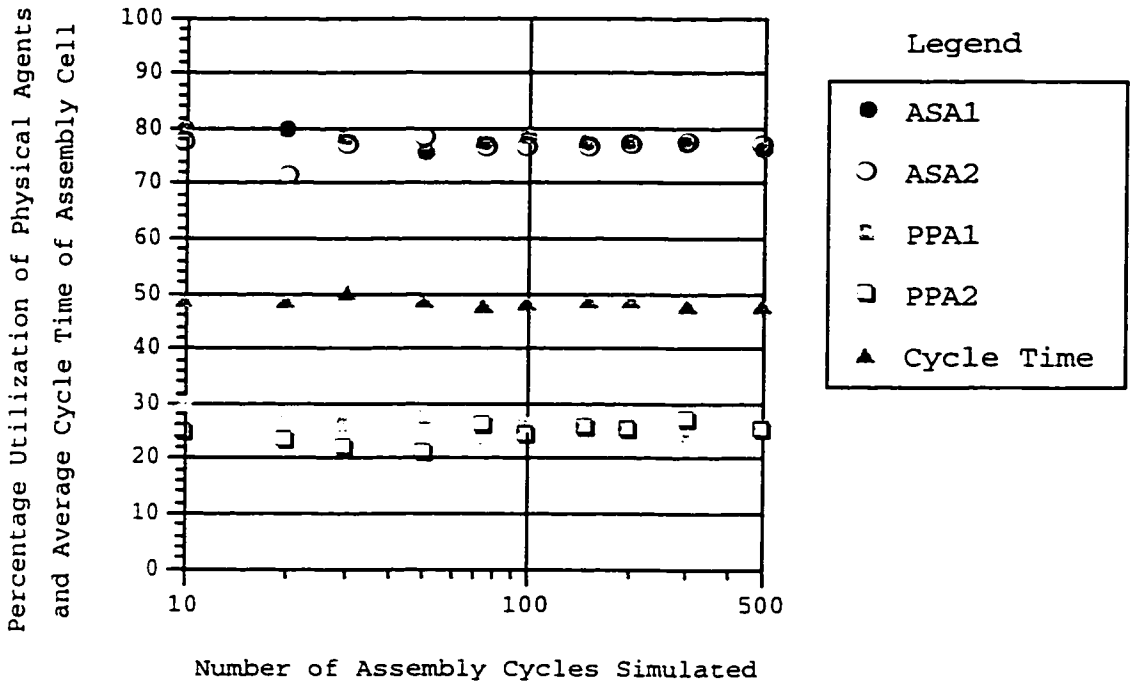


Figure 6.5: Estimates of the performance measures of configuration 3 using different cycles.

The matter of how the estimates of the performance measures vary with the number of cycles that are simulated during a simulation run is addressed. Figure 6.5 displays a plot of the estimates of the performance measures of configuration 3 of our assembly cell when the number of assembly cycles was varied from 10 to 500. In this diagram, the percent utilization of the physical agents — *ASA1, ASA2, PPA1, and PPA2* — during a complete simulation run are plotted against the number of cycles simulated. The average time needed to complete a single cycle of production

is also plotted against the number of cycles simulated. As can be seen, the estimates of these performance measure are quite stable and do converge.

Simulation Run	Average Assembly Cycle Time	Percent Utilization of Physical Agents				
		ASA1	ASA2	ASA3	PPA1	PPA2
1	45.137	54.92	54.02	55.83	25.68	26.37
2	45.014	51.55	57.35	55.59	25.68	29.20
3	45.669	54.96	56.93	55.38	29.53	26.97
4	45.831	57.30	53.17	50.02	27.17	26.64
5	45.029	57.83	50.66	55.51	24.30	25.60
6	48.395	52.23	54.12	56.03	26.95	25.32
7	42.544	58.34	54.02	56.30	27.14	24.20
8	44.288	54.22	53.52	54.20	24.21	24.73
9	45.626	53.36	56.66	58.11	27.07	25.12
10	44.997	57.25	54.56	53.74	26.67	26.63

Table 6.4: Ten estimates of the performance measures for configuration 4.

The calculations to determine the confidence interval of a performance measure are illustrated. Table 6.4 shows the results of estimating the performance measures of configuration 4 of our hypothetical car assembly cell during ten simulation runs using the means for the exponential distributions presented in this subsection. Each simulation run consisted of 100 assembly cycles. More specifically, the confidence interval of the percent utilization of *ASA1* is calculated. The first steps are to calculate the mean, standard deviation, and the standard error of this performance measure. The results of these calculations are as follows:

$$\hat{\theta}_{\text{percent utilization of ASA1}} = 55.20$$

$$S_{\text{percent utilization of ASA1}} = 2.40$$

$$n_{\text{percent utilization of ASA1}} = 10$$

$$\hat{\sigma}(\hat{\theta})_{\text{percent utilization of ASA1}} = 0.76$$

The next steps are to choose a level of significance, look up the t-test value needed to claim this level of significance, and finally calculate the confidence interval. A level of significance of 5% was chosen to calculate the confidence intervals for all performance measures. With $\alpha = 5$ and the number of degrees of freedom, f=9, then the value of $t_{\alpha/2,\, f}$ from the t-test table is 2.262. The final results of the confidence interval calculations are as follows:

$$w_{\text{percent utilization of ASA1}} = 1.72$$

$$53.48 \leq \theta_{\text{percent utilization of ASA1}} \leq 56.92 \text{ or}$$

$$\theta_{\text{percent utilization of ASA1}} = 55.20 \pm 3.1\%$$

Thus, with a 95% confidence level the percent utilization of *ASA1* is between 53.48 and 56.92. The last result in this series expresses the confidence interval as a percentage of the point estimate of a performance measure. This percentage way to express the confidence interval of a performance measure is used to present the performance measures of all five configurations of the hypothetical car assembly cell. The point estimates and confidence intervals of the performance measures are calculated in this way for all simulations.

Table 6.5 summarizes the results of the DES for the five different physical configurations when using exponential distributions to model the temporal variability of executing mechanical operations. The first column identifies the configuration according to the numbers that were used in table 6.2. The remaining seven columns display the 95% confidence intervals of the performance measures for each of these configurations as a percentage of their point estimates. In particular, the second column displays the average cycle time of this assembly cell for each configuration. The remaining six columns display the percentage utilization of the physical agents in each configuration. A blank value in these six columns indicates that this physical agent was not part of a particular configuration.

Figure 6.6 graphically displays the cycle time of the car assembly cell versus its five physical configurations for exponential distributions. The two smaller dashes on the vertical line

Number of Config.	Average Cycle Time	Percentage Utilization of the Physical Agents					
		ASA1	ASA2	ASA3	PPA1	PPA2	PPA3
1	74.87 ± 2.1%	95.0 ± 0.1%			32.1 ± 2.3%		
2	51.38 ± 2.2%	80.7 ± 1.6%	75.9 ± 1.6%		46.8 ± 1.6%		
3	47.27 ± 1.0%	76.1 ± 1.0%	75.1 ± 1.5%		25.1 ± 3.3%	25.7 ± 3.2%	
4	45.25 ± 2.3%	55.2 ± 3.1%	54.5 ± 2.7%	55.1 ± 2.8%	26.4 ± 4.2%	26.1 ± 3.9%	
5	43.35 ± 1.6%	53.5 ± 2.7%	54.0 ± 3.5%	57.2 ± 2.4%	18.2 ± 4.6%	18.5 ± 5.5%	19.0 ± 5.1%

Table 6.5: Summary of the simulation results with exponential distributions for the mechanical operations.

segments indicate the 95% confidence interval of the average cycle time of this cell. The two longer dashes that are placed at the ends of the vertical line segments are located one standard deviation to either side of the average cycle time. This standard deviation is the average of the standard deviations of the cycle times calculated during the ten simulation runs. The large variances of the cycle times clearly reflect the fact that the exponential distributions were used to model the times to carry out the mechanical operations. At the same time, the average cycle time decreases in nonuniform steps from one configuration to the next.

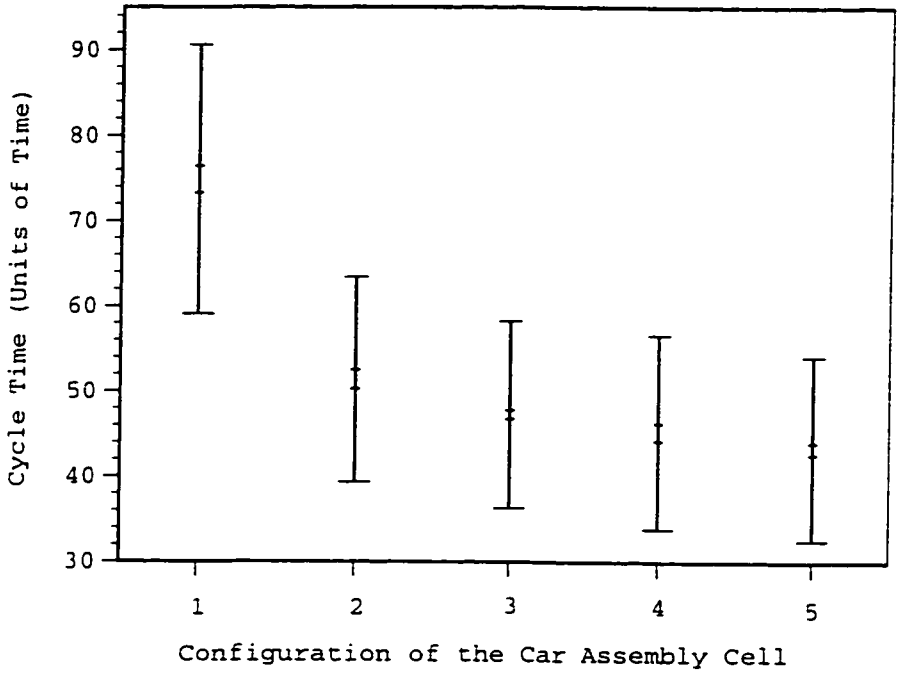


Figure 6.6: Cycle time statistics for each configuration when using exponentially distributed times.

Figure 6.7 graphically plots the percentage utilization of the physical agents of the car assembly cell versus its five physical configurations for exponential distributions. The symbols in this figure are located at the point estimates of these percent utilizations. The legend of this plot specifies which symbols correspond to which physical agent. Symbols of the same physical agent are connected together with line segments. An observation from this plot is that the percent utilization of all ASAs and PPAs for a particular configuration are approximately the same. The

reason for this similarity is that in each configuration of our hypothetical assembly cell these two types of FABRIC agents respectively have comparable capabilities. That is, each ASA or PPA has an identical theoretical likelihood to be randomly selected by a manager agent for performing a task.

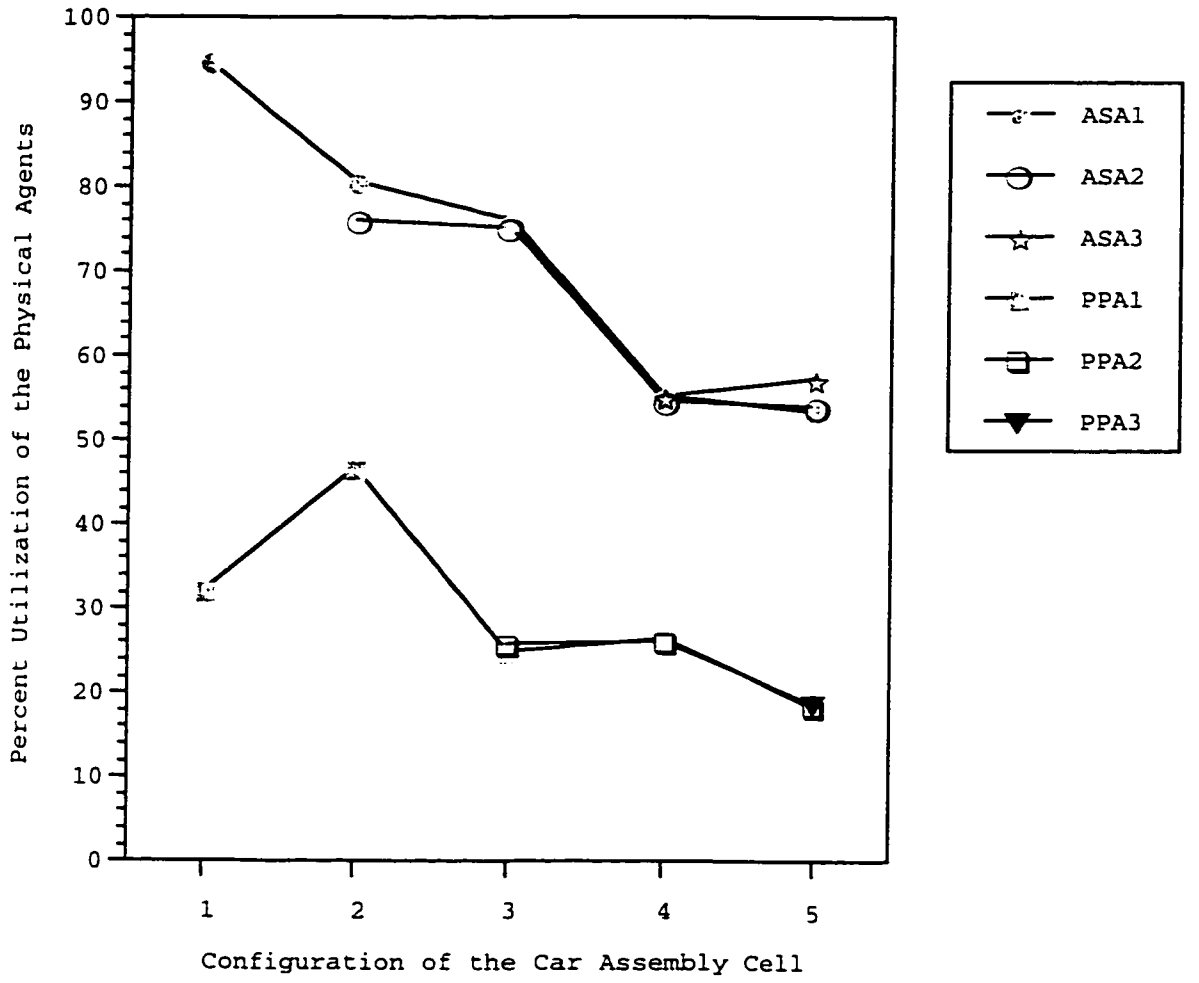


Figure 6.7: Physical agent utilization for different configurations when using exponentially distributed times.

6.4.3 Results with Normal Distributions

Having established the overall characteristics of our simulations in the previous subsection, this subsection presents the simulation results when normal distributions instead of exponential

distributions are used to model the variability in time of the different mechanical operations.

Table 6.6 shows the mean and standard deviation of the normal distributions that were used to model this variability in time. The means of all these normal distributions are the same as the previous exponential distributions. The standard deviations of all these normal distributions are simply calculated to be 10% of the value of their respective means. The descriptions of the mechanical operations are the same as table 6.3.

Description of Mechanical Operation	Mean and Standard Deviation of Normal Distributions
a PPA fetches a part for an ASA	$\mu = 4, \sigma = 0.4$
an ASA picks up a fetched part	$\mu = 1, \sigma = 0.1$
an ASA moves to an assembly location	$\mu = 1, \sigma = 0.1$
an ASA executes task T1 or T2	$\mu = 6, \sigma = 0.6$
an ASA executes task T3, T4, T5, or T6	$\mu = 4, \sigma = 0.4$
resetting the assembly cell for the next cycle	$\mu = 1, \sigma = 0.1$

Table 6.6: List of the mechanical operations and the parameters of their normal distributions.

The conventions of how the simulation results are presented in tabular and graphical format that were described in the previous subsection are used again in this subsection. Specifically, table 6.7, figure 6.8, and figure 6.9 follow the conventions of table 6.5, figure 6.6, and figure 6.7 respectively.

Table 6.7 presents the 95% confidence intervals of the performance measures for the five different physical configurations when using the normal distributions specified in table 6.6. Figure 6.8 graphically displays the cycle time of the car assembly cell versus its five physical configurations for normal distributions. As expected, the confidence intervals and the standard deviations of the cycle times have much less variance than the simulation results with exponential distributions. In this case, however, the cycle time does not decrease from every configuration to the next. In particular, both the mean and variance of the cycle time increase from configuration 3 to 4.

Number of Config.	Average Cycle Time	Percentage Utilization of the Physical Agents					
		ASA1	ASA2	ASA3	PPA1	PPA2	PPA3
1	74.09 ± 0.2%	94.9 ± 0.03%			32.5 ± 0.2%		
2	47.09 ± 0.2%	86.3 ± 0.6%	81.3 ± 0.7%		51.0 ± 0.2%		
3	39.94 ± 0.2%	89.9 ± 0.2%	89.1 ± 0.2%		30.1 ± 0.4%	30.1 ± 0.4%	
4	41.13 ± 0.5%	60.4 ± 2.9%	61.6 ± 3.0%	59.9 ± 1.9%	29.2 ± 2.8%	29.1 ± 3.3%	
5	39.21 ± 0.4%	61.9 ± 2.7%	59.8 ± 2.1%	62.0 ± 1.3%	19.9 ± 4.6%	19.8 ± 5.2%	21.6 ± 3.5%

Table 6.7: Summary of the simulation results with normal distributions for the mechanical operations.

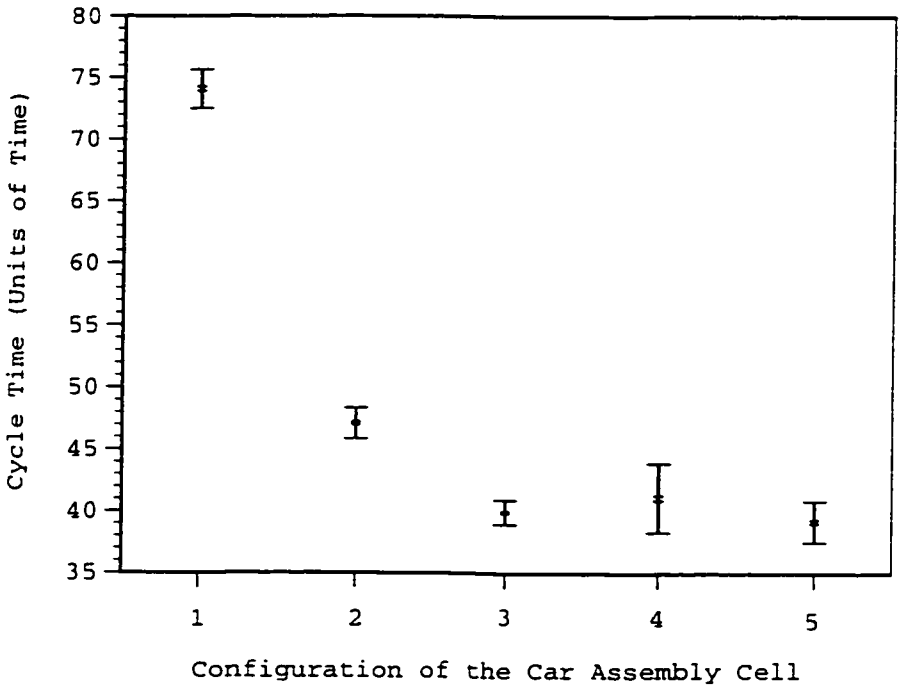


Figure 6.8: Cycle time statistics for each configuration when using normally distributed times.

Figure 6.9 graphically plots the percentage utilization of the physical agents of the car assembly cell versus its five physical configurations for normal distributions. With one exception, the shape of this plot corresponds with the plot based upon exponential distribution times. From configuration 2 to 3, the percent utilization of the ASAs slightly decreases with the exponential distributions, while it slightly increases with the normal distributions.

The question of how the DES of FABRIC systems compares with the previous simulations of MRAC systems [Petriu et al., 1993, 1994] is addressed. Table 6.8 compares the major characteristics of these simulations. The first three points in the two columns are identical for both simulations. That is, the probability distributions that model variability of mechanical operation times, the five physical configurations, and the performance measures used, are the same for these simulations. The last four points indicate differences between these simulations. A key difference is that the components of the MRAC are now agents of FABRIC with a CSM, a knowledge base,

and a message queue. The discrete event behaviour of these agents was completely simulated in software. Second, the detailed description and implementation of the contract-net messages to announce and grant tasks to the contractor agents of the FABRIC is another distinction. A third fundamental difference is that the electronic delays were not assumed to be negligible in the agent-based DES. Lastly, the space conflicts are handled by the SSA in FABRIC simulations.

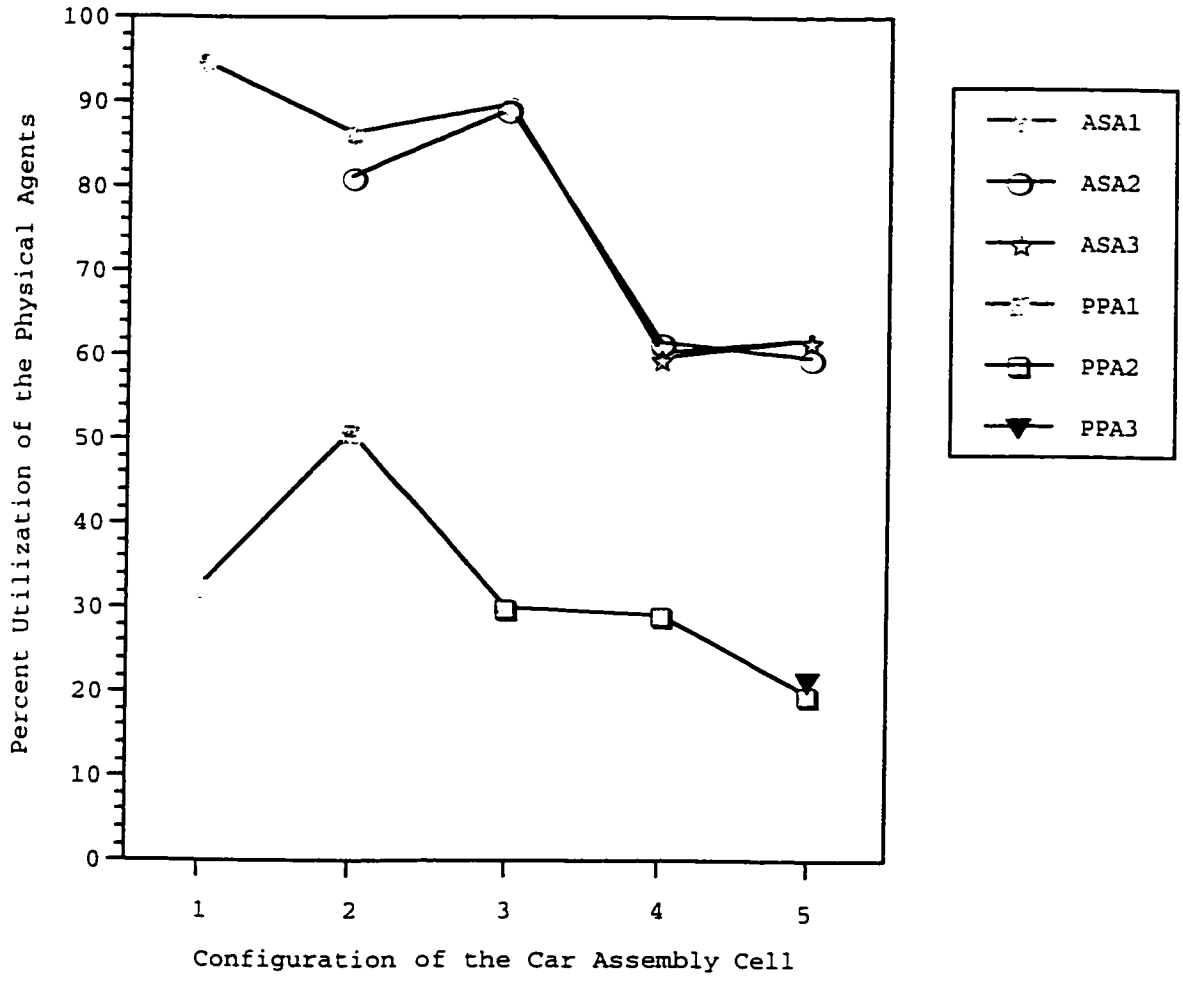


Figure 6.9: Physical agent utilization for different configurations when using normally distributed times.

Previous Simulations of the Multi-Robot Assembly Cell System	Agent-based DES of the Multi-Robot Assembly Cell System
exponential and normal distribution times	exponential and normal distribution times
5 physical configurations simulated	5 physical configurations simulated
performance measures: cycle time and equipment utilization	performance measures: cycle time and equipment utilization
components of the MRAC have a CSM	*agents have a CSM plus a knowledge base and a message queue*
choose potential contractor(s) for a task based upon global availability and select specific contractor randomly	*potential contractor(s) obtained via contract-net messages and specific contractor can be selected randomly or based upon qualifications*
only mechanical execution times are modelled; all electronic delays are assumed to be negligible	*electronic operations also have delays; for example, incorporate message delays into simulation*
space conflicts handled by token availablity	*space conflicts handled by shared space agent*

Table 6.8: Comparison of the previous simulations with the current agent-based DES.

6.5 Comparison of FABRIC Systems

This section is partitioned into three subsections. The first subsection introduces the topic of comparing alternative systems using DES and then discusses how the different configurations of the car assembly cell compare with each other. This comparison is done using the simulation results of subsections 6.4.2 and 6.4.3. The second subsection presents a detailed example of comparing the identical configurations of this assembly cell under slightly different operating

circumstances. The third subsection gives a general closing discussion on all of the simulation results.

6.5.1 Introduction and Comparison of Different Configurations

A straightforward technique to compare two systems is to compare the confidence intervals of the same performance measure [Banks and Carson, 1984]. If the confidence intervals of this performance measure do not overlap, then the hypothesis that one system is better than the other system with respect to this measure is strongly statistically supported. If the confidence intervals of this performance measure do overlap, then the differences between these systems does not have strong statistical support [Banks and Carson, 1984]. In this latter case, the degree of confidence in the differences between these two systems could be statistically calculated based upon a normal distribution test or t-test [Kennedy and Neville, 1974; Spiegel, 1975]. The normal distribution test can be used if the size of the sample is greater than or equal to thirty; otherwise, the t-test should be used.

The question now becomes the confidence intervals of which performance measures to check for overlap when comparing different FABRIC configurations. The two performance measures that we estimated in our simulation runs were average cycle time and the percent utilization of the physical agents of a configuration. Since only these two performance measures were estimated, we use both of them to compare different FABRIC configurations.

Examples of comparing different configurations of the car assembly cell that were simulated with exponential distribution times are given using the performance measures, which are summarized in table 6.5. One interesting question is does each successive configuration from one to five decrease the average cycle time of this assembly cell? In fact, the adjacent confidence intervals of this performance measure do not overlap for all five configurations. The five successive configurations statistically decrease the average cycle time of this assembly cell. Another potential question is whether or not the percent utilization of *ASA3* has improved from configuration 4 to 5. The percent utilization changed from 55.1 ± 2.8% in configuration 4 to

57.2% ± 2.4% in configuration 5. The confidence intervals of this performance measure do overlap; therefore, this change in configuration does not represent a statistically significant improvement in the utilization of *ASA3*.

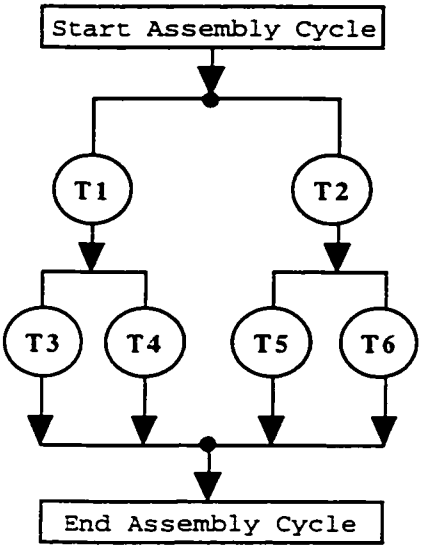
The comparative questions that were answered with exponential distributions times are now answered for the case where the car assembly cell was simulated with normal distributions times. In this case, the performance measures summarized in table 6.7 are used. Four of the five successive configurations statistically decrease the average cycle time under these operating conditions. The single exception is that the confidence intervals of the average cycle time of configuration 3 are statistically better than configuration 4. Thus, adding *ASA3* in configuration 4, on average, increases the cycle time of our assembly cell. At the same time, the confidence interval of configuration 5 is still statistically better than the interval of configuration 3. With respect to the percent utilization of *ASA3*, it changed from 59.5 ± 1.9% in configuration 4 to 62.0 ± 1.3% in configuration 5. These confidence intervals do not overlap; therefore, this change in configuration is a statistically significant improvement in its utilization.

6.5.2 Comparison of Identical Configurations

The task precedence graph of assembly operations is a key input of a FABRIC system. This graph determines the degree of potential parallelism in the operation of a particular FABRIC configuration. This subsection presents a task precedence graph that allows for greater parallelism than the original one described in subsection 6.2.2. This greater parallelism should improve the performance of FABRIC configurations with multiple robotic manipulators the most. The simulation results for this changed graph but the exact same exponentially and normally distributed times for the mechanical operations are presented. These results are used in conjunction with the results already summarized in table 6.5 and table 6.7 in order to compare identical configurations of the car assembly cell.

Figure 6.10 displays the changed task precedence graph for a single assembly cycle of our hypothetical car assembly cell. This graph has two parallel branches with identical structures.

Each branch concerns assembling an axle to a car subassembly before assembling the wheels onto this axle. At the beginning of the assembly cycle, both assembly tasks *T1* and *T2* can be launched. When assembly task *T1* is completed, then assembly tasks *T3* and *T4* can be launched. When assembly task *T2* is completed, then assembly tasks *T5* and *T6* can be launched. The primary difference with the previous task precedence graph is that assembly tasks *T1* and *T2* do not both have to be completed before all of the remaining four assembly tasks can be launched. When all six tasks are completed, a single cycle of the assembly cell is finished.




```
T1:  weld  axle  rear
T2:  weld  axle  front
T3:  simple_peg_in_hole  wheel  rear_left
T4:  simple_peg_in_hole  wheel  rear_right
T5:  simple_peg_in_hole  wheel  front_left
T6:  simple_peg_in_hole  wheel  front_right
```

Figure 6.10: Task precedence graph of the assembly operations with greater parallelism.

Using the same conventions as table 6.5 and table 6.7, table 6.9 and table 6.10 summarize the simulation results with the changed task precedence graph for exponentially and normally distributed times respectively for the mechanical operations.

Number of Config.	Average Cycle Time	Percentage Utilization of the Physical Agents					
		ASA1	ASA2	ASA3	PPA1	PPA2	PPA3
1	74.57 ± 2.0%	95.25 ± 0.1%			32.55 ± 2.3%		
2	48.01 ± 1.3%	81.52 ± 0.7%	83.01 ± 1.1%		49.89 ± 3.5%		
3	44.36 ± 2.2%	78.39 ± 1.5%	80.63 ± 1.1%		27.45 ± 4.4%	26.87 ± 5.2%	
4	39.62 ± 1.3%	61.51 ± 3.6%	60.28 ± 3.2%	63.41 ± 2.2%	30.54 ± 3.9%	29.18 ± 2.7%	
5	38.01 ± 2.3%	59.25 ± 2.6%	60.28 ± 3.2%	66.25 ± 1.8%	21.35 ± 6.4%	21.31 ± 4.0%	20.65 ± 2.8%

Table 6.9: Summary of the simulation results with exponential distributions and a changed task precedence graph.

Number of Config.	Average Cycle Time	Percentage Utilization of the Physical Agents					
		ASA1	ASA2	ASA3	PPA1	PPA2	PPA3
1	73.85 ± 0.1%	95.25 ± 0.03%			32.56 ± 0.2%		
2	42.94 ± 0.1%	87.47 ± 0.6%	87.52 ± 0.4%		55.94 ± 0.3%		
3	39.16 ± 0.1%	90.69 ± 0.2%	90.62 ± 0.2%		30.66 ± 0.4%	30.61 ± 0.5%	
4	39.15 ± 0.3%	63.83 ± 2.4%	63.36 ± 1.9%	62.30 ± 1.5%	30.82 ± 1.8%	30.54 ± 1.8%	
5	37.92 ± 0.3%	62.06 ± 2.0%	63.35 ± 2.1%	63.40 ± 1.6%	21.50 ± 3.0%	20.70 ± 3.0%	21.14 ± 3.7%

Table 6.10: Summary of the simulation results with normal distributions and a changed task precedence graph.

Figure 6.11 displays the cycle time of our assembly cell versus its five physical configurations for both task precedence graphs and using exponential distribution times. The vertical line segments with the two small dashes at their ends signify the 95% confidence intervals of the average cycle time of each configuration of this cell. The standard deviations of the average cycle times are not shown in this figure. The vertical line segments that are connected together with a black line represent the values of this performance measure with the original graph. The vertical line segments that are connected together with a gray line represent the values of this performance measure with the changed graph. Based upon confidence intervals, this changed graph clearly improves the average cycle time of all physical configurations that have multiple ASAs.

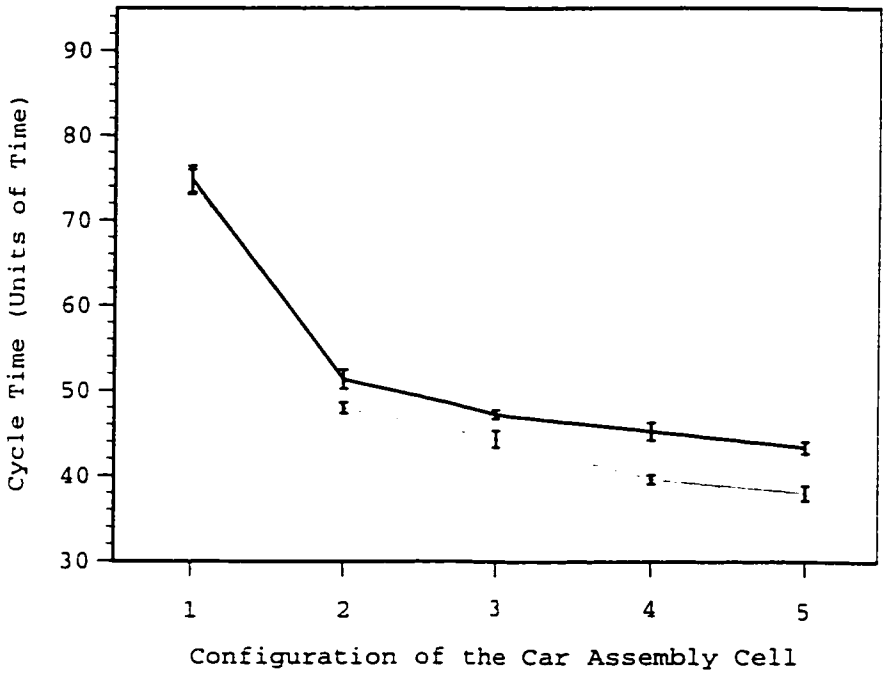


Figure 6.11: Confidence intervals (95%) of the average cycle times compared for different task-level graphs when using exponentially distributed times.

Figure 6.12 plots the percentage utilization of *ASA1* and *PPA1* of the car assembly cell for both task precedence graphs and using exponential distribution times. Only the percent utilization

of these agents was plotted because the utilization of the remaining ASAs and PPAs have comparable values and also to keep the diagram clear. The symbols in this figure are located at the point estimates of these percent utilizations. The legend of this plot specifies which symbols correspond to which physical agent. Graph 1 refers to the original task precedence graph while graph 2 refers to the changed task precedence graph. Symbols of the same physical agent for the original graph are connected together with black line segments. Symbols of the same physical agent for the changed graph are connected together with gray line segments. From this plot, it is obvious that the point estimates of the percent utilization of these physical agents increase for each configuration with multiple ASAs; however, whether this increase is statistically significant needs to be ascertained by comparing the confidence intervals. As might be expected, the reduced cycle time of the different configurations is reflected in a more efficient utilization of the physical agents.

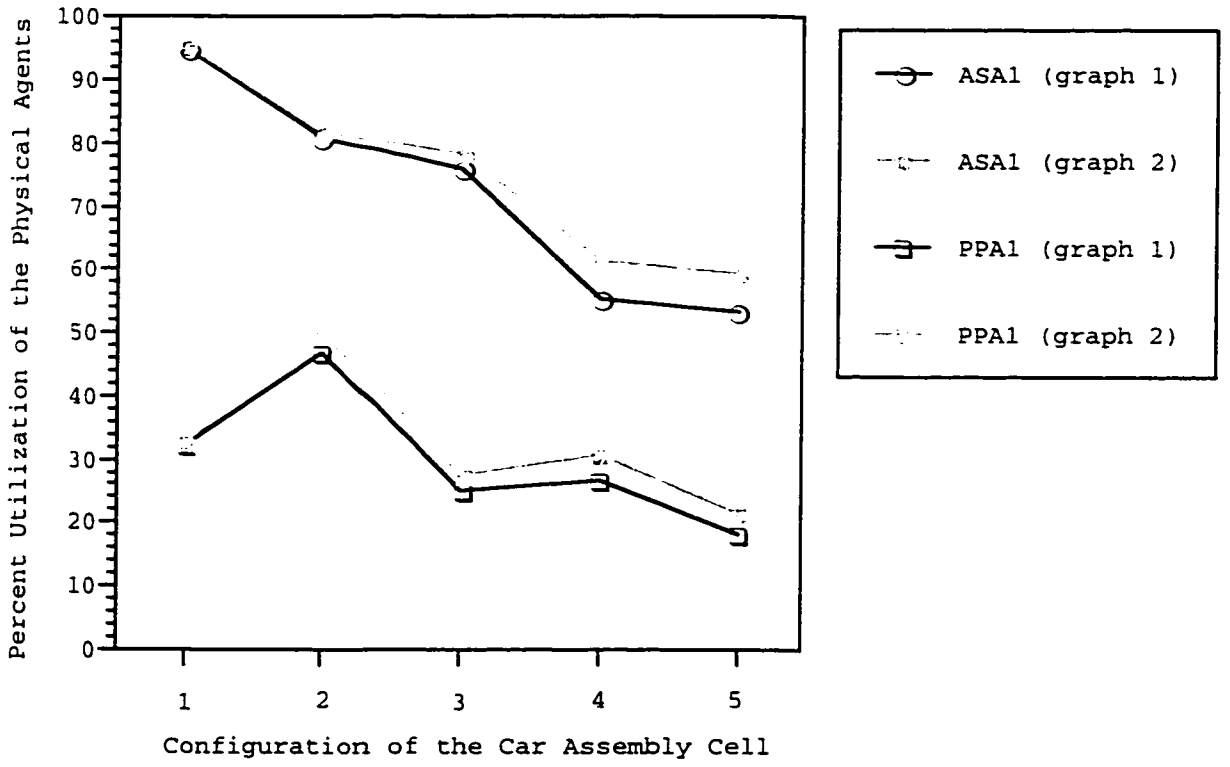


Figure 6.12: Point estimates of agent utilization compared for different task-level graphs when using exponentially distributed times.

Figure 6.13 and figure 6.14 use the same conventions as figure 6.11 and figure 6.12 respectively. Figure 6.13 displays the cycle time of our assembly cell versus its five physical configurations for both task precedence graphs and using normal distribution times. One can again observe that based upon confidence intervals, changing the task precedence graph clearly improves the average cycle time of all physical configurations that have multiple ASAs. Figure 6.14 plots the percentage utilization of *ASA1* and *PPA1* of the car assembly cell for both task precedence graphs and using normal distribution times. The increases in the point estimates of the percent utilization of these physical agents is less pronounced than for the case with the exponential distribution times.

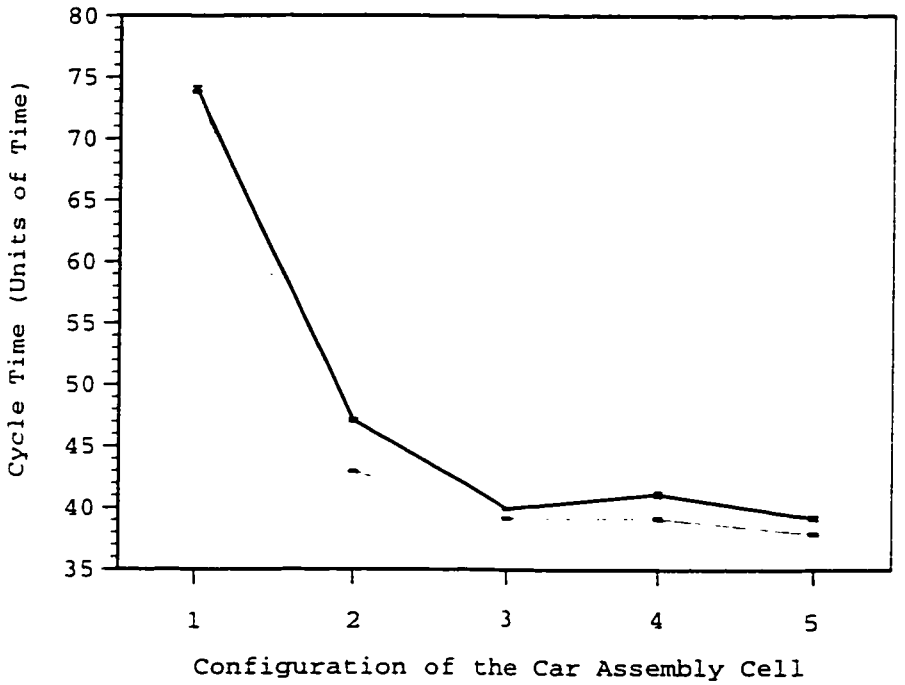


Figure 6.13: Confidence intervals (95%) of the average cycle times compared for different task-level graphs when using normally distributed times.

6.5.3 Closing Discussion

This section presented examples of comparing FABRIC configurations in a statistical sense

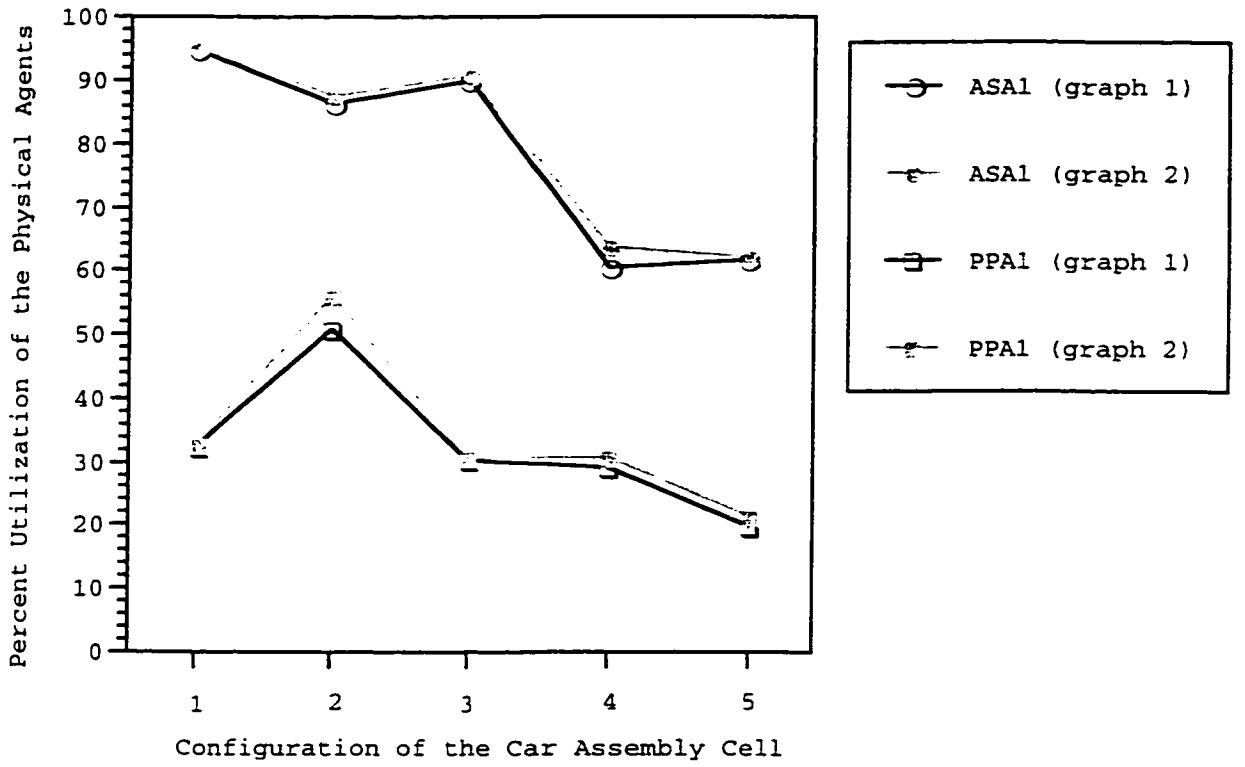


Figure 6.14: Point estimates of agent utilization compared for different task-level graphs when using normally distributed times.

using the results of the DES. This statistical comparison is one one way to evaluate alternative FABRIC systems. The question of whether these statistical differences of various performance measures also have a practical significance depends upon many factors in a particular robotic assembly situation [Rampersad, 1995b, 1996]. For example, is the cost of installing another ASA or PPA worth the reduction in cycle time. Our simulations provide the decision maker only with information regarding how a system change affects various performance measures statistically.

Some basic comments regarding the upper and lower boundaries of the average cycle time are made using the simulation results for the original task precedence graph and using normal distribution times. The results for this case are summarized in table 6.7. These comments are based upon simple calculations that use the means of the times to model the mechanical operations. Based upon the means in table 6.6 and assuming that the electronic delays are negligible, the

average cycle time of configuration 1 should theoretically be 65 units of time. The point estimate of the average cycle time for this configuration from table 6.7 was determined to be 74.09 units of time. What this implies is that in the worst case of contract-net negotiation, the electronic delays account for approximately 12.3% of the total time of an assembly cycle in our hypothetical assembly cell. In the case of configuration 5, the average cycle time should theoretically be 39 units of time. The point estimate of the average cycle time for this configuration was determined to be 39.21 units of time. Clearly, the electronic delays are less significant in this configuration. Moreover, if a physical configuration had four ASA and four PPAs that allowed for all four wheels to be delivered in parallel and assembled in parallel then the average cycle time would theoretically be reduced to 31 units of time.

Although the contract net is used to simulate all five configurations, the choosing of a contractor agent only applied in configuration 4 and 5. Even though this is the case, the simulation results of configuration 1, 2, and 3 provided a baseline to compare the results of the last two configurations. Moreover, the proper operation of the car assembly cell in the first three configurations demonstrates a point about redundancy. That is, if an ASA or PPA had an equipment failure without disrupting the operation of an assembly cell and the remaining ASAs and PPAs could perform all of the necessary assembly tasks, then the operation of a FABRIC system could continue without interruption.

Chapter 7

Conclusions and Future Research Directions

This dissertation proposed and investigated a solution to the research problem of managing the on-line, high-level operation of a task-level MRAC. In the process, this dissertation introduced a number of concepts. These concepts, which were implemented using object-oriented software technology, were partly validated through physical experiments and they were also evaluated through simulation experiments. The first section of this chapter examines these concepts and experiments in relation to the contributions of this dissertation in order to provide some conclusions. The second section discusses potential directions for future research.

7.1 Conclusions

The major concept that was introduced was a generic, agent-based framework as a solution to this research problem. In order to define this framework, the concepts of a task scheduling agent, an assembling agent, a part presentation agent, and a shared space agent were introduced and detailed. In addition, the contents of the messages that were exchanged between these agents in order to coordinate their actions were defined. Having introduced these concepts and definitions, the next stage was to implement and test them. As a result of this research process, several contributions were made.

Two contributions are associated with this framework. The first contribution was the framework solution itself. A basic question regarding this solution is how does it measure up against an ideal solution that has as many of the properties of being modular, extensible, reliable,

efficient, robust, maintainable, and adaptable as possible. In our opinion, this framework does reasonably well with respect to the properties of modularity, extensibility, robustness, and maintainability; however, there is room for improvement regarding the properties of efficiency and adaptability. The second contribution, which is related to the definition of the message contents, was the unique application of the contract net within this framework. Specifically, although many researchers have investigated applications of the contract net to the field of manufacturing, our application is unique by addressing the dynamic task-level scheduling of a MRAC.

Two contributions are associated with the concept of an assembling agent. This agent plays a crucial role in the overall operation of a task-level MRAC. In this case, the first contribution was the physical implementation of a proof of concept, assembling agent with a sensor-based, Puma 500, robotic system. The second contribution concerns the internal operation of this physical assembling agent. In particular, a novel approach to the problem of task planning was proposed and implemented as part of the internal operation of this physical assembling agent.

Two contributions are associated with the development of software in order to evaluate this framework through simulation experiments. The first contribution was the development of one of the few frame-based knowledge bases in the object-oriented programming language C++. The implementation of this knowledge base was required in order to accurately simulate the internal behaviour of the FABRIC agents. The second contribution was the development of a prototyping tool to simulate the discrete event behaviour of MRACs managed with our framework.

7.2 Future Research Directions

Given the interdisciplinary nature of the research studied in this dissertation, there are several potential directions for future research. This section discusses three categories of research directions.

The first category concerns research developments regarding the physical system. Research concerning the basic robotic skills needed by an assembling agent in order to perform a

set of assembly operations along with their implementation and testing need to be explored. The physical implementation of a complete research FABRIC system would probably encounter several subtle issues that would have to be addressed.

The second category of research directions concerns the simulation of FABRIC systems. In particular, the existing simulator could be improved upon or extended in several ways. First, one of the more time consuming tasks of simulating a particular FABRIC configuration is setting up the knowledge base of each agent. This problem is well recognized as the knowledge bottleneck by the AI community. This problem could be addressed by creating an interactive tool to setup the knowledge of the agents. Second, the agents of FABRIC could be improved upon by integrating some learning techniques into their simulation behaviour. Third, various options, such as different performance measures, different probability distributions, alternative techniques for task negotiation, agents with modifiable communicating state machines, etc., could be studied through simulation.

The third category of research directions concerns the integration of aspects of the first two categories. In this case, the research and development of a virtual prototyping environment for FABRIC systems could be investigated. This investigation would require that the discrete event simulation would have to extended by modelling and simulating the actions of the physical agents using a commercial 3D solid modelling and animation package.

Appendix A

Program Listing of the Class Agent

This appendix and the following two appendices contain the listings of the C++ program files that implement certain agents of FABRIC in software. A few comments regarding these listings are necessary. First, the software was written using Symantec C++* development environment version 7.0.5 for the Macintosh™. Macros that begin with *TCL_* are specific to this environment and belong to the Think Class Library* are found throughout these listings. Second, in these listings the header file of a class (name_of_file.h) is reproduced before the source file of a class (name_of_file.cp). Appendix A contains the program files *Agent.h* and *Agent.cp*. Appendix B contains the program files *Task_Scheduling_Agent.h* and *Task_Scheduling_Agent.cp*. Appendix C contains the program files *Assembling_Agent.h* and *Assembling_Agent.cp*.

---

* Trademark of Symantec Corporation.

```
/*=========================================================================*/
/*      Title:      Agent.h                                                */
/*      Author: Jagdeep S. Basran                                          */
/*      Date:       June 6, 1996                                           */
/*      Comments:   header file for the class Agent                        */
/*=========================================================================*/

#ifndef AGENT_H
#define AGENT_H

//  include files needed

#include "Knowledge_Base.h"
#include "Basic_List.h"
#include "Basic_Queue.h"
#include "Random.h"
#include "Standard_File.h"

// Advance declaration of classes

class Acquaintance_Agent_Frame;
class Bid_Frame;
class Grant_Frame;
class Message_Frame;
class Multi_Slot;
class Report_Frame;
class Simulator;
class State;
class Task_Announcement_Frame;

//  Class definition

class Agent
{

TCL_DECLARE_CLASS

    friend class Acquaintance_Agent_Frame;
    friend class Simulator;

private:

    //  private variables

    char            *_name_of_agent, *_name_of_direct_parent;
    Standard_File   _the_trace_file;
    double          _the_last_switch_time, _total_busy_time;

    //  private methods

    void    Initialize_Name_of_Agent(const char *name_for_agent);
    void    Initialize_Name_of_Direct_Parent(
                                        const char *name_of_direct_parent);
    void    Setup_Basic_Knowledge(void);
    void    Set_Default_Agent_Values(void);
    void    Free_Agent_Memory(void);

    void    Setup_Trace_Information(const char *name_of_agent);
    void    Add_Agent_Utilization_to_Trace_File(void);

protected:

    // class variables
```

```
static RandomStream       *_the_message_delay;
static RandomStream       *_the_mean_is_1;
static RandomStream       *_the_mean_is_4;
static RandomStream       *_the_mean_is_6;

//  protected constants, variables, and methods

#define MODEL_OF_SELF         "model of self"
#define MODEL_OF_ENVIRONMENT  "model of environment"
#define DELIMITER_STRING      ":"
#define DELIMITER_CHAR        ':'

Knowledge_Base                _the_knowledge_base;
Basic_Queue<Message_Frame *>  _the_message_queue;

//  miscellaneous methods

char         *get_agent_identity(void) const;
const State  &get_current_state(void);
void         set_current_state(const State &new_state);

//  contract net methods

Boolean      Eligibility_Requirements_Met(
                                    Task_Announcement_Frame *the_task);
Boolean      Bid_Submission_Deadline_Met(
                                    Task_Announcement_Frame *the_task);
void         Get_First_Potential_Task(Task_Announcement_Frame *&the_task);

void         Setup_Bid_Information(Task_Announcement_Frame *the_task,
                                          Multi_Slot &bid_information);
void         Get_Bids_for_Task(CString &task_identity,
                                    Basic_List<Bid_Frame *> &the_bids);
void         Select_Best_Bid_Randomly(Basic_List<Bid_Frame *> &the_bids,
                                                 size_t &best_index);
void         Select_Best_Bid_based_on_Minimum_Time(
                 Basic_List<Bid_Frame *> &the_bids, size_t &best_index);

Boolean      Received_the_Grant(Grant_Frame *the_grant);
Boolean      Get_the_Grant_Message(Grant_Frame *&the_grant);
Boolean      Get_the_Report_Message(Report_Frame *&the_report);
Report_Frame     *Create_Report_Frame_Response(Message_Frame *the_message,
                                          CString &recipient_identity);

//  methods to send messages

void     Send_Request_Message_to_SSA(const CString &task_string,
                                     const CString &region_of_space);
void     Send_Release_Message_to_SSA(const CString &task_string,
                                     const CString &region_of_space);

void     Send_Message(const char *recipient_agent,
                            const Message_Frame *message_to_send);
void     Send_Limited_Broadcast(const char *type_of_acquaintance_agent,
                                     Message_Frame *message_to_send);

//  simulation methods

void         Register_Sending_of_Message(const char * recipient_agent,
                                     Message_Frame *message_to_send);
void         Register_Trace_Message(const char *trace_message);
void         Switch_to_Busy(void);
void         Switch_to_Idle(void);
```

```
    //  Constructors and destructor

    Agent(void);
    Agent(const char *name_of_agent, const char *name_of_direct_parent);

    virtual ~Agent(void);

    //  virtual methods

    virtual void Add_Acquaintance(Acquaintance_Agent_Frame *new_agent_frame);
    virtual void Receive_Message(const Message_Frame *message_to_receive);
    virtual void Execute(int type_of_event);

};

#endif
```

```
/*=============================================================================*/
/*      Title:      Agent.cp                                                   */
/*      Parent:  None                                                          */
/*      Author:  Jagdeep S. Basran                                             */
/*      Date:       June 6, 1996                                               */
/*      Comments:   this class defines a base class for a FABRIC agent.        */
/*                  these agents have a communicating state-machine, a         */
/*                  knowledge base, and a message queue and they cooperate     */
/*                  using the contract net.                                    */
/*=============================================================================*/

//  include files needed for this class

#include "Agent.h"
#include "Multi_Slot.h"
#include "Simulator.h"

#include "Acquaintance_ASA_Frame.h"
#include "Acquaintance_PPA_Frame.h"
#include "Acquaintance_SSA_Frame.h"
#include "Acquaintance_TSA_Frame.h"
#include "Bid_Frame.h"
#include "Grant_Frame.h"
#include "Message_Frame.h"
#include "Report_Frame.h"
#include "Request_Frame.h"
#include "Model_of_Agent_Frame.h"
#include "Model_of_Environment_Frame.h"
#include "Task_Announcement_Frame.h"

#include <Float.h>
#include <time.h>

//  the following two defines allow the user to turn the detailed trace
//  files on or off and to determine whether the normal distribution is
//  used instead of the default exponential distribution.

#define TRACE_INFORMATION
#define NORMAL_DISTRIBUTION

// the following lines allocate and intialize the class variables of the
//  class Agent and its subclasses

RandomStream     *Agent::_the_message_delay = new NormalStream(0.10, 0.02);

#ifdef NORMAL_DISTRIBUTION
    RandomStream     *Agent::_the_mean_is_1 = new NormalStream(1.0, 0.01);
    RandomStream     *Agent::_the_mean_is_4 = new NormalStream(4.0, 0.04);
    RandomStream     *Agent::_the_mean_is_6 = new NormalStream(6.0, 0.06);
#else
    RandomStream     *Agent::_the_mean_is_1 = new ExponentialStream(1.0);
    RandomStream     *Agent::_the_mean_is_4 = new ExponentialStream(4.0);
    RandomStream     *Agent::_the_mean_is_6 = new ExponentialStream(6.0);

#endif

TCL_DEFINE_CLASS_M0(Agent)


/*****************************************************************************
 *  Initialize_Name_of_Agent (private)
 *
 *  this method initializes the name of this agent. it assumes that the
 *  previous name is deleted.
```

```
 *
 ****************************************************************************/

void Agent::Initialize_Name_of_Agent(const char *name_for_agent)
{
    _name_of_agent = new char [strlen(name_for_agent) + 1];
    strcpy(_name_of_agent, name_for_agent);
}

/****************************************************************************
 *  Initialize_Name_of_Direct_Parent (private)
 *
 *  this method initializes the name of the direct parent of this agent.
 *  it assumes that the previous name is deleted.
 *
 ****************************************************************************/

void Agent::Initialize_Name_of_Direct_Parent(
    const char *name_of_direct_parent)
{
    _name_of_direct_parent = new char [strlen(name_of_direct_parent) + 1];
    strcpy(_name_of_direct_parent, name_of_direct_parent);
}

/****************************************************************************
 *  Setup_Basic_Knowledge (private)
 *
 *  this method adds the frames that represent the model of self and the
 *  model of the environment to the knowledge base of an agent.  it is up
 *  to the derived agent to define its own states and specific
 *  capabilities in the model of self frame.  the derived agent also adds
 *  its own information regarding its environment to the second frame.
 *
 ****************************************************************************/

void Agent::Setup_Basic_Knowledge(void)
{
    State                          null_state;
    Model_of_Agent_Frame      model_of_self(MODEL_OF_SELF, null_state);
    Model_of_Environment_Frame    model_of_environment(MODEL_OF_ENVIRONMENT);

    _the_knowledge_base.Add_Frame(&model_of_self);
    _the_knowledge_base.Add_Frame(&model_of_environment);
}

/****************************************************************************
 *  Set_Default_Agent_Values (private)
 *
 *  this method sets up default values for the private variables.
 *
 ****************************************************************************/

void Agent::Set_Default_Agent_Values(void)
{
    _name_of_agent = NULL;
    _name_of_direct_parent = NULL;
}

/****************************************************************************
 *  Free_Agent_Memory (private)
 *
 *  this method releases all of the dynamic memory used by an agent.
 *
 ****************************************************************************/
```

```
void Agent::Free_Agent_Memory(void)
{
    if(_name_of_agent)
        delete [] _name_of_agent;
    if(_name_of_direct_parent)
        delete [] _name_of_direct_parent;
}

/*****************************************************************************
 *  Setup_Trace_Information (private)
 *
 *  given the name of an agent, this method sets up a trace file in order
 *  to trace the execution of an agent.
 *
 *****************************************************************************/

void Agent::Setup_Trace_Information(const char *name_of_agent)
{
    CString      trace_file_name;

    trace_file_name = DELIMITER_STRING;
    trace_file_name += "Agent Traces";
    trace_file_name += DELIMITER_STRING;
    trace_file_name += name_of_agent;
    trace_file_name += ".log";

    _the_trace_file.open_file(trace_file_name, "w+");

    _the_last_switch_time = _total_busy_time = 0.0;
}

/*****************************************************************************
 *  Add_Agent_Utilization_to_Trace_File (private)
 *
 *  this method calculates the utilization of an agent and adds this
 *  information to its trace file.
 *
 *****************************************************************************/

void Agent::Add_Agent_Utilization_to_Trace_File(void)
{
    double   total_simulation_time = the_simulator->
                                          Get_Current_Simulation_Time();
    double   percent_utilization;
    char     buffer[256];

    percent_utilization = (_total_busy_time / total_simulation_time) * 100;

    sprintf(buffer, "\n\nagent utilization:\t%.2lf\n", percent_utilization);

    _the_trace_file.write_file(buffer);
}

/*****************************************************************************
 *  get_agent_identity (protected)
 *
 *  this method returns a copy of the complete identity of the agent.
 *  that is, it returns the concatenation of the name of the direct
 *  parent, a colon, and the particular name of an agent.  it is up to the
 *  user to free the memory associated with this string.
 *
 *****************************************************************************/
```

```
char *Agent::get_agent_identity(void) const
{
    if(!_name_of_direct_parent)
        return(NULL);

    size_t   string_length;
    char     *agent_identity;

    string_length = strlen(_name_of_direct_parent) +
                                                    strlen(_name_of_agent) + 2;
    agent_identity = new char [string_length];

    strcpy(agent_identity, _name_of_direct_parent);
    strcat(agent_identity, DELIMITER_STRING);
    strcat(agent_identity, _name_of_agent);

    return(agent_identity);
}

/*************************************************************************
 *  get_current_state (protected)
 *
 *  this method simply retrieves the current state of an agent.
 *
 *************************************************************************/

const State &Agent::get_current_state(void)
{
    Base_Frame               *frame_base_class;
    Model_of_Agent_Frame *self_model_frame;

    frame_base_class = _the_knowledge_base.Get_Frame(MODEL_OF_SELF);
    self_model_frame = TCL_DYNAMIC_CAST(Model_of_Agent_Frame,
                                                        frame_base_class);

    return(self_model_frame->Get_Agent_State());
}

/*************************************************************************
 *  set_current_state (protected)
 *
 *  this method sets the current state of an agent.
 *
 *************************************************************************/

void Agent::set_current_state(const State &new_state)
{
    Base_Frame               *frame_base_class;
    Model_of_Agent_Frame *self_model_frame;

    frame_base_class = _the_knowledge_base.Get_Frame(MODEL_OF_SELF);
    self_model_frame = TCL_DYNAMIC_CAST(Model_of_Agent_Frame,
                                                        frame_base_class);

    self_model_frame->Set_Agent_State(new_state);
}

/*************************************************************************
 *  Eligibility_Requirements_Met (protected)
 *
 *  this method returns TRUE if an agent meets the eligibility
 *  requirements of a particular task (part assembling, part fetching)
 *
 *************************************************************************/
```

```
Boolean Agent::Eligibility_Requirements_Met(
    Task_Announcement_Frame *the_task)
{
    Base_Frame               *frame_base_class;
    Model_of_Agent_Frame *self_model_frame;
    Generic_Slot             *slot_base_class;
    Multi_Slot               *eligibility_slots;

    frame_base_class = _the_knowledge_base.Get_Frame(MODEL_OF_SELF);
    self_model_frame = TCL_DYNAMIC_CAST(Model_of_Agent_Frame,
                                                    frame_base_class);
    slot_base_class = the_task->Get_Slot(ELIGIBILITY_SPECIFICATION);
    eligibility_slots = TCL_DYNAMIC_CAST(Multi_Slot, slot_base_class);

    unsigned long    slot_index;
    Generic_Slot     *current_slot;

    for(slot_index=0; slot_index<eligibility_slots->
                                Get_Current_Cardinality(); slot_index++)
    {
        current_slot = eligibility_slots->Get_Slot(slot_index);
        if(!self_model_frame->Eligibilty_Satisfied(current_slot))
            return(FALSE);
    };

    return(TRUE);
}

/*****************************************************************************
 *  Bid_Submission_Deadline_Met (protected)
 *
 *  this method returns TRUE if the bid submission deadline is satisfied
 *  by an agent; otherwise, it returns FALSE.
 *
 *****************************************************************************/

Boolean Agent::Bid_Submission_Deadline_Met(
    Task_Announcement_Frame *the_task)
{
    Generic_Slot     *slot_base_class;
    Slot<double>     *expiration_slot;
    double           current_time, expiration_time;

    slot_base_class = the_task->Get_Slot(EXPIRATION_TIME);
    expiration_slot = TCL_DYNAMIC_CAST(Slot<double>, slot_base_class);
    expiration_time = expiration_slot->Get_Slot_Value();

    current_time = the_simulator->Get_Current_Simulation_Time();
    if(current_time < expiration_time)
        return(TRUE);
    else
        return(FALSE);
}

/*****************************************************************************
 *  Get_First_Potential_Task (protected)
 *
 *  this method returns the first task in the message queue that an
 *  agent can do and that is still available.
 *
 *****************************************************************************/

void Agent::Get_First_Potential_Task(Task_Announcement_Frame *&first_task)
```

```
{
    Message_Frame                       *the_message;
    Task_Announcement_Frame             *the_task;
    Basic_Queue<Message_Frame *>  other_messages;

    first_task = NULL;
    the_message = _the_message_queue.Pop_from_Front();
    while(the_message)
    {
        the_task = TCL_DYNAMIC_CAST(Task_Announcement_Frame, the_message);
        if(the_task)
        {
            if((!Eligibility_Requirements_Met(the_task)) ||
                                (!Bid_Submission_Deadline_Met(the_task)))
                delete the_task;
            else
            {
                first_task = the_task;
                break;
            }
        }
        else
            other_messages.Push_to_Front(the_message);

        the_message = _the_message_queue.Pop_from_Front();
    };

    the_message = other_messages.Pop_from_Front();
    while(the_message)
    {
        _the_message_queue.Push_to_Front(the_message);
        the_message = other_messages.Pop_from_Front();
    };
}

/*****************************************************************************
 *  Setup_Bid_Information (protected)
 *
 *  given the bid specification contained in the task announcement frame,
 *  this method sets up the bid information in the specified multi-slot.
 *
 *****************************************************************************/

void Agent::Setup_Bid_Information(
    Task_Announcement_Frame *the_task, Multi_Slot &bid_information)
{
    Generic_Slot              *slot_base_class;
    Slot<CString>             *task_slot;
    CString                   task_abstraction;
    Multi_Slot                *bid_slots;
    Base_Frame                *frame_base_class;
    Model_of_Agent_Frame *self_model_frame;

    slot_base_class = the_task->Get_Slot(TASK_ABSTRACTION);
    task_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    task_abstraction = task_slot->Get_Slot_Value();
    slot_base_class = the_task->Get_Slot(BID_SPECIFICATION);
    bid_slots = TCL_DYNAMIC_CAST(Multi_Slot, slot_base_class);

    frame_base_class = _the_knowledge_base.Get_Frame(MODEL_OF_SELF);
    self_model_frame = TCL_DYNAMIC_CAST(Model_of_Agent_Frame,
                                                        frame_base_class);

    unsigned long    slot_index;
```

```
    Generic_Slot      *information_slot;
    Slot<CString>     *current_slot;
    CString           information_string;

    for(slot_index=0; slot_index<bid_slots->Get_Current_Cardinality();
                                                              slot_index++)
    {
        slot_base_class = bid_slots->Get_Slot(slot_index);
        current_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
        if(current_slot)
        {
            information_string = current_slot->Get_Slot_Value();
            information_slot = self_model_frame->Get_Information(
                                                    information_string);

            if(information_slot)
            {
                bid_information.Append_Slot(information_slot);
                delete information_slot;
            };
        };
    };
}

/*****************************************************************************
 *  Get_Bids_for_Task (protected)
 *
 *  this method gets all bids that were sent in time from contractor
 *  agents to a manager agent.  only the bids that correspond to the task
 *  whose submission deadline has passed are returned in the list provided
 *  by the user.
 *
 *****************************************************************************/

void Agent::Get_Bids_for_Task(
    CString &task_identity, Basic_List<Bid_Frame *> &the_bids)
{
    Message_Frame                       *the_message;
    Bid_Frame                           *the_bid;
    Basic_Queue<Message_Frame *>  other_messages;

    the_message = _the_message_queue.Pop_from_Front();
    while(the_message)
    {
        the_bid = TCL_DYNAMIC_CAST(Bid_Frame, the_message);
        if(the_bid)
            the_bids.Add_to_Back(the_bid);
        else
            other_messages.Push_to_Back(the_message);

        the_message = _the_message_queue.Pop_from_Front();
    };

    size_t            total_bids = the_bids.number_of_elements();
    CString           bid_string;
    long              index;

    for(index=total_bids-1; index>=0; index--)
    {
        bid_string = the_bids[index]->get_name_of_instance();
        if(strcmp(bid_string, task_identity))
        {
            _the_message_queue.Push_to_Front(the_bids[index]);
            the_bids.Remove_At(index);
```

```
        };
    };

    the_message = other_messages.Pop_from_Front();
    while(the_message)
    {
        _the_message_queue.Push_to_Back(the_message);
        the_message = other_messages.Pop_from_Front();
    };
}

/*****************************************************************************
 *  Select_Best_Bid_Randomly (protected)
 *
 *  given all of the bids received by a manager agent, this method
 *  selects a contractor agent randomly.
 *
 *****************************************************************************/

void Agent::Select_Best_Bid_Randomly(
    Basic_List<Bid_Frame *> &the_bids, size_t &best_index)
{
    size_t  total_bids = the_bids.number_of_elements();

    if(total_bids<=1)
    {
        best_index=0;
        return;
    };

    double  random_number;

    random_number = rand();
    best_index = random_number * total_bids / (RAND_MAX+1);
}

/*****************************************************************************
 *  Select_Best_Bid_based_on_Minimum_Time (protected)
 *
 *  given all of the bids received by a manager agent, this method
 *  selects a contractor agent by picking the agent that can perform
 *  the required task in the least amount of time.
 *
 *****************************************************************************/

void Agent::Select_Best_Bid_based_on_Minimum_Time(
    Basic_List<Bid_Frame *> &the_bids, size_t &best_index)
{
    size_t          total_bids = the_bids.number_of_elements();
    size_t          index;
    double          minimum_execution_time=DBL_MAX;
    Generic_Slot    *slot_base_class;
    Multi_Slot      *bid_information;
    Slot<double>    *execution_time_slot;
    double          execution_time;

    for(index=0; index<total_bids; index++)
    {
        slot_base_class = the_bids[index]->Get_Slot(BID_INFORMATION);
        bid_information = TCL_DYNAMIC_CAST(Multi_Slot, slot_base_class);
        if(bid_information->Get_Current_Cardinality())
        {
            slot_base_class = bid_information->Get_Slot(0);
            execution_time_slot = TCL_DYNAMIC_CAST(Slot<double>,
```

```
                                                                    slot_base_class);
            execution_time = execution_time_slot->Get_Slot_Value();
            if(execution_time<minimum_execution_time)
            {
                best_index=index;
                minimum_execution_time=execution_time;
            };
        }
        else
        {
            best_index=0;
            break;
        }
    };
}

/*****************************************************************************
 *  Received_the_Grant (protected)
 *
 *  this method returns TRUE if a grant message is for an agent.
 *
 *****************************************************************************/

Boolean Agent::Received_the_Grant(Grant_Frame *the_grant)
{
    char            *agent_identity = get_agent_identity();
    Generic_Slot    *slot_base_class;
    Slot<CString>   *recipient_slot;
    CString         recipient_string;
    const char      *recipient_identity;
    Boolean         received_grant;

    slot_base_class = the_grant->Get_Slot(RECIPIENT);
    recipient_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    recipient_string = recipient_slot->Get_Slot_Value();
    recipient_identity = recipient_string;
    if (strcmp(agent_identity, recipient_identity))
        received_grant = FALSE;
    else
        received_grant = TRUE;

    delete [] agent_identity;
    return(received_grant);
}

/*****************************************************************************
 *  Get_the_Grant_Message (protected)
 *
 *  this method returns TRUE if the next message in the queue is a grant
 *  message for this agent.  this message is returned in the given
 *  variable.
 *
 *****************************************************************************/

Boolean Agent::Get_the_Grant_Message(Grant_Frame *&the_grant)
{
    Message_Frame    *the_message;

    the_message = _the_message_queue.Pop_from_Front();
    the_grant = TCL_DYNAMIC_CAST(Grant_Frame, the_message);
    if(!the_grant)
        return(FALSE);

    char            *agent_identity = get_agent_identity();
```

```
    Generic_Slot     *slot_base_class;
    Slot<CString>    *recipient_slot;
    CString          recipient_string;
    const char       *recipient_identity;

    slot_base_class = the_grant->Get_Slot(RECIPIENT);
    recipient_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    recipient_string = recipient_slot->Get_Slot_Value();
    recipient_identity = recipient_string;
    if (strcmp(agent_identity, recipient_identity))
    {
        delete [] agent_identity;
        delete the_grant;
        the_grant = NULL;
        return(FALSE);
    }
    else
    {
        delete [] agent_identity;
        return(TRUE);
    }
}

/*****************************************************************************
 *  Get_the_Report_Message (protected)
 *
 *  this method returns TRUE if the next message in the queue is a report
 *  message.  this message is returned in the given variable.
 *
 *****************************************************************************/

Boolean Agent::Get_the_Report_Message(Report_Frame *&the_report)
{
    Message_Frame    *the_message;

    the_message = _the_message_queue.Pop_from_Front();
    the_report = TCL_DYNAMIC_CAST(Report_Frame, the_message);
    if(!the_report)
        return(FALSE);
    else
        return(TRUE);
}

/*****************************************************************************
 *  Create_Report_Frame_Response (protected)
 *
 *  this method creates a report message response to the specified
 *  message and returns the identity of the message recipient.  it is up
 *  to the user to add the appropriate information slots to this report
 *  message and delete it when no longer needed.
 *
 *****************************************************************************/

Report_Frame *Agent::Create_Report_Frame_Response(
    Message_Frame *the_message, CString &recipient_identity)
{
    CString          report_identity;
    char             *sender_identity;
    Generic_Slot     *slot_base_class;
    Slot<CString>    *message_sender_slot;
    Report_Frame     *the_report;

    report_identity = the_message->get_name_of_instance();
    sender_identity = get_agent_identity();
```

```
    slot_base_class = the_message->Get_Slot(SENDER);
    message_sender_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    recipient_identity = message_sender_slot->Get_Slot_Value();

    the_report = new Report_Frame(report_identity, sender_identity,
                                                    recipient_identity);
    delete [] sender_identity;
    return(the_report);
}

/*****************************************************************************
 *  Send_Request_Message_to_SSA (protected)
 *
 *  given a task identifying string and the region of space, this method
 *  sends a requests for this region to the shared space agent.
 *
 *****************************************************************************/

void Agent::Send_Request_Message_to_SSA(
    const CString &task_string, const CString &region_of_space)
{
    CString                 recipient_string;
    const char              *recipient_identity;
    char                    *sender_identity;
    Base_Frame              *frame_base_class;
    Acquaintance_SSA_Frame  *shared_space_agent;

    frame_base_class = _the_knowledge_base.Get_Frame(SPACE_MANAGER_STRING);
    shared_space_agent = TCL_DYNAMIC_CAST(Acquaintance_SSA_Frame,
                                                    frame_base_class);
    recipient_string = shared_space_agent->get_agent_reference();
    recipient_identity = recipient_string;

    sender_identity = get_agent_identity();

    Request_Frame    *the_request;

    the_request = new Request_Frame(task_string, sender_identity,
                    recipient_identity, MUTUAL_EXCLUSION, region_of_space);

    Send_Message(recipient_identity, the_request);

    delete [] sender_identity;
    delete the_request;
}

/*****************************************************************************
 *  Send_Release_Message_to_SSA (protected)
 *
 *  given a task identifying string and the region of space, this method
 *  sends a release message to the shared space agent informing it that
 *  a previously occupied region of space is now available.
 *
 *****************************************************************************/

void Agent::Send_Release_Message_to_SSA(
    const CString &task_string, const CString &region_of_space)
{
    CString                 recipient_string;
    const char              *recipient_identity;
    char                    *sender_identity;
    Base_Frame              *frame_base_class;
    Acquaintance_SSA_Frame  *shared_space_agent;
```

```
    frame_base_class = _the_knowledge_base.Get_Frame(SPACE_MANAGER_STRING);
    shared_space_agent = TCL_DYNAMIC_CAST(Acquaintance_SSA_Frame,
                                                    frame_base_class);
    recipient_string = shared_space_agent->get_agent_reference();
    recipient_identity = recipient_string;

    sender_identity = get_agent_identity();

    Report_Frame     *the_report;
    Slot<CString>    region_released(REGION_RELEASED, region_of_space);

    the_report = new Report_Frame(task_string, sender_identity,
                                                    recipient_identity);
    the_report->Add_Slot(&region_released,
                                TCL_CLASSNAME_FROM_POINTER(the_report));

    Send_Message(recipient_identity, the_report);

    delete [] sender_identity;
    delete the_report;
}

/*****************************************************************************
 *  Send_Message (protected)
 *
 *  this method sends the specified message to the agent specified by the
 *  character string.
 *
 *****************************************************************************/

void Agent::Send_Message(
    const char *recipient_agent, const Message_Frame *message_to_send)
{
    Message_Frame    *the_message = (Message_Frame *) message_to_send;

    the_message->Set_Message_Time(the_simulator->
                                        Get_Current_Simulation_Time());
    the_simulator->Send_Message(recipient_agent, message_to_send);

    #ifdef TRACE_INFORMATION

        Register_Sending_of_Message(recipient_agent, the_message);

    #endif
}

/*****************************************************************************
 *  Send_Limited_BroadCast (protected)
 *
 *  given the type of acquaintance agent, this method sends the
 *  specified message to all acquaintance agent of this type that an
 *  an agent has stored in its knowledge base.
 *
 *****************************************************************************/

void Agent::Send_Limited_Broadcast(
    const char *type_of_acquaintance_agent, Message_Frame *message_to_send)
{
    Base_Frame                  **the_agent_frames;
    long                        total_agents, agent_index;
    Acquaintance_Agent_Frame *the_agent;
    CString                     recipient_string;
    const char                  *recipient_identity;
```

```
    _the_knowledge_base.Get_All_Frames(type_of_acquaintance_agent,
                                       the_agent_frames, total_agents);

    for(agent_index=0; agent_index<total_agents; agent_index++)
    {
        the_agent = TCL_DYNAMIC_CAST(Acquaintance_Agent_Frame,
                                     the_agent_frames[agent_index]);
        recipient_string = the_agent->get_agent_reference();
        recipient_identity = recipient_string;
        message_to_send->Set_Message_Recipient(recipient_identity);
        Send_Message(recipient_identity, message_to_send);
    };

    delete [] the_agent_frames;
}

/*************************************************************************
 *  Register_Sending_of_Message (protected)
 *
 *  this method registers the sending of a message in the trace file
 *  of an agent.
 *
 *************************************************************************/

void Agent::Register_Sending_of_Message(
    const char * recipient_agent, Message_Frame *message_to_send)
{
    CString      message_type, message_identifier;

    if(TCL_DYNAMIC_CAST(Bid_Frame, message_to_send))
        message_type = "bid message";
    else if(TCL_DYNAMIC_CAST(Grant_Frame, message_to_send))
        message_type = "grant message";
    else if(TCL_DYNAMIC_CAST(Report_Frame, message_to_send))
        message_type = "report message";
    else if(TCL_DYNAMIC_CAST(Request_Frame, message_to_send))
        message_type = "request message";
    else if(TCL_DYNAMIC_CAST(Task_Announcement_Frame, message_to_send))
        message_type = "task announcement message";

    message_identifier = message_to_send->get_name_of_instance();

    Generic_Slot     *slot_base_class;
    Slot<CString>    *sender_slot;
    CString          sender, trace_message;

    slot_base_class = message_to_send->Get_Slot(SENDER);
    sender_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    sender = sender_slot->Get_Slot_Value();

    trace_message = "SENDER:\t\t";        trace_message += sender;
    trace_message += "\nMESSAGE:\t";  trace_message += message_type;
    trace_message += " - '";          trace_message += message_identifier;
    trace_message += "'";                 trace_message += "\nRECIPIENT:\t";
    trace_message += recipient_agent;

    double   message_time = message_to_send->Get_Message_Time();
    char     buffer[256];

    sprintf(buffer, "\n\nTIME:\t%.3lf\n", message_time);
    strcat(buffer, trace_message);
    _the_trace_file.write_file(buffer);
}
```

```
/*****************************************************************************
 *  Register_Trace_Message (protected)
 *
 *  this method registers the trace message with a time stamp of the
 *  current simulation time.
 *
 *****************************************************************************/

void Agent::Register_Trace_Message(const char *trace_message)
{
    #ifdef TRACE_INFORMATION

        double  time_of_message = the_simulator->
                                            Get_Current_Simulation_Time();
        char    buffer[256];

        sprintf(buffer, "\n\nTIME:\t%.3lf\n", time_of_message);
        strcat(buffer, _name_of_agent);
        strcat(buffer, DELIMITER_STRING);
        strcat(buffer, trace_message);

        _the_trace_file.write_file(buffer);

    #endif
}

/*****************************************************************************
 *  Switch_to_Busy (protected)
 *
 *  assuming that an agent was idle, this method notes the time that an
 *  agent switches to being busy.  this method is used to help record
 *  the time that an agent is utilized and therefore allows calculating
 *  the percent utilization of an agent at the end of a simulation run.
 *
 *****************************************************************************/

void Agent::Switch_to_Busy(void)
{
    _the_last_switch_time = the_simulator->Get_Current_Simulation_Time();
}

/*****************************************************************************
 *  Switch_to_Idle (protected)
 *
 *  assuming that an agent was busy, this method determines the time that
 *  it was busy and then adds this time to the total busy time of an agent.
 *  this timekeeping allows calculating the percent utilization of an
 *  agent at the end of a simulation run.
 *
 *****************************************************************************/

void Agent::Switch_to_Idle(void)
{
    double  time_of_switch = the_simulator->Get_Current_Simulation_Time();

    _total_busy_time += (time_of_switch - _the_last_switch_time);
    _the_last_switch_time = time_of_switch;
}

/*****************************************************************************
 *  Agent (default constructor - protected)
 *
 *  the default constructor ensures that the internal variables are
 *  properly initialized.  technically, this constructor should never get
```

```
 *   called.  that is, all derived agents should call the next constructor.
 *
 ****************************************************************************/

Agent::Agent(void)
{
    Set_Default_Agent_Values();
}

/****************************************************************************
 *   Agent (constructor - protected)
 *
 *   the constructor initializes the agent with the given name and the
 *   given name for the direct parent.
 *
 ****************************************************************************/

Agent::Agent(const char *name_of_agent, const char *name_of_direct_parent)
{
    Set_Default_Agent_Values();
    Initialize_Name_of_Agent(name_of_agent);
    Initialize_Name_of_Direct_Parent(name_of_direct_parent);
    Setup_Basic_Knowledge();
    Setup_Trace_Information(name_of_agent);
}

/****************************************************************************
 *   ~Agent (destructor)
 *
 *   the destructor adds an agent's utilization to its trace file and then
 *   simply frees the dynamic memory being used.
 *
 ****************************************************************************/

Agent::~Agent(void)
{
    Add_Agent_Utilization_to_Trace_File();
    Free_Agent_Memory();
}

/****************************************************************************
 *   Add_Acquaintance (protected - virtual)
 *
 *   this method provides a default method for an agent to add another
 *   agent to its list of acquaintances.
 *
 ****************************************************************************/

void Agent::Add_Acquaintance(Acquaintance_Agent_Frame *new_agent_frame)
{
    Acquaintance_ASA_Frame    *ASA_Frame;

    ASA_Frame = TCL_DYNAMIC_CAST(Acquaintance_ASA_Frame, new_agent_frame);
    if(ASA_Frame)
    {
        _the_knowledge_base.Add_Frame(ASA_Frame);
        return;
    };

    Acquaintance_PPA_Frame    *PPA_Frame;

    PPA_Frame = TCL_DYNAMIC_CAST(Acquaintance_PPA_Frame, new_agent_frame);
    if(PPA_Frame)
    {
```

```
        _the_knowledge_base.Add_Frame(PPA_Frame);
        return;
    };

    Acquaintance_SSA_Frame    *SSA_Frame;

    SSA_Frame = TCL_DYNAMIC_CAST(Acquaintance_SSA_Frame, new_agent_frame);
    if(SSA_Frame)
    {
        _the_knowledge_base.Add_Frame(SSA_Frame);
        return;
    };

    Acquaintance_TSA_Frame    *TSA_Frame;

    TSA_Frame = TCL_DYNAMIC_CAST(Acquaintance_TSA_Frame, new_agent_frame);
    if(TSA_Frame)
    {
        _the_knowledge_base.Add_Frame(TSA_Frame);
        return;
    };
}

/*****************************************************************************
 *  Receive_Message (protected - virtual)
 *
 *  the default method simply puts a copy of the message at the back of
 *  the queue.  all FABRIC agents override this method.
 *
 *****************************************************************************/

void Agent::Receive_Message(const Message_Frame *message_to_receive)
{
    Base_Frame       *frame_base_class;
    Message_Frame    *copied_message;

    frame_base_class = message_to_receive->Clone();
    copied_message = TCL_DYNAMIC_CAST(Message_Frame, frame_base_class);

    _the_message_queue.Push_to_Back(copied_message);
}

/*****************************************************************************
 *  Execute (protected - virtual)
 *
 *  this virtual method must be overridden by each agent with its own
 *  specific form of execution.  the default method is empty.
 *
 *****************************************************************************/

void Agent::Execute(int type_of_event)
{
}
```

Appendix B

Program Listing of the Class Task_Scheduling_Agent

```
/*=========================================================================*/
/*      Title:      Task_Scheduling_Agent.h                                */
/*      Author: Jagdeep S. Basran                                          */
/*      Date:       June 6, 1996                                           */
/*      Comments:   header file for the class Task_Scheduling_Agent        */
/*=========================================================================*/

#ifndef TASK_SCHEDULING_AGENT_H
#define TASK_SCHEDULING_AGENT_H

//  include files needed

#include "Agent.h"
#include "Variance.h"

// Advance declaration of classes

class Bid_Frame;
class Grant_Frame;
class Message_Frame;
class Multi_Slot;
class Report_Frame;
class Task_Announcement_Frame;
class Task_Frame;

//  Class definition

class Task_Scheduling_Agent : public Agent
{

TCL_DECLARE_CLASS

private:

    //  private constants and variables

    enum State_of_TSA { IDLE=1, INITIALIZATION, ANNOUNCING_TASKS,
                        WAITING_FOR_MESSAGES, BIDS_MADE_FOR_TASK,
                        LAST_TASK_COMPLETED, ASSEMBLY_DONE };

    enum Transition_of_TSA { INIT_START=1, INIT_COMPLETE,
                             NO_TASKS_TO_LAUNCH, TASK_TO_LAUNCH,
                             TK_TIMES_OUT, SINGLE_BID_RECEIVED,
                             BIDS_NOT_MADE, BIDS_WERE_MADE, TK_TERMINATED,
                             LAST_TASK_DONE, LAST_TASK_NOT_DONE };

    Basic_Queue<Task_Frame *>     _ready_to_launch_tasks;
    unsigned short                _tasks_left_in_cycle;
    unsigned short                _cycles_left_to_do, _total_cycles;
    double                        _cycle_start_time;
    Variance                      _cycle_time_variance;
    Basic_List<CString>           _tasks_announced;
    Basic_List<double>            _tasks_expiration_time;
    Boolean                       _handling_previous_report;

    //  private methods

    Boolean  Precedence_Constraints_Satisfied(
                                        Multi_Slot *the_preceding_tasks);
    Boolean  Get_Random_Boolean(void);
    void     Shuffle_the_Ready_to_Launch_Tasks(void);
    void     Setup_All_Ready_to_Launch_Tasks(void);
    Task_Announcement_Frame *Create_Task_Announcement_Frame(
                    Task_Frame *the_task, const char *sender_identity,
```

```
                                                    double time_to_wait);
    void     Retransmit_Task_Announcement(CString &task_identity);
    Grant_Frame *Create_Grant_Task_Frame(Bid_Frame *the_bid);
    void     Select_Best_Bid_and_Grant_Task(
                                          Basic_List<Bid_Frame *> &the_bids);

    void     Check_Message_Queue_and_Generate_Event_if_Necessary(void);
    void     Record_Cycle_Time(
                             double &reset_assembly_time = *(double *) NULL);

protected:

    void     Set_Tasks_Left_in_Cycle(void);
    void     Setup_Knowledge_Base(void);

    //  protected constructor

    Task_Scheduling_Agent(const char *name_of_agent,
                                          const char *name_of_direct_parent);

    //  methods to implement its CSM

    void     Do_Initialization(void);
    void     Process_the_Bids(void);
    void     Process_Completed_Task(Report_Frame *the_report);
    Boolean  Have_Tasks_to_Launch(void);

    void     Execute_Initialization_State(State &current_state);
    void     Execute_Initialization_Completion(State &current_state);
    void     Execute_Single_Task_Announcement(void);
    void     Execute_Single_Task_Announcement_and_Generate_Event(void);
    void     Execute_Announce_Completion(State &current_state);
    void     Execute_Store_Single_Bid(State &current_state);
    void     Execute_Check_Bids_for_Task(State &current_state);
    void     Execute_Last_Task_Completed_State(State &current_state);
    void     Execute_Bids_Not_Made_Transition(State &current_state);
    void     Execute_Bids_Were_Made_Transition(State &current_state);
    void     Execute_Update_Ready_to_Launch(State &current_state);
    void     Execute_All_Tasks_Done(State &current_state);
    void     Execute_Assembly_Done_State(State &current_state);

    //  methods overridden

    virtual void Receive_Message(const Message_Frame *message_to_receive);
    virtual void Execute(int type_of_event);

public:

    //  Constructors and destructor

    Task_Scheduling_Agent(void);
    Task_Scheduling_Agent(const char *name_of_agent);

    virtual ~Task_Scheduling_Agent(void);

    //  public method

    void Set_Total_Number_of_Cycles(unsigned short number_of_cycles)
    {
        _cycles_left_to_do = _total_cycles = number_of_cycles;
    }
};

#endif
```

```
/*=========================================================================*/
/*      Title:      Task_Scheduling_Agent.cp                               */
/*      Parent:  Agent                                                     */
/*      Author:  Jagdeep S. Basran                                         */
/*      Date:       June 6, 1996                                           */
/*      Comments:   this class implements the behaviour of the task        */
/*                  scheduling agent (TSA) in a flexible agent-based       */
/*                  robotic assembly cell (FABRIC).                        */
/*=========================================================================*/

//  include files needed for this class

#include "Task_Scheduling_Agent.h"

#include "Acquaintance_ASA_Frame.h"
#include "Bid_Frame.h"
#include "Grant_Frame.h"
#include "Model_of_Agent_Frame.h"
#include "Report_Frame.h"
#include "Task_Frame.h"
#include "Task_Announcement_Frame.h"

#include "Multi_Slot.h"
#include "Simulator.h"

#include <Float.h>
#include <math.h>

TCL_DEFINE_CLASS_D1(Task_Scheduling_Agent, Agent)


/*****************************************************************************
 *  Precedence_Constraints_Satisfied (private)
 *
 *  this method returns TRUE if all of the precedence constraints of an
 *  assembly operation specified by the multi-slot are satisfied;
 *  otherwise, it returns FALSE.
 *
 *****************************************************************************/

Boolean Task_Scheduling_Agent::Precedence_Constraints_Satisfied(
    Multi_Slot *the_preceding_tasks)
{
    unsigned long    the_cardinality, slot_index;
    Generic_Slot     *slot_base_class;
    Slot<CString>    *the_task_string;
    Base_Frame       *frame_base_class;
    Task_Frame       *the_task;

    the_cardinality = the_preceding_tasks->Get_Current_Cardinality();
    for(slot_index=0; slot_index<the_cardinality; slot_index++)
    {
        slot_base_class = the_preceding_tasks->Get_Slot(slot_index);
        the_task_string = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
        frame_base_class = _the_knowledge_base.Get_Frame(
                                        the_task_string->Get_Slot_Value());
        the_task = TCL_DYNAMIC_CAST(Task_Frame, frame_base_class);
        if(!the_task->Task_Is_Completed())
            return(FALSE);
    };

    return(TRUE);
}
```

```
/*****************************************************************************
 *  Get_Random_Boolean (private)
 *
 *  this method randomly returns a zero or a one.
 *
 *****************************************************************************/

Boolean Task_Scheduling_Agent::Get_Random_Boolean(void)
{
    int     random_number;
    Boolean random_boolean;

    random_number = rand();
    random_boolean = (random_number << 1) / (RAND_MAX+1);
    return(random_boolean);
}

/*****************************************************************************
 *  Shuffle_the_Ready_to_Launch_tasks (private)
 *
 *  if there is more than one ready to run task, then this method
 *  shuffles the ready to run tasks.  this is done to ensure that the
 *  the contractor agents are given an equal opportunity to complete
 *  their tasks during the course of a simulation run.
 *
 *****************************************************************************/

void Task_Scheduling_Agent::Shuffle_the_Ready_to_Launch_Tasks(void)
{
    if(_ready_to_launch_tasks.Is_Empty())
        return;

    Basic_Queue<Task_Frame *>    temp_tasks;
    Task_Frame                   *current_task;
    long                         number_of_tasks=0;

    current_task = _ready_to_launch_tasks.Pop_from_Front();
    while(current_task)
    {
        temp_tasks.Push_to_Back(current_task);
        number_of_tasks++;
        current_task = _ready_to_launch_tasks.Pop_from_Front();
    };

    if(number_of_tasks==1)
        return;

    current_task = temp_tasks.Pop_from_Front();
    while(current_task)
    {
        if(Get_Random_Boolean())
            _ready_to_launch_tasks.Push_to_Back(current_task);
        else
            _ready_to_launch_tasks.Push_to_Front(current_task);

        current_task = temp_tasks.Pop_from_Front();
    };
}

/*****************************************************************************
 *  Setup_All_Ready_to_Launch_Tasks (private)
 *
 *  this method sets up the ready to launch task queue given the current
 *  state of the task frames in the knowledge base.
```

```
 *
 ****************************************************************************/

void Task_Scheduling_Agent::Setup_All_Ready_to_Launch_Tasks(void)
{
    Base_Frame        **the_task_frames;
    long              total_tasks;

    _the_knowledge_base.Get_All_Frames("Task_Frame", the_task_frames,
                                                          total_tasks);

    if(!total_tasks)
        return;

    Task_Frame        *the_task;
    long              index;

    for(index=0; index<total_tasks; index++)
    {
        the_task = TCL_DYNAMIC_CAST(Task_Frame, the_task_frames[index]);
        if(!the_task->Task_Is_Announced())
        {
            Generic_Slot      *slot_base_class;
            Multi_Slot        *the_preceding_tasks;
            unsigned long     the_cardinality;

            slot_base_class = the_task->Get_Slot(PRECEDING_TASKS);
            the_preceding_tasks = TCL_DYNAMIC_CAST(Multi_Slot,
                                                    slot_base_class);
            the_cardinality = the_preceding_tasks->
                                                Get_Current_Cardinality();
            if(!the_cardinality)
                _ready_to_launch_tasks.Push_to_Back(the_task);
            else if(Precedence_Constraints_Satisfied(the_preceding_tasks))
                _ready_to_launch_tasks.Push_to_Back(the_task);
        };
    };

    Shuffle_the_Ready_to_Launch_Tasks();
    delete [] the_task_frames;
}

/****************************************************************************
 *   Create_Task_Announcement_Frame (private)
 *
 *   given a task frame describing an assembly operation, this method
 *   creates, sets up, and returns a task announcement message as a frame.
 *   it is up to the user to free the memory of this frame when it is no
 *   longer needed.
 *
 ****************************************************************************/

Task_Announcement_Frame *Task_Scheduling_Agent::Create_Task_Announcement_Frame
    (Task_Frame *the_task, const char *sender_identity,
                                                    double time_to_wait)
{
    CString           task_string, task_abstraction;
    Generic_Slot      *slot_base_class;
    Slot<CString>     *assembly_operation, *part, *assembly_location;

    task_string = the_task->get_name_of_instance();
    task_abstraction = the_task->Get_Task_Abstraction();
    slot_base_class = the_task->Get_Slot(ASSEMBLY_OPERATION);
    assembly_operation = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
```

```
    slot_base_class = the_task->Get_Slot(PART_TO_ASSEMBLY);
    part = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    slot_base_class = the_task->Get_Slot(ASSEMBLY_LOCATION);
    assembly_location = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);

    Multi_Slot       eligibility_specification, bid_specification;
    CString          the_string;
    Slot<CString>    string_slot;

    the_string = CAN_DO_STRING;       the_string += DELIMITER_STRING;
    the_string += assembly_operation->Get_Slot_Value();
    string_slot.Set_Slot_Value(the_string);
    eligibility_specification.Append_Slot(&string_slot);

    the_string = CAN_GRASP_STRING;    the_string += DELIMITER_STRING;
    the_string += part->Get_Slot_Value();
    string_slot.Set_Slot_Value(the_string);
    eligibility_specification.Append_Slot(&string_slot);

    the_string = CAN_REACH_STRING;    the_string += DELIMITER_STRING;
    the_string += assembly_location->Get_Slot_Value();
    string_slot.Set_Slot_Value(the_string);
    eligibility_specification.Append_Slot(&string_slot);

    the_string = assembly_operation->Get_Slot_Value();
    the_string += DELIMITER_STRING;
    the_string += part->Get_Slot_Value();
    the_string += DELIMITER_STRING;
    the_string += AVERAGE_EXECUTION_TIME_STRING;
    string_slot.Set_Slot_Value(the_string);
    bid_specification.Append_Slot(&string_slot);

    Task_Announcement_Frame  *the_announcement;

    the_announcement = new Task_Announcement_Frame(task_string,
            sender_identity, NULL_STRING, task_abstraction, time_to_wait,
                        &eligibility_specification, &bid_specification);

    return(the_announcement);
}

/*****************************************************************************
 *  Retransmit_Task_Announcement (private)
 *
 *  this method simply retransmits the task announcement referred to by
 *  the task identity string.
 *
 *****************************************************************************/

void Task_Scheduling_Agent::Retransmit_Task_Announcement(
    CString &task_identity)
{
    Base_Frame       *frame_base_class;
    Task_Frame       *the_task;

    frame_base_class = _the_knowledge_base.Get_Frame(task_identity);
    the_task = TCL_DYNAMIC_CAST(Task_Frame, frame_base_class);
    _ready_to_launch_tasks.Push_to_Back(the_task);
    Execute_Single_Task_Announcement();
}

/*****************************************************************************
 *  Create_Grant_Task_Frame (private)
 *
```

```
 *   give the best bid from a contractor agent, this method creates a
 *   grant task message frame.  it is up to the user to free the memory
 *   of this message when it is no longer needed.
 *
 ****************************************************************************/

Grant_Frame *Task_Scheduling_Agent::Create_Grant_Task_Frame(
    Bid_Frame *the_bid)
{
    char             *sender_identity;
    Generic_Slot     *slot_base_class;
    Slot<CString>    *the_sender_slot;
    CString          task_identifier, task_specification;
    CString          agent_with_best_bid;
    Base_Frame       *frame_base_class;
    Task_Frame       *the_task;
    Grant_Frame      *the_grant;

    task_identifier = the_bid->get_name_of_instance();
    frame_base_class = _the_knowledge_base.Get_Frame(task_identifier);
    the_task = TCL_DYNAMIC_CAST(Task_Frame, frame_base_class);
    task_specification = the_task->Get_Task_Specification();

    sender_identity = get_agent_identity();
    slot_base_class = the_bid->Get_Slot(SENDER);
    the_sender_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    agent_with_best_bid = the_sender_slot->Get_Slot_Value();

    the_grant = new Grant_Frame(task_identifier, sender_identity,
                                agent_with_best_bid, task_specification);

    delete [] sender_identity;
    return(the_grant);
}

/****************************************************************************
 *   Select_Best_Bid_and_Grant_Task (private)
 *
 *   this method selects the best bid for an announced assembly task and
 *   then sends a grant message to all of the bidding agents.
 *
 ****************************************************************************/

void Task_Scheduling_Agent::Select_Best_Bid_and_Grant_Task(
    Basic_List<Bid_Frame *> &the_bids)
{
    size_t       best_index;
    Grant_Frame  *the_grant;

    Select_Best_Bid_Randomly(the_bids, best_index);

    the_grant = Create_Grant_Task_Frame(the_bids[best_index]);

    size_t           loop;
    size_t           total_bids = the_bids.number_of_elements();
    Generic_Slot     *slot_base_class;
    Slot<CString>    *the_sender_slot;
    const char       *recipient_identity;
    CString          recipient_string;

    for(loop=0; loop<total_bids; loop++)
    {
        slot_base_class = the_bids[loop]->Get_Slot(SENDER);
        the_sender_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
```

```
        recipient_string = the_sender_slot->Get_Slot_Value();
        recipient_identity = recipient_string;
        Send_Message(recipient_identity, the_grant);
    };

    delete the_grant;

    CString      task_identifier;
    Base_Frame   *frame_base_class;
    Task_Frame   *the_task;

    task_identifier = the_bids[best_index]->get_name_of_instance();
    frame_base_class = _the_knowledge_base.Get_Frame(task_identifier);
    the_task = TCL_DYNAMIC_CAST(Task_Frame, frame_base_class);
    the_task->Set_Task_State_to_Started();
}

/*****************************************************************************
 *  Check_Message_Queue_and_Generate_Event_if_Necessary (private)
 *
 *  this method gets the first message in the message queue and
 *  generates the corresponding event if necessary.
 *
 *****************************************************************************/

void Task_Scheduling_Agent::
                Check_Message_Queue_and_Generate_Event_if_Necessary(void)
{
    Message_Frame    *the_message;
    Report_Frame     *the_report;
    Bid_Frame        *the_bid;

    the_message = _the_message_queue.Pop_from_Front();
    if(!the_message)
        return;
    the_report = TCL_DYNAMIC_CAST(Report_Frame, the_message);
    the_bid = TCL_DYNAMIC_CAST(Bid_Frame, the_message);

    if(the_report)
    {
        the_simulator->Add_to_Event_List_with_Delay(0, this, TK_TERMINATED);
        _handling_previous_report = TRUE;
    }
    else if(the_bid)
        the_simulator->Add_to_Event_List_with_Delay(0, this,
                                                    SINGLE_BID_RECEIVED);

    _the_message_queue.Push_to_Front(the_message);
}

/*****************************************************************************
 *  Record_Cycle_Time (private)
 *
 *  this method records the cycle time for a single assembly cycle.
 *
 *****************************************************************************/

void Task_Scheduling_Agent::Record_Cycle_Time(double &reset_assembly_time)
{
    double  the_reset_time  = (*_the_mean_is_1) ();
    double  simulation_time = the_simulator->Get_Current_Simulation_Time();
    double  cycle_end_time;

    cycle_end_time = simulation_time + the_reset_time;
```

```
    _cycle_time_variance += (cycle_end_time - _cycle_start_time);
    _cycle_start_time = cycle_end_time;

    if(&reset_assembly_time)
        reset_assembly_time = the_reset_time;
}

/*****************************************************************************
 *  Set_Tasks_Left_in_Cycle (protected)
 *
 *  this method initially sets the number of assembly tasks in an
 *  assembly cycle.
 *
 *****************************************************************************/

void Task_Scheduling_Agent::Set_Tasks_Left_in_Cycle(void)
{
    Base_Frame   **the_task_frames;
    long         total_tasks;

    _the_knowledge_base.Get_All_Frames("Task_Frame", the_task_frames,
                                                          total_tasks);
    _tasks_left_in_cycle = total_tasks;

    delete [] the_task_frames;
}

/*****************************************************************************
 *  Setup_Knowledge_Base (protected)
 *
 *  this method sets up some basic knowledge of the task scheduling agent.
 *
 *****************************************************************************/

void Task_Scheduling_Agent::Setup_Knowledge_Base(void)
{
    State   startup_state(IDLE, IDLE_STRING);

    set_current_state(startup_state);

    _cycle_start_time = 0.0;
    _handling_previous_report = FALSE;
    Set_Total_Number_of_Cycles(50);

    the_simulator->Add_to_Event_List_at_Time(0, this, INIT_START);
}

/*****************************************************************************
 *  Task_Scheduling_Agent (constructor - protected)
 *
 *  this constructor initializes the task scheduling agent with the given
 *  name and the given name for the direct parent.  this is the primary
 *  agent constructor for derived TSAs.
 *
 *****************************************************************************/

Task_Scheduling_Agent::Task_Scheduling_Agent(
    const char *name_of_agent, const char *name_of_direct_parent) :
                            Agent(name_of_agent, name_of_direct_parent)
{
    Setup_Knowledge_Base();
}

/*****************************************************************************
```

```
 *  Do_Initialization (protected)
 *
 *  this method initializes the task scheduling agent at the start of
 *  an assembly cycle.
 *
 ******************************************************************************/

void Task_Scheduling_Agent::Do_Initialization(void)
{
    Setup_All_Ready_to_Launch_Tasks();
}

/******************************************************************************
 *  Process_the_Bids (protected)
 *
 *  this method processes the bids that were received when a task
 *  announcement times out.
 *
 ******************************************************************************/

void Task_Scheduling_Agent::Process_the_Bids(void)
{
    double   simulation_time = the_simulator->Get_Current_Simulation_Time();
    size_t   index;
    long     time_difference;
    CString  task_identity;

    for(index=0; index<_tasks_expiration_time.number_of_elements(); index++)
    {
        time_difference = (simulation_time - _tasks_expiration_time[index])
                                                                    * 10000;
        if(time_difference >= 0)
            break;
    };

    task_identity = _tasks_announced[index];
    _tasks_announced.Remove_At(index);
    _tasks_expiration_time.Remove_At(index);

    Basic_List<Bid_Frame *>  the_bids;
    double                   time_to_process=0.15;

    Get_Bids_for_Task(task_identity, the_bids);
    if(!the_bids.number_of_elements())
    {
        Retransmit_Task_Announcement(task_identity);
        the_simulator->Add_to_Event_List_with_Delay(time_to_process, this,
                                                    BIDS_NOT_MADE);
        return;
    };

    Select_Best_Bid_and_Grant_Task(the_bids);

    size_t   loop;

    for(loop=0; loop<the_bids.number_of_elements(); loop++)
        delete the_bids[loop];

    the_simulator->Add_to_Event_List_with_Delay(time_to_process, this,
                                                BIDS_WERE_MADE);
}

/******************************************************************************
 *  Process_Completed_Task (protected)
```

```
 *
 *   this method processes the completion of an assembly task that is at
 *   the front of the message queue.
 *
 ****************************************************************************/

void Task_Scheduling_Agent::Process_Completed_Task(Report_Frame *the_report)
{
    CString      task_identity;
    Base_Frame   *frame_base_class;
    Task_Frame   *the_task;

    task_identity = the_report->get_name_of_instance();
    frame_base_class = _the_knowledge_base.Get_Frame(task_identity);
    the_task = TCL_DYNAMIC_CAST(Task_Frame, frame_base_class);
    if(the_task)
    {
        the_task->Set_Task_State_to_Completed();

        CString  trace_message(SINGLE_QUOTE_STRING);

        trace_message += task_identity;
        trace_message += QUOTE_COMPLETED_STRING;
        Register_Trace_Message(trace_message);
    };
}

/****************************************************************************
 *   Have_Tasks_to_Launch (protected)
 *
 *   this method sets up all of the ready to launch tasks and then returns
 *   TRUE if tasks remain to launch; otherwise, it returns FALSE.
 *
 ****************************************************************************/

Boolean Task_Scheduling_Agent::Have_Tasks_to_Launch(void)
{
    Setup_All_Ready_to_Launch_Tasks();

    return(!(_ready_to_launch_tasks.Is_Empty()));
}

/****************************************************************************
 *   Execute_Initialization_State (protected)
 *
 *   this method defines the execution of the initialization state for the
 *   task scheduling agent.
 *
 ****************************************************************************/

void Task_Scheduling_Agent::Execute_Initialization_State(
    State &current_state)
{
    current_state.set_state(INITIALIZATION, INITIALIZATION_STRING);
    set_current_state(current_state);

    Register_Trace_Message(START_INITIALIZATION_STRING);

    Do_Initialization();

    double  time_to_initialize=0.005;

    the_simulator->Add_to_Event_List_with_Delay(time_to_initialize, this,
                                                INIT_COMPLETE);
```

```
}

/*****************************************************************************
 *   Execute_Initialization_Completion (protected)
 *
 *   this method defines the execution of exiting the initialization state
 *   for the task scheduling agent.
 *
 *****************************************************************************/

void Task_Scheduling_Agent::Execute_Initialization_Completion(
    State &current_state)
{
    current_state.set_state(ANNOUNCING_TASKS,
                                          ANNOUNCING_READY_TASKS_STRING);
    set_current_state(current_state);

    Register_Trace_Message(ANNOUNCING_READY_TASKS_STRING);

    int type_of_event;

    if(_ready_to_launch_tasks.Is_Empty())
        type_of_event=NO_TASKS_TO_LAUNCH;
    else
        type_of_event=TASK_TO_LAUNCH;

    the_simulator->Add_to_Event_List_with_Delay(0, this, type_of_event);
}

/*****************************************************************************
 *   Execute_Single_Task_Announcement (protected)
 *
 *   this method announces a single assembly operation, task to the
 *   assembling agents.
 *
 *****************************************************************************/

void Task_Scheduling_Agent::Execute_Single_Task_Announcement(void)
{
    Task_Frame   *the_task;

    the_task = _ready_to_launch_tasks.Pop_from_Front();
    if(the_task)
    {
        Task_Announcement_Frame   *the_announcement;
        char                      *sender_identity;
        double                    waiting_time=0.5;
        CString                   task_identity;

        sender_identity = get_agent_identity();
        the_announcement = Create_Task_Announcement_Frame(the_task,
                                          sender_identity, waiting_time);

        Send_Limited_Broadcast("Acquaintance_ASA_Frame", the_announcement);

        the_simulator->Add_to_Event_List_with_Delay(waiting_time, this,
                                                    TK_TIMES_OUT);

        delete the_announcement;
        delete [] sender_identity;

        task_identity = the_task->get_name_of_instance();
        _tasks_announced.Add_to_Back(task_identity);
        _tasks_expiration_time.Add_to_Back(the_simulator->
```

```
                                Get_Current_Simulation_Time() + waiting_time);
            the_task->Set_Task_State_to_Announced();
        };
}

/*************************************************************************
 *  Execute_Single_Task_Announcement_and_Generate_Event (protected)
 *
 *  this method announces a single assembly operation, task to the
 *  assembling agents and then generates the proper event for the first
 *  announcement of a task.
 *
 *************************************************************************/

void Task_Scheduling_Agent::
    Execute_Single_Task_Announcement_and_Generate_Event(void)
{
    Execute_Single_Task_Announcement();

    int type_of_event;

    if(_ready_to_launch_tasks.Is_Empty())
        type_of_event=NO_TASKS_TO_LAUNCH;
    else
        type_of_event=TASK_TO_LAUNCH;

    double  time_to_launch=0.15;

    the_simulator->Add_to_Event_List_with_Delay(time_to_launch, this,
                                                    type_of_event);
}

/*************************************************************************
 *  Execute_Announce_Completion (protected)
 *
 *  this method changes the state of the assembling agent after completing
 *  the announcements for all ready to launch tasks.
 *
 *************************************************************************/

void Task_Scheduling_Agent::Execute_Announce_Completion(
    State &current_state)
{
    current_state.set_state(WAITING_FOR_MESSAGES,
                                        WAITING_FOR_MESSAGES_STRING);
    set_current_state(current_state);

    Register_Trace_Message(WAITING_FOR_MESSAGES_STRING);

    Check_Message_Queue_and_Generate_Event_if_Necessary();
}

/*************************************************************************
 *  Execute_Store_Single_Bid (protected)
 *
 *  this method simply notes that a single bid was received by the
 *  task scheduling agent.
 *
 *************************************************************************/

void Task_Scheduling_Agent::Execute_Store_Single_Bid(State &current_state)
{
    Register_Trace_Message(SINGLE_BID_MESSAGE_RECEIVED);
}
```

```
/*****************************************************************************
 *   Execute_Check_Bids_for_Task (protected)
 *
 *   this method checks whether bids were received for a previous task
 *   announcement and responds accordingly.
 *
 *****************************************************************************/

void Task_Scheduling_Agent::Execute_Check_Bids_for_Task(
    State &current_state)
{
    current_state.set_state(BIDS_MADE_FOR_TASK, BIDS_MADE_FOR_TASK_STRING);
    set_current_state(current_state);

    Register_Trace_Message(CHECKING_BIDS_FOR_TASK);

    Process_the_Bids();
}

/*****************************************************************************
 *   Execute_Last_Task_Completed_State (protected)
 *
 *   this method defines the execution of the last task completed state
 *   for the task scheduling agent.
 *
 *****************************************************************************/

void Task_Scheduling_Agent::Execute_Last_Task_Completed_State(
    State &current_state)
{
    Report_Frame      *the_report;
    Generic_Slot      *slot_base_class;
    Slot<CString>     *report_slot;
    CString           report_type;
    const char        *report_type_ptr;

    if(!Get_the_Report_Message(the_report))
        return;

    slot_base_class = the_report->Get_Slot(TYPE_OF_REPORT);
    report_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    report_type = report_slot->Get_Slot_Value();
    report_type_ptr = report_type;
    if(strcmp(report_type_ptr, TASK_TERMINATED_REPORT))
        return;

    current_state.set_state(LAST_TASK_COMPLETED,
                                                LAST_TASK_COMPLETED_STRING);
    set_current_state(current_state);

    Register_Trace_Message(START_CHECK_LAST_TASK_STRING);

    Process_Completed_Task(the_report);
    delete the_report;
    _handling_previous_report = FALSE;

    double  time_to_check=0.001;

    _tasks_left_in_cycle--;
    if(Have_Tasks_to_Launch() || _tasks_left_in_cycle)
        the_simulator->Add_to_Event_List_with_Delay(time_to_check, this,
                                                    LAST_TASK_NOT_DONE);
    else
```

```
        the_simulator->Add_to_Event_List_with_Delay(time_to_check, this,
                                                    LAST_TASK_DONE);
}

/*************************************************************************
 *   Execute_Bids_Not_Made_Transition (protected)
 *
 *   this method makes the transition back to the waiting state after
 *   no bids were received for a previous task announcement.
 *
 *************************************************************************/

void Task_Scheduling_Agent::Execute_Bids_Not_Made_Transition(
    State &current_state)
{
    current_state.set_state(WAITING_FOR_MESSAGES,
                                        WAITING_FOR_MESSAGES_STRING);
    set_current_state(current_state);

    Register_Trace_Message(WAITING_FOR_MESSAGES_STRING);

    Check_Message_Queue_and_Generate_Event_if_Necessary();
}

/*************************************************************************
 *   Execute_Bids_Were_Made_Transition (protected)
 *
 *   this method makes the transition back to the waiting state after
 *   bids were received for a previous task announcement.
 *
 *************************************************************************/

void Task_Scheduling_Agent::Execute_Bids_Were_Made_Transition(
    State &current_state)
{
    current_state.set_state(WAITING_FOR_MESSAGES,
                                        WAITING_FOR_MESSAGES_STRING);
    set_current_state(current_state);

    Register_Trace_Message(WAITING_FOR_MESSAGES_STRING);

    Check_Message_Queue_and_Generate_Event_if_Necessary();
}

/*************************************************************************
 *   Execute_Update_Ready_to_Launch (protected)
 *
 *   this method updates the ready to launch tasks, after an assembly
 *   task was completed.
 *
 *************************************************************************/

void Task_Scheduling_Agent::Execute_Update_Ready_to_Launch(
    State &current_state)
{
    Execute_Initialization_Completion(current_state);
}

/*************************************************************************
 *   Execute_All_Tasks_Done (protected)
 *
 *   this method executes when all of the assembly tasks for a single
 *   assembly cycle have been completed.
 *
```

```
 ***************************************************************************/

void Task_Scheduling_Agent::Execute_All_Tasks_Done(State &current_state)
{
    current_state.set_state(ASSEMBLY_DONE, ASSEMBLY_DONE_STRING);
    set_current_state(current_state);

    the_simulator->Add_to_Event_List_with_Delay(0, this, INIT_START);
}

/***************************************************************************
 *  Execute_Assembly_Done_State (protected)
 *
 *  this method determines whether another cycle of this assembly cell
 *  needs to be simulated and also records the cycle time for the just
 *  completed cycle.
 *
 ***************************************************************************/

void Task_Scheduling_Agent::Execute_Assembly_Done_State(
    State &current_state)
{
    Register_Trace_Message(ASSEMBLY_DONE_STRING);

    printf("Cycle number = %d\n", (_total_cycles - _cycles_left_to_do + 1));
    _cycles_left_to_do--;
    if(!_cycles_left_to_do)
    {
        Record_Cycle_Time();
        _cycle_time_variance.print(cout);
        return;
    };

    Base_Frame   **the_task_frames;
    long         total_tasks;

    _the_knowledge_base.Get_All_Frames("Task_Frame", the_task_frames,
                                                        total_tasks);

    if(!total_tasks)
        return;

    Task_Frame       *the_task;
    long             index;

    for(index=0; index<total_tasks; index++)
    {
        the_task = TCL_DYNAMIC_CAST(Task_Frame, the_task_frames[index]);
        the_task->Set_Task_State_to_Unannounced();
    };

    _tasks_left_in_cycle = total_tasks;
    delete [] the_task_frames;

    double   reset_assembly_time;

    Record_Cycle_Time(reset_assembly_time);

    current_state.set_state(IDLE, IDLE_STRING);
    set_current_state(current_state);

    the_simulator->Add_to_Event_List_with_Delay(reset_assembly_time, this,
                                                        INIT_START);
}
```

```
/*************************************************************************
 *  Receive_Message (protected - override)
 *
 *  this method makes a copy of the given message and responds to it
 *  differently depending upon its type and the current state of the TSA.
 *
 *************************************************************************/

void Task_Scheduling_Agent::Receive_Message(
    const Message_Frame *message_to_receive)
{
    Base_Frame       *frame_base_class;
    Message_Frame    *copied_message;
    Report_Frame     *the_report;
    Bid_Frame        *the_bid;
    double           message_time, message_delay = (*_the_message_delay)();

    frame_base_class = message_to_receive->Clone();
    copied_message = TCL_DYNAMIC_CAST(Message_Frame, frame_base_class);
    message_time = copied_message->Get_Message_Time() + message_delay;
    the_report = TCL_DYNAMIC_CAST(Report_Frame, copied_message);
    the_bid = TCL_DYNAMIC_CAST(Bid_Frame, copied_message);

    if(the_report)
        _the_message_queue.Push_to_Front(copied_message);
    else if(the_bid)
        _the_message_queue.Push_to_Back(copied_message);

    State    current_state = get_current_state();

    switch (current_state.get_state_id())
    {
        case WAITING_FOR_MESSAGES:
            if(the_report && (!_handling_previous_report))
            {
                the_simulator->Add_to_Event_List_at_Time(message_time, this,
                                                         TK_TERMINATED);
                _handling_previous_report = TRUE;
            }
            else if(the_bid)
                the_simulator->Add_to_Event_List_at_Time(message_time, this,
                                                         SINGLE_BID_RECEIVED);
            break;
    };
}

/*************************************************************************
 *  Execute (protected - override)
 *
 *  this method defines the execution of a task scheduling agent.
 *
 *************************************************************************/

void Task_Scheduling_Agent::Execute(int type_of_event)
{
    State    current_state = get_current_state();

    switch (current_state.get_state_id())
    {
        case IDLE:
            if(type_of_event==INIT_START)
                Execute_Initialization_State(current_state);
            break;
```

```
            case INITIALIZATION:
                if(type_of_event==INIT_COMPLETE)
                    Execute_Initialization_Completion(current_state);
                break;
            case ANNOUNCING_TASKS:
                if(type_of_event==TASK_TO_LAUNCH)
                    Execute_Single_Task_Announcement_and_Generate_Event();
                else if(type_of_event==NO_TASKS_TO_LAUNCH)
                    Execute_Announce_Completion(current_state);
                else if(type_of_event==TK_TIMES_OUT)
                    the_simulator->Add_to_Event_List_with_Delay(0.05, this,
                                                                type_of_event);
                break;
            case WAITING_FOR_MESSAGES:
                if(type_of_event==SINGLE_BID_RECEIVED)
                    Execute_Store_Single_Bid(current_state);
                else if(type_of_event==TK_TIMES_OUT)
                    Execute_Check_Bids_for_Task(current_state);
                else if(type_of_event==TK_TERMINATED)
                    Execute_Last_Task_Completed_State(current_state);
                break;
            case BIDS_MADE_FOR_TASK:
                if(type_of_event==BIDS_NOT_MADE)
                    Execute_Bids_Not_Made_Transition(current_state);
                else if(type_of_event==BIDS_WERE_MADE)
                    Execute_Bids_Were_Made_Transition(current_state);
                else if(type_of_event==TK_TIMES_OUT)
                    the_simulator->Add_to_Event_List_with_Delay(0.05, this,
                                                                type_of_event);
                break;
            case LAST_TASK_COMPLETED:
                if(type_of_event==LAST_TASK_NOT_DONE)
                    Execute_Update_Ready_to_Launch(current_state);
                else if(type_of_event==LAST_TASK_DONE)
                    Execute_All_Tasks_Done(current_state);
                else if(type_of_event==TK_TIMES_OUT)
                    the_simulator->Add_to_Event_List_with_Delay(0.05, this,
                                                                type_of_event);
                break;
            case ASSEMBLY_DONE:
                Execute_Assembly_Done_State(current_state);
                break;
        }
    }

/*****************************************************************************
 *  Task_Scheduling_Agent (default constructor - public)
 *
 *  the default constructor is needed for run-time type identification and
 *  is not meant to be used.
 *
 *****************************************************************************/

Task_Scheduling_Agent::Task_Scheduling_Agent(void) : Agent()
{
}

/*****************************************************************************
 *  Task_Scheduling_Agent (constructor - public)
 *
 *  this constructor initializes the task scheduling agent with the name
 *  specified by the user.
 *
 *****************************************************************************/
```

```
Task_Scheduling_Agent::Task_Scheduling_Agent(const char *name_of_agent) :
                      Agent(name_of_agent, TCL_CLASSNAME_FROM_POINTER(this))
{
    Setup_Knowledge_Base();
}

/*****************************************************************************
 *   ~Task_Scheduling_Agent (destructor - public)
 *
 *   this destructor has nothing to free.
 *
 *****************************************************************************/

Task_Scheduling_Agent::~Task_Scheduling_Agent(void)
{
}
```

Appendix C

Program Listing of the Class Assembling_Agent

```
/*============================================================================*/
/*      Title:      Assembling_Agent.h                                        */
/*      Author: Jagdeep S. Basran                                             */
/*      Date:       June 6, 1996                                              */
/*      Comments:   header file for the class Assembling_Agent                */
/*============================================================================*/

#ifndef ASSEMBLING_AGENT_H
#define ASSEMBLING_AGENT_H

//  include files needed

#include "Agent.h"

// Advance declaration of classes

class Grant_Frame;
class Multi_Slot;
class Report_Frame;
class Task_Announcement_Frame;

//  Class definition

class Assembling_Agent : public Agent
{

TCL_DECLARE_CLASS

private:

    //  private constants and variables

    enum State_of_ASA { IDLE=1, WAITING_FOR_GRANT, WAITING_FOR_BIDS,
                        BID_MADE_FOR_PART_REQUEST, WAITING_FOR_PART,
                        PART_GRASPED, WAITING_FOR_FREE_REGION,
                        PART_SUCCESSFULLY_ASSEMBLED,
                                        TRY_PART_ACQUISITION_AGAIN };

    enum Transition_of_ASA { TASK_LAUNCHED, BID_FOR_TASK, TASK_NOT_GRANTED,
                             TASK_GRANTED, SINGLE_BID_RECEIVED,
                             REQUEST_PART_TIMES_OUT, BIDS_NOT_MADE,
                             BIDS_WERE_MADE, PART_POSE_RECEIVED,
                             PART_GRASPED_SUCCESSFULLY,
                             PART_NOT_GRASPED_SUCCESSFULLY,
                             REGION_REQUEST_GRANTED, PART_WAS_ASSEMBLED,
                                 PART_WAS_NOT_ASSEMBLED, TRY_AGAIN } ;

    #define REQUEST_REGION_STRING          "request_region"
    #define RELEASE_REGION_STRING          "release_region"

    Report_Frame              *_pickup_message;
    CString                   _identity_of_PPA, _current_fetch_part_task;
    Task_Announcement_Frame   *_potential_task;
    Boolean                   _checking_qualifications;

    //  private methods

    CString  Get_Fetch_Part_Abstraction(Grant_Frame *the_task);
    CString  Get_Fetch_Part_Specification(Grant_Frame *the_task);
    Task_Announcement_Frame *Create_Task_Announcement_Frame(
                    Grant_Frame *the_task, const char *sender_identity,
                                                   double time_to_wait);
    void     Request_Part(Grant_Frame *the_grant);
    void     Retransmit_Task_Announcement(void);
```

```
    Grant_Frame *Create_Grant_Task_Frame(Bid_Frame *the_bid);
    void    Send_Not_Granted_Task_Message(Bid_Frame *the_bid);
    void    Select_Best_Bid_and_Grant_Task(
                                        Basic_List<Bid_Frame *> &the_bids);

    void    Setup_Pickup_Message(Report_Frame *the_report,
                                                Boolean grasp_successful);
    void    Request_Region_of_Assembly_Location(void);
    void    Release_Region(void);

    void    Remove_Any_Bid_Messages_from_Queue(void);
    void    Check_Message_Queue_and_Generate_Event_if_Necessary(void);

    void    Get_Delimiters(CString &task_specification,
                long &first_delimiter = *((long *) NULL),
                long &second_delimiter = *((long *) NULL),
                long &third_delimiter = *((long *) NULL));
protected:

    //  protected constant

    #define CURRENT_TASK "current task"

    //  parsing methods and method to get current assembly task

    CString Get_Assembly_Operation(CString &task_specification);
    CString Get_Part_Identity(CString &task_specification);
    CString Get_Assembly_Location(CString &task_specification);
    Grant_Frame  *Get_Current_Task(void);

    //  methods to add capabilities to an ASA's model of self.

    void    Add_Can_Do_Capability(CString &assembly_operation);
    void    Add_Can_Grasp_Capability(CString &the_part);
    void    Add_Can_Reach_Capability(CString &assembly_location);
    void    Add_Execution_Time_for_Capability(CString &assembly_operation,
                CString &the_part, double average_execution_time);

    void    Setup_Knowledge_Base(void);

    //  protected constructor

    Assembling_Agent(const char *name_of_agent,
                                        const char *name_of_direct_parent);

    //  virtual protected methods of an ASA

    virtual Boolean Move_and_Grasp_Part(Report_Frame *the_report,
                                                double &time_to_grasp_part);
    virtual void    Move_to_Assembly_Location(double &time_to_move);
    virtual Boolean Assemble_the_Part(double &time_to_assemble);
    virtual void    Discard_the_Part(double &time_to_discard_part);

    //  methods to implement its CSM

    void    Process_the_Bids(void);

    void    Execute_Check_if_Qualified(State &current_state);
    void    Execute_Submit_Bid_for_Task(State &current_state);
    void    Execute_Request_Part(State &current_state);
    void    Execute_Task_Not_Granted(State &current_state);
    void    Execute_Store_Single_Bid(State &current_state);
    void    Execute_Check_Bids_for_Task(State &current_state);
```

```
    void    Execute_Bids_Not_Made_Transition(State &current_state);
    void    Execute_Grant_Part_Task(State &current_state);
    void    Execute_Grasp_Part(State &current_state);
    void    Execute_Part_Grasped_Transition(State &current_state);
    void    Execute_Part_Not_Grasped_Transition(State &current_state);
    void    Execute_Assemble_Part(State &current_state);
    void    Execute_Successful_Assembly_Transition(State &current_state);
    void    Execute_Unsuccessful_Assembly_Transition(State &current_state);
    void    Execute_Request_Part_Again(State &current_state);

    //  methods overridden

    virtual void Receive_Message(const Message_Frame *message_to_receive);
    virtual void Execute(int type_of_event);

public:

    //  Constructors and destructor

    Assembling_Agent(void);
    Assembling_Agent(const char *name_of_agent);

    virtual ~Assembling_Agent(void);

};

#endif
```

```
/*==========================================================================*/
/*      Title:      Assembling_Agent.cp                                     */
/*      Parent:  Agent                                                      */
/*      Author:  Jagdeep S. Basran                                          */
/*      Date:       June 6, 1996                                            */
/*      Comments:   this class implements the generic behaviour of an       */
/*                  assembling agent (ASA) in a flexible agent-based        */
/*                  robotic assembly cell (FABRIC).                         */
/*==========================================================================*/

//  include files needed for this class

#include "Assembling_Agent.h"

#include "Bid_Frame.h"
#include "Grant_Frame.h"
#include "Model_of_Agent_Frame.h"
#include "Report_Frame.h"
#include "Task_Announcement_Frame.h"
#include "Task_Frame.h"

#include "Multi_Slot.h"
#include "RPY_Location.h"
#include "Simulator.h"

TCL_DEFINE_CLASS_D1(Assembling_Agent, Agent)


/*****************************************************************************
 *  Get_Fetch_Part_Abstraction (private)
 *
 *  given a granted assembly task, this method returns the task-level
 *  abstraction to send to all part presentation agents.
 *
 *****************************************************************************/

CString Assembling_Agent::Get_Fetch_Part_Abstraction(
    Grant_Frame *the_task)
{
    Generic_Slot     *slot_base_class;
    Slot<CString>    *task_slot;
    CString          task_specification, part_id, task_string;

    slot_base_class = the_task->Get_Slot(TASK_SPECIFICATION);
    task_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);

    task_specification = task_slot->Get_Slot_Value();
    part_id = Get_Part_Identity(task_specification);

    task_string = FETCH_PART_STRING;  task_string += DELIMITER_STRING;
    task_string += part_id;

    return(task_string);
}

/*****************************************************************************
 *  Get_Fetch_Part_Specification (private)
 *
 *  given a granted assembly task, this method returns the task-level
 *  specification to send to all part presentation agents.  currently,
 *  the specification and the abstraction are the same.
 *
 *****************************************************************************/
```

```
CString Assembling_Agent::Get_Fetch_Part_Specification(
    Grant_Frame *the_task)
{
    return(Get_Fetch_Part_Abstraction(the_task));
}

/*****************************************************************************
 *  Create_Task_Announcement_Frame (private)
 *
 *  given the three parameters, this method creates a message to announce
 *  the task of fetching a part for an ASA.  it is up to the user to free
 *  the memory of this message frame when it is no longer needed.
 *
 *****************************************************************************/

Task_Announcement_Frame *Assembling_Agent::Create_Task_Announcement_Frame(
    Grant_Frame *the_task,
                    const char *sender_identity, double time_to_wait)
{
    CString  task_string, task_abstraction;

    task_abstraction = Get_Fetch_Part_Abstraction(the_task);
    task_string = the_task->get_name_of_instance();
    task_string += DELIMITER_STRING;
    task_string += FETCH_PART_STRING;
    _current_fetch_part_task = task_string;

    Generic_Slot     *slot_base_class;
    Slot<CString>    *task_slot;
    CString          task_specification, part_id;

    slot_base_class = the_task->Get_Slot(TASK_SPECIFICATION);
    task_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    task_specification = task_slot->Get_Slot_Value();
    part_id = Get_Part_Identity(task_specification);

    Multi_Slot       eligibility_specification, bid_specification;
    CString          the_string;
    Slot<CString>    string_slot;
    int              index;
    const char       *sender_name;


    the_string = CAN_DELIVER_STRING;       the_string += DELIMITER_STRING;
    the_string += part_id;
    string_slot.Set_Slot_Value(the_string);
    eligibility_specification.Append_Slot(&string_slot);

    index=0;
    while(sender_identity[index]!=DELIMITER_CHAR)
        index++;
    sender_name = &(sender_identity[index+1]);
    the_string = CAN_DELIVER_TO_STRING;        the_string += DELIMITER_STRING;
    the_string += sender_name;
    string_slot.Set_Slot_Value(the_string);
    eligibility_specification.Append_Slot(&string_slot);

    the_string = task_abstraction;             the_string += DELIMITER_STRING;
    the_string += AVERAGE_EXECUTION_TIME_STRING;
    string_slot.Set_Slot_Value(the_string);
    bid_specification.Append_Slot(&string_slot);

    Task_Announcement_Frame   *the_announcement;
```

```
    the_announcement = new Task_Announcement_Frame(task_string,
                        sender_identity, NULL_STRING, task_abstraction,
                            time_to_wait, &eligibility_specification,
                                                    &bid_specification);

    return(the_announcement);
}

/*****************************************************************************
 *  Request_Part (private)
 *
 *  this method requests the part required for an assembly task that was
 *  granted to an ASA by the specified grant message.
 *
 *****************************************************************************/

void Assembling_Agent::Request_Part(Grant_Frame *the_grant)
{
    Task_Announcement_Frame   *the_announcement;
    char                      *sender_identity;
    double                    waiting_time=0.5;

    sender_identity = get_agent_identity();
    the_announcement = Create_Task_Announcement_Frame(the_grant,
                                            sender_identity, waiting_time);

    Send_Limited_Broadcast("Acquaintance_PPA_Frame", the_announcement);

    the_simulator->Add_to_Event_List_with_Delay(waiting_time, this,
                                                    REQUEST_PART_TIMES_OUT);
    delete the_announcement;
    delete [] sender_identity;
}

/*****************************************************************************
 *  Retransmit_Task_Announcement (private)
 *
 *  this method simply retransmits an announcement message for the
 *  part fetching task of the current assembly operation.
 *
 *****************************************************************************/

void Assembling_Agent::Retransmit_Task_Announcement(void)
{
    Grant_Frame      *current_task = Get_Current_Task();

    Request_Part(current_task);
}

/*****************************************************************************
 *  Create_Grant_Task_Frame (private)
 *
 *  give the best bid from a part presentation agent, this method creates
 *  a grant task message frame.  it is up to the user to free the memory
 *  of this message when it is no longer needed.
 *
 *****************************************************************************/

Grant_Frame *Assembling_Agent::Create_Grant_Task_Frame(Bid_Frame *the_bid)
{
    CString          task_string, assembly_task_id, task_specification;
    char             *sender_identity;
    Generic_Slot     *slot_base_class;
    Slot<CString>    *the_sender_slot;
```

```
    CString          agent_with_best_bid;
    Base_Frame       *frame_base_class;
    Grant_Frame      *the_task, *the_grant;

    task_string = the_bid->get_name_of_instance();
    assembly_task_id = Get_Assembly_Operation(task_string);
    frame_base_class = _the_knowledge_base.Get_Frame(assembly_task_id);
    the_task = TCL_DYNAMIC_CAST(Grant_Frame, frame_base_class);
    task_specification = Get_Fetch_Part_Specification(the_task);

    sender_identity = get_agent_identity();
    slot_base_class = the_bid->Get_Slot(SENDER);
    the_sender_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    agent_with_best_bid = the_sender_slot->Get_Slot_Value();

    the_grant = new Grant_Frame(task_string, sender_identity,
                                agent_with_best_bid, task_specification);

    delete [] sender_identity;
    return(the_grant);
}

/*****************************************************************************
 *   Send_Not_Granted_Task_Message (private)
 *
 *   this method sends a not granted task message to the agent who sent the
 *   specified bid.
 *
 *****************************************************************************/

void Assembling_Agent::Send_Not_Granted_Task_Message(Bid_Frame *the_bid)
{
    CString          task_string;
    char             *sender_identity;
    Grant_Frame      *the_grant;

    task_string = the_bid->get_name_of_instance();
    sender_identity = get_agent_identity();

    the_grant = new Grant_Frame(task_string, sender_identity,
                                                NULL_STRING, NULL_STRING);
    delete [] sender_identity;

    Generic_Slot     *slot_base_class;
    Slot<CString>    *the_sender_slot;
    const char       *recipient_identity;
    CString          recipient_string;

    slot_base_class = the_bid->Get_Slot(SENDER);
    the_sender_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    recipient_string = the_sender_slot->Get_Slot_Value();
    recipient_identity = recipient_string;
    Send_Message(recipient_identity, the_grant);

    delete the_grant;
}

/*****************************************************************************
 *   Select_Best_Bid_and_Grant_Task (private)
 *
 *   this method selects the best bid for an announced part fetching task
 *   and then sends a grant message to all of the bidding agents.
 *
 *****************************************************************************/
```

```
void Assembling_Agent::Select_Best_Bid_and_Grant_Task(
    Basic_List<Bid_Frame *> &the_bids)
{
    size_t       best_index;
    Grant_Frame  *the_grant;

    Select_Best_Bid_Randomly(the_bids, best_index);

    the_grant = Create_Grant_Task_Frame(the_bids[best_index]);

    size_t           loop;
    size_t           total_bids = the_bids.number_of_elements();
    Generic_Slot     *slot_base_class;
    Slot<CString>    *the_sender_slot;
    const char       *recipient_identity;
    CString          recipient_string;

    for(loop=0; loop<total_bids; loop++)
    {
        slot_base_class = the_bids[loop]->Get_Slot(SENDER);
        the_sender_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
        recipient_string = the_sender_slot->Get_Slot_Value();
        recipient_identity = recipient_string;
        Send_Message(recipient_identity, the_grant);
    };

    delete the_grant;
}

/*****************************************************************************
 *  Setup_Pickup_Message (private)
 *
 *  this method sets up a message to inform a part presentation agent that
 *  a part was successfully picked up.
 *
 *****************************************************************************/

void Assembling_Agent::Setup_Pickup_Message(
    Report_Frame *the_report, Boolean grasp_successful)
{
    Slot<Boolean>    grasp_slot(PART_WAS_GRASPED, grasp_successful);

    _pickup_message = Create_Report_Frame_Response(the_report,
                                                   _identity_of_PPA);
    _pickup_message->Add_Slot(&grasp_slot,
                              TCL_CLASSNAME_FROM_POINTER(_pickup_message));
}

/*****************************************************************************
 *  Request_Region_of_Assembly_Location (private)
 *
 *  given that a part was just grasped, this method requests the region
 *  of space of the current assembly operation from the shared space
 *  agent.
 *
 *****************************************************************************/

void Assembling_Agent::Request_Region_of_Assembly_Location(void)
{
    Grant_Frame      *current_task;
    Generic_Slot     *slot_base_class;
    Slot<CString>    *task_specification_slot;
    CString          task_specification, region_of_space, task_string;
```

```
    current_task = Get_Current_Task();
    slot_base_class = current_task->Get_Slot(TASK_SPECIFICATION);
    task_specification_slot = TCL_DYNAMIC_CAST(Slot<CString>,
                                                    slot_base_class);
    task_specification = task_specification_slot->Get_Slot_Value();

    region_of_space = Get_Assembly_Location(task_specification);
    task_string = current_task->get_name_of_instance();
    task_string += DELIMITER_STRING;
    task_string += REQUEST_REGION_STRING;

    Send_Request_Message_to_SSA(task_string, region_of_space);
}

/*************************************************************************
 *  Release_Region (private)
 *
 *  this method releases the region of space that was previously requested
 *  from the SSA in order to carry out an assembly operation.
 *
 *************************************************************************/

void Assembling_Agent::Release_Region(void)
{
    Grant_Frame      *current_task;
    Generic_Slot     *slot_base_class;
    Slot<CString>    *task_specification_slot;
    CString          task_specification, region_of_space, task_string;

    current_task = Get_Current_Task();
    slot_base_class = current_task->Get_Slot(TASK_SPECIFICATION);
    task_specification_slot = TCL_DYNAMIC_CAST(Slot<CString>,
                                                    slot_base_class);
    task_specification = task_specification_slot->Get_Slot_Value();

    region_of_space = Get_Assembly_Location(task_specification);
    task_string = current_task->get_name_of_instance();
    task_string += DELIMITER_STRING;
    task_string += RELEASE_REGION_STRING;

    Send_Release_Message_to_SSA(task_string, region_of_space);
}

/*************************************************************************
 *  Remove_Any_Bid_Messages_from_Queue (private)
 *
 *  this method removes any bid messages that were left in the
 *  message queue.
 *
 *************************************************************************/

void Assembling_Agent::Remove_Any_Bid_Messages_from_Queue(void)
{
    Message_Frame    *the_message = _the_message_queue.Pop_from_Front();

    if(!the_message)
        return;

    Basic_Queue<Message_Frame *>  other_messages;
    Bid_Frame                     *the_bid;

    do
    {
```

```
            the_bid = TCL_DYNAMIC_CAST(Bid_Frame, the_message);
            if(the_bid)
            {
                Send_Not_Granted_Task_Message(the_bid);
                delete the_bid;
            }
            else
                other_messages.Push_to_Front(the_message);

            the_message = _the_message_queue.Pop_from_Front();
        }
        while(the_message);

        the_message = other_messages.Pop_from_Front();
        while(the_message)
        {
            _the_message_queue.Push_to_Front(the_message);
            the_message = other_messages.Pop_from_Front();
        };
}

/*****************************************************************************
 *  Check_Message_Queue_and_Generate_Event_if_Necessary (private)
 *
 *  this method gets the first message in the message queue and generates
 *  the corresponding event if necessary.
 *
 *****************************************************************************/

void Assembling_Agent::
                Check_Message_Queue_and_Generate_Event_if_Necessary(void)
{
    Message_Frame    *the_message;

    the_message = _the_message_queue.Pop_from_Front();
    if(!the_message)
    {
        _checking_qualifications = FALSE;
        return;
    };

    Task_Announcement_Frame  *the_task;

    the_task = TCL_DYNAMIC_CAST(Task_Announcement_Frame, the_message);
    if(the_task)
    {
        the_simulator->Add_to_Event_List_with_Delay(0, this, TASK_LAUNCHED);
        _checking_qualifications = TRUE;
    };

    _the_message_queue.Push_to_Front(the_message);
}

/*****************************************************************************
 *  Get_Delimiters (private)
 *
 *  given the task specification of an assembly operation, this method
 *  retrieves the delimiters requested by the user.
 *
 *****************************************************************************/

void Assembling_Agent::Get_Delimiters(CString &task_specification,
    long &first_delimiter, long &second_delimiter, long &third_delimiter)
{
```

```
    long    index, total_delimiters;
    long    string_length = task_specification.length();
    char    *the_string = new char [string_length+1];

    strcpy(the_string, task_specification);
    total_delimiters=0;
    string_length = task_specification.length();
    for(index=0; index<=string_length; index++)
        if((the_string[index]==DELIMITER_CHAR) || (the_string[index]==0))
        {
            total_delimiters++;
            if((total_delimiters==1) && (&first_delimiter))
                first_delimiter=index;
            else if ((total_delimiters==2) && (&second_delimiter))
                second_delimiter=index;
            else if ((total_delimiters==3) && (&third_delimiter))
            {
                third_delimiter=index;
                break;
            };
        };

    delete [] the_string;
}

/*************************************************************************
 *  Get_Assembly_Operation (protected)
 *
 *  given the task specification of an assembly operation, this method
 *  returns the string corresponding to the specific assembly operation.
 *
 *************************************************************************/

CString Assembling_Agent::Get_Assembly_Operation(CString &task_specification)
{
    long    first_delimiter, string_length;
    CString assembly_operation;

    Get_Delimiters(task_specification, first_delimiter);

    string_length = first_delimiter+1;
    if(!string_length)
        return(assembly_operation);

    char    *operation = new char [string_length];
    long    index;

    for(index=0; index<string_length-1; index++)
        operation[index] = task_specification[index];

    operation[string_length-1] = 0;

    assembly_operation = operation;
    delete [] operation;
    return(assembly_operation);
}

/*************************************************************************
 *  Get_Part_Identity (protected)
 *
 *  given the task specification of an assembly operation, this method
 *  returns the string corresponding to the identity of the part to be
 *  assembled.
 *
```

```
 ***************************************************************************/

CString Assembling_Agent::Get_Part_Identity(CString &task_specification)
{
    long    first_delimiter, second_delimiter, string_length;
    CString part_string;

    Get_Delimiters(task_specification, first_delimiter, second_delimiter);

    string_length = second_delimiter - first_delimiter;
    if(!string_length)
        return(part_string);

    char    *part = new char [string_length];
    long    index;

    for(index=0; index<string_length-1; index++)
        part[index] = task_specification[index+first_delimiter+1];

    part[string_length-1] = 0;

    part_string = part;
    delete [] part;
    return(part_string);
}

/***************************************************************************
 *  Get_Assembly_Location (protected)
 *
 *  given the task specification of an assembly operation, this method
 *  returns the string corresponding to the location of where to assemble
 *  the part.
 *
 ***************************************************************************/

CString Assembling_Agent::Get_Assembly_Location(CString &task_specification)
{
    long    second_delimiter, third_delimiter, string_length;
    CString assembly_location;

    Get_Delimiters(task_specification, *((long *) NULL), second_delimiter,
                                                         third_delimiter);

    string_length = third_delimiter - second_delimiter;
    if(!string_length)
        return(assembly_location);

    char    *location = new char [string_length];
    long    index;

    for(index=0; index<string_length-1; index++)
        location[index] = task_specification[index+second_delimiter+1];

    location[string_length-1] = 0;

    assembly_location = location;
    delete [] location;
    return(assembly_location);
}

/***************************************************************************
 *  Get_Current_Task (protected)
 *
 *  this method simply returns the grant message corresponding to the
```

```
 *   current task that an assembling agent is doing.
 *
 ***************************************************************************/

Grant_Frame *Assembling_Agent::Get_Current_Task(void)
{
    Base_Frame               *frame_base_class;
    Model_of_Agent_Frame *model_of_self;
    Generic_Slot             *slot_base_class;
    Slot<CString>            *current_task;
    Grant_Frame              *the_task;

    frame_base_class = _the_knowledge_base.Get_Frame(MODEL_OF_SELF);
    model_of_self = TCL_DYNAMIC_CAST(Model_of_Agent_Frame, frame_base_class);
    slot_base_class = model_of_self->Get_Slot(CURRENT_TASK);
    current_task = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    frame_base_class = _the_knowledge_base.Get_Frame(
                                            current_task->Get_Slot_Value());
    the_task = TCL_DYNAMIC_CAST(Grant_Frame, frame_base_class);

    return(the_task);
}

/***************************************************************************
 *   Add_Can_Do_Capability (protected)
 *
 *   this method adds the given assembly operation to one of the operations
 *   that an ASA can do.
 *
 ***************************************************************************/

void Assembling_Agent::Add_Can_Do_Capability(CString &assembly_operation)
{
    Base_Frame               *frame_base_class;
    Model_of_Agent_Frame *model_of_self;

    frame_base_class = _the_knowledge_base.Get_Frame(MODEL_OF_SELF);
    model_of_self = TCL_DYNAMIC_CAST(Model_of_Agent_Frame, frame_base_class);

    CString  can_do_capability_label;

    can_do_capability_label = CAN_DO_STRING;
    can_do_capability_label += DELIMITER_STRING;
    can_do_capability_label += assembly_operation;

    Slot<Boolean>    can_do_slot(can_do_capability_label, TRUE);

    model_of_self->Add_Slot(&can_do_slot,
                            TCL_CLASSNAME_FROM_POINTER(model_of_self));
}

/***************************************************************************
 *   Add_Can_Grasp_Capability (protected)
 *
 *   this method adds the given part to one of the parts that an ASA can
 *   grasp.
 *
 ***************************************************************************/

void Assembling_Agent::Add_Can_Grasp_Capability(CString &the_part)
{
    Base_Frame               *frame_base_class;
    Model_of_Agent_Frame *model_of_self;
```

```
    frame_base_class = _the_knowledge_base.Get_Frame(MODEL_OF_SELF);
    model_of_self = TCL_DYNAMIC_CAST(Model_of_Agent_Frame, frame_base_class);

    CString can_grasp_capability_label;

    can_grasp_capability_label = CAN_GRASP_STRING;
    can_grasp_capability_label += DELIMITER_STRING;
    can_grasp_capability_label += the_part;

    Slot<Boolean>    can_grasp_slot(can_grasp_capability_label, TRUE);

    model_of_self->Add_Slot(&can_grasp_slot,
                            TCL_CLASSNAME_FROM_POINTER(model_of_self));
}

/*****************************************************************************
 *  Add_Can_Reach_Capability (protected)
 *
 *  this method adds the given assembly location to one of the locations
 *  that an ASA can reach.
 *
 *****************************************************************************/

void Assembling_Agent::Add_Can_Reach_Capability(CString &assembly_location)
{
    Base_Frame               *frame_base_class;
    Model_of_Agent_Frame *model_of_self;

    frame_base_class = _the_knowledge_base.Get_Frame(MODEL_OF_SELF);
    model_of_self = TCL_DYNAMIC_CAST(Model_of_Agent_Frame, frame_base_class);

    CString can_reach_capability_label;

    can_reach_capability_label = CAN_REACH_STRING;
    can_reach_capability_label += DELIMITER_STRING;
    can_reach_capability_label += assembly_location;

    Slot<Boolean>    can_reach_slot(can_reach_capability_label, TRUE);

    model_of_self->Add_Slot(&can_reach_slot,
                            TCL_CLASSNAME_FROM_POINTER(model_of_self));
}

/*****************************************************************************
 *  Add_Execution_Time_for_Capability (protected)
 *
 *  for the given assembly operation and part, this method sets the
 *  average execution time.
 *
 *****************************************************************************/

void Assembling_Agent::Add_Execution_Time_for_Capability(
    CString &assembly_operation, CString &the_part,
    double average_execution_time)
{
    Base_Frame               *frame_base_class;
    Model_of_Agent_Frame *model_of_self;

    frame_base_class = _the_knowledge_base.Get_Frame(MODEL_OF_SELF);
    model_of_self = TCL_DYNAMIC_CAST(Model_of_Agent_Frame, frame_base_class);

    CString execution_time_label;

    execution_time_label = assembly_operation;
```

```
    execution_time_label += DELIMITER_STRING;
    execution_time_label += the_part;
    execution_time_label += DELIMITER_STRING;
    execution_time_label += AVERAGE_EXECUTION_TIME_STRING;

    Slot<double>     time_slot(execution_time_label, average_execution_time);

    model_of_self->Add_Slot(&time_slot,
                                    TCL_CLASSNAME_FROM_POINTER(model_of_self));
}

/*****************************************************************************
 *  Setup_Knowledge_Base (protected)
 *
 *  this method sets up the basic knowledge of an assembling agent.
 *
 *****************************************************************************/

void Assembling_Agent::Setup_Knowledge_Base(void)
{
    State    startup_state(IDLE, IDLE_STRING);

    set_current_state(startup_state);

    Base_Frame               *frame_base_class;
    Model_of_Agent_Frame *model_of_self;
    Slot<CString>            current_task(CURRENT_TASK, NULL_STRING);

    frame_base_class = _the_knowledge_base.Get_Frame(MODEL_OF_SELF);
    model_of_self = TCL_DYNAMIC_CAST(Model_of_Agent_Frame, frame_base_class);
    model_of_self->Add_Slot(&current_task,
                                  TCL_CLASSNAME_FROM_POINTER(model_of_self));

    _checking_qualifications = FALSE;
}

/*****************************************************************************
 *  Assembling_Agent (constructor - protected)
 *
 *  this constructor initializes the assembling agent with the given
 *  name and the given name for its direct parent.  this is the primary
 *  agent constructor for derived ASAs.
 *
 *****************************************************************************/

Assembling_Agent::Assembling_Agent(
    const char *name_of_agent, const char *name_of_direct_parent) :
                                  Agent(name_of_agent, name_of_direct_parent)
{
    Setup_Knowledge_Base();
}

/*****************************************************************************
 *  Move_and_Grasp_Part (protected - virtual)
 *
 *  given the report frame sent by a part presentation agent, this method
 *  must be overridden by a specific assembling agent in order to move
 *  and grasp the part as delivered by this PPA.  this method returns
 *  TRUE if the grasp is successful; otherwise, it returns FALSE.  it
 *  also sets the time to grasp the part for the discrete event
 *  simulation.  the default method sets the grasp time to 1 unit and
 *  returns TRUE.
 *
 *****************************************************************************/
```

```
Boolean Assembling_Agent::Move_and_Grasp_Part(
    Report_Frame *the_report, double &time_to_grasp_part)
{
    time_to_grasp_part = 1.0;

    return(TRUE);
}

/*****************************************************************************
 *  Move_to_Assembly_Location (protected - virtual)
 *
 *  this method must be overridden by a specific assembling agent in
 *  order to move from its current location to the location where the
 *  assembly is going to take place.  this method also sets the time to
 *  move to the assembly location for the discrete event simulation.
 *  the default method simply sets the move time to 1 unit.
 *
 *****************************************************************************/

void Assembling_Agent::Move_to_Assembly_Location(double &time_to_move)
{
    time_to_move = 1.0;
}

/*****************************************************************************
 *  Assemble_the_Part (protected - virtual)
 *
 *  this method must be overridden by a specific assembling agent in
 *  order to assemble the part according to the requirements of an
 *  assembly operation.  this method returns TRUE if the assembly
 *  operation is successfully completed; otherwise, it returns FALSE.  it
 *  also sets the time to assemble the part for the discrete event
 *  simulation.  the default method sets the part assembling time to 6.0
 *  units and returns TRUE.
 *
 *****************************************************************************/

Boolean Assembling_Agent::Assemble_the_Part(double &time_to_assemble)
{
    time_to_assemble = 6.0;

    return(TRUE);
}

/*****************************************************************************
 *  Discard_the_Part (protected - virtual)
 *
 *  this method should be overridden by a specific assembling agent in
 *  order to discard the part when an assembly operation was not
 *  successful.  the default method simply sets the time to discard
 *  the part to 1 unit.
 *
 *****************************************************************************/

void Assembling_Agent::Discard_the_Part(double &time_to_discard_part)
{
    time_to_discard_part = 1.0;
}

/*****************************************************************************
 *  Process_the_Bids (protected)
 *
 *  this method processes the bids that were received for the announcement
```

```
 *   of a part fetching task.  it returns TRUE, if bids were submitted and
 *   the task is granted to a particular part presentation agent; otherwise,
 *   it returns FALSE.
 *
 ****************************************************************************/

void Assembling_Agent::Process_the_Bids(void)
{
    Basic_List<Bid_Frame *>  the_bids;
    double                   time_to_process=0.05;

    Get_Bids_for_Task(_current_fetch_part_task, the_bids);
    if(!the_bids.number_of_elements())
    {
        Retransmit_Task_Announcement();
        the_simulator->Add_to_Event_List_with_Delay(time_to_process, this,
                                                    BIDS_NOT_MADE);
        return;
    };

    Select_Best_Bid_and_Grant_Task(the_bids);

    size_t   loop;

    for(loop=0; loop<the_bids.number_of_elements(); loop++)
        delete the_bids[loop];

    the_simulator->Add_to_Event_List_with_Delay(time_to_process, this,
                                                BIDS_WERE_MADE);
}

/****************************************************************************
 *   Execute_Check_if_Qualified (protected)
 *
 *   this method checks whether an ASA is qualified for any announced
 *   assembly tasks and also generates its corresponding events.
 *
 ****************************************************************************/

void Assembling_Agent::Execute_Check_if_Qualified(State &current_state)
{
    double   time_to_send=0.02;

    Get_First_Potential_Task(_potential_task);

    if(_potential_task)
        the_simulator->Add_to_Event_List_with_Delay(time_to_send, this,
                                                    BID_FOR_TASK);
    else
        Check_Message_Queue_and_Generate_Event_if_Necessary();
}

/****************************************************************************
 *   Execute_Submit_Bid_for_Task (protected)
 *
 *   this method submits a bid for an assembly task that an assembling
 *   agent is qualified to perform.
 *
 ****************************************************************************/

void Assembling_Agent::Execute_Submit_Bid_for_Task(State &current_state)
{
    current_state.set_state(WAITING_FOR_GRANT, WAITING_FOR_GRANT_STRING);
    set_current_state(current_state);
```

```
    _checking_qualifications = FALSE;

    Generic_Slot     *slot_base_class;
    Slot<CString>    *message_sender_slot;
    CString          task_identity, recipient_identity;
    Multi_Slot       bid_information;

    task_identity = _potential_task->get_name_of_instance();
    slot_base_class = _potential_task->Get_Slot(SENDER);
    message_sender_slot = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    recipient_identity = message_sender_slot->Get_Slot_Value();
    Setup_Bid_Information(_potential_task, bid_information);

    char             *sender_identity = get_agent_identity();
    Bid_Frame        *the_bid_frame;

    the_bid_frame = new Bid_Frame(task_identity, sender_identity,
                                  recipient_identity, &bid_information);
    Send_Message(recipient_identity, the_bid_frame);

    delete [] sender_identity;
    _the_knowledge_base.Add_Frame(the_bid_frame);
    delete the_bid_frame;
    delete _potential_task;
}

/*****************************************************************************
 *  Execute_Task_Not_Granted (protected)
 *
 *  this method makes the transition back to the idle state, if an
 *  ASA was not granted an assembly task for which it bid.
 *
 *****************************************************************************/

void Assembling_Agent::Execute_Task_Not_Granted(State &current_state)
{
    Grant_Frame  *the_grant;

    Get_the_Grant_Message(the_grant);

    current_state.set_state(IDLE, IDLE_STRING);
    set_current_state(current_state);
    Register_Trace_Message(NOT_GRANTED_TASK_STRING);
}

/*****************************************************************************
 *  Execute_Request_Part (protected)
 *
 *  this method requests the part needed to carry out an assembly
 *  operation from the part presentation agents.
 *
 *****************************************************************************/

void Assembling_Agent::Execute_Request_Part(State &current_state)
{
    Grant_Frame      *the_grant;

    if(!Get_the_Grant_Message(the_grant))
        return;

    Switch_to_Busy();

    Base_Frame               *frame_base_class;
    Model_of_Agent_Frame *model_of_self;
```

```
    Generic_Slot              *slot_base_class;
    Slot<CString>             *current_task;

    frame_base_class = _the_knowledge_base.Get_Frame(MODEL_OF_SELF);
    model_of_self = TCL_DYNAMIC_CAST(Model_of_Agent_Frame, frame_base_class);
    slot_base_class = model_of_self->Get_Slot(CURRENT_TASK);
    current_task = TCL_DYNAMIC_CAST(Slot<CString>, slot_base_class);
    current_task->Set_Slot_Value(the_grant->get_name_of_instance());

    _the_knowledge_base.Remove_Frame(the_grant->get_name_of_instance());
    _the_knowledge_base.Add_Frame(the_grant);

    Request_Part(the_grant);
    delete the_grant;

    current_state.set_state(WAITING_FOR_BIDS, WAITING_FOR_BIDS_STRING);
    set_current_state(current_state);
}

/*****************************************************************************
 *  Execute_Store_Single_Bid (protected)
 *
 *  this method simply notes that a single bid was received by an
 *  ASA.
 *
 *****************************************************************************/

void Assembling_Agent::Execute_Store_Single_Bid(State &current_state)
{
    Register_Trace_Message(SINGLE_BID_MESSAGE_RECEIVED);
}

/*****************************************************************************
 *  Execute_Check_Bids_for_Task (protected)
 *
 *  this method checks whether bids were received for a previously
 *  announced part fetching task and responds accordingly.
 *
 *****************************************************************************/

void Assembling_Agent::Execute_Check_Bids_for_Task(State &current_state)
{
    current_state.set_state(BID_MADE_FOR_PART_REQUEST,
                                    BIDS_MADE_FOR_FETCH_PART_STRING);
    set_current_state(current_state);

    Process_the_Bids();
}

/*****************************************************************************
 *  Execute_Bids_Not_Made_Transition (protected)
 *
 *  this method makes the transition back to the waiting state when
 *  no bids were received for a part fetching task.
 *
 *****************************************************************************/

void Assembling_Agent::Execute_Bids_Not_Made_Transition(
    State &current_state)
{
    current_state.set_state(WAITING_FOR_BIDS, WAITING_FOR_BIDS_STRING);
    set_current_state(current_state);

    Register_Trace_Message(WAITING_FOR_BIDS_STRING);
```

```
}

/*****************************************************************************
 *   Execute_Grant_Part_Task (protected)
 *
 *   this method makes the transition to the waiting for part state for
 *   an ASA.
 *
 *****************************************************************************/

void Assembling_Agent::Execute_Grant_Part_Task(State &current_state)
{
    current_state.set_state(WAITING_FOR_PART, WAITING_FOR_PART_STRING);
    set_current_state(current_state);

    Register_Trace_Message(WAITING_FOR_PART_STRING);
}

/*****************************************************************************
 *   Execute_Grasp_Part (protected)
 *
 *   this method executes the grasp part transition in the CSM of an ASA.
 *
 *****************************************************************************/

void Assembling_Agent::Execute_Grasp_Part(State &current_state)
{
    Report_Frame     *the_report;
    double           time_to_grasp_part;
    Boolean          grasp_successful;

    if(!Get_the_Report_Message(the_report))
        return;

    grasp_successful = Move_and_Grasp_Part(the_report, time_to_grasp_part);
    if(grasp_successful)
        the_simulator->Add_to_Event_List_with_Delay(time_to_grasp_part,
                                            this, PART_GRASPED_SUCCESSFULLY);
    else
        the_simulator->Add_to_Event_List_with_Delay(time_to_grasp_part,
                                        this, PART_NOT_GRASPED_SUCCESSFULLY);

    Generic_Slot     *slot_base_class;
    Slot<Boolean>    *pickup_message_slot;

    slot_base_class = the_report->Get_Slot(PICKUP_MESSAGE_NEEDED);
    pickup_message_slot = TCL_DYNAMIC_CAST(Slot<Boolean>, slot_base_class);
    if(pickup_message_slot->Get_Slot_Value())
        Setup_Pickup_Message(the_report, grasp_successful);
    else
        _pickup_message = NULL;

    delete the_report;

    current_state.set_state(PART_GRASPED, PART_GRASPED_STRING);
    set_current_state(current_state);
}

/*****************************************************************************
 *   Execute_Part_Grasped_Transition (protected)
 *
 *   after having successfully grasped the part, this method makes the
 *   transition to the waiting for free region state of an ASA.
 *
```

```
 ***************************************************************************/

void Assembling_Agent::Execute_Part_Grasped_Transition(State &current_state)
{
    if(_pickup_message)
    {
        Send_Message(_identity_of_PPA, _pickup_message);
        delete _pickup_message;
    }

    Request_Region_of_Assembly_Location();

    current_state.set_state(WAITING_FOR_FREE_REGION,
                                        TRY_PART_ACQUISITION_AGAIN_STRING);
    set_current_state(current_state);

    Register_Trace_Message(WAITING_FOR_FREE_REGION_STRING);
}

/***************************************************************************
 *   Execute_Part_Not_Grasped_Transition (protected)
 *
 *   if the part was not successfully grasped, this method makes the
 *   transition to the try part acquistion again state of an ASA.
 *
 ***************************************************************************/

void Assembling_Agent::Execute_Part_Not_Grasped_Transition(
    State &current_state)
{
    if(_pickup_message)
    {
        Send_Message(_identity_of_PPA, _pickup_message);
        delete _pickup_message;
    }

    current_state.set_state(TRY_PART_ACQUISITION_AGAIN,
                                        TRY_PART_ACQUISITION_AGAIN_STRING);
    set_current_state(current_state);

    Register_Trace_Message(TRY_PART_ACQUISITION_AGAIN_STRING);
}

/***************************************************************************
 *   Execute_Assemble_Part (protected)
 *
 *   this method executes the move to assembly location and proceed to
 *   assemble the part transition of the CSM of an ASA.
 *
 ***************************************************************************/

void Assembling_Agent::Execute_Assemble_Part(State &current_state)
{
    Report_Frame     *the_report;
    double           time_to_move;

    if(!Get_the_Report_Message(the_report))
        return;

    delete the_report;

    Register_Trace_Message(START_MOVE_ASSEMBLY_LOC_STRING);

    Move_to_Assembly_Location(time_to_move);
```

```
    Boolean assembly_successful;
    double  time_to_assemble;

    assembly_successful = Assemble_the_Part(time_to_assemble);
    if(assembly_successful)
        the_simulator->Add_to_Event_List_with_Delay(
                time_to_move+time_to_assemble, this, PART_WAS_ASSEMBLED);
    else
        the_simulator->Add_to_Event_List_with_Delay(
                time_to_move+time_to_assemble, this, PART_WAS_NOT_ASSEMBLED);

    current_state.set_state(PART_SUCCESSFULLY_ASSEMBLED,
                                        PART_SUCCESSFULLY_ASSEMBLED_STRING);
    set_current_state(current_state);
}

/*****************************************************************************
 *  Execute_Successful_Assembly_Transition (protected)
 *
 *  this method executes the transition of an ASA when the part assembling
 *  was successful.
 *
 *****************************************************************************/

void Assembling_Agent::Execute_Successful_Assembly_Transition(
    State &current_state)
{
    Register_Trace_Message(END_PART_ASSEMBLY_STRING);

    Release_Region();

    Grant_Frame       *the_task;
    Report_Frame      *the_report;
    Slot<CString>     type_of_report(TYPE_OF_REPORT, TASK_TERMINATED_REPORT);
    CString           recipient_identity;

    the_task = Get_Current_Task();

    the_report = Create_Report_Frame_Response(the_task, recipient_identity);
    the_report->Add_Slot(&type_of_report,
                                    TCL_CLASSNAME_FROM_POINTER(the_report));

    Send_Message(recipient_identity, the_report);

    delete the_report;
    _the_knowledge_base.Remove_Frame(the_task->get_name_of_instance());

    current_state.set_state(IDLE, "idle");
    set_current_state(current_state);
    Switch_to_Idle();

    Remove_Any_Bid_Messages_from_Queue();
    Check_Message_Queue_and_Generate_Event_if_Necessary();
}

/*****************************************************************************
 *  Execute_Unsuccessful_Assembly_Transition (protected)
 *
 *  this method executes the transition of an ASA when the part assembling
 *  was unsuccessful.
 *
 *****************************************************************************/
```

```
void Assembling_Agent::Execute_Unsuccessful_Assembly_Transition(
    State &current_state)
{
    double   time_to_discard_part;

    Register_Trace_Message(START_DISCARD_PART_STRING);

    Discard_the_Part(time_to_discard_part);

    current_state.set_state(TRY_PART_ACQUISITION_AGAIN,
                                        TRY_PART_ACQUISITION_AGAIN_STRING);
    set_current_state(current_state);

    Release_Region();

    the_simulator->Add_to_Event_List_with_Delay(time_to_discard_part, this,
                                                              TRY_AGAIN);
}

/*****************************************************************************
 *  Execute_Request_Part_Again (protected)
 *
 *  if the part was not successfully grasped or the part assembling
 *  was unsuccessful, this method requests the part from the PPAs again.
 *
 *****************************************************************************/

void Assembling_Agent::Execute_Request_Part_Again(State &current_state)
{
    Retransmit_Task_Announcement();

    current_state.set_state(WAITING_FOR_BIDS, WAITING_FOR_BIDS_STRING);
    set_current_state(current_state);

    Register_Trace_Message(WAITING_FOR_BIDS_STRING);
}

/*****************************************************************************
 *  Receive_Message (protected - override)
 *
 *  this method makes a copy of the given message and responds to it
 *  differently depending upon its type and the current state of an ASA.
 *
 *****************************************************************************/

void Assembling_Agent::Receive_Message(
    const Message_Frame *message_to_receive)
{
    Base_Frame       *frame_base_class;
    Message_Frame    *copied_message;
    Grant_Frame      *the_grant;
    Report_Frame     *the_report;
    Bid_Frame        *the_bid;
    double           message_time, message_delay = (*_the_message_delay)();

    frame_base_class = message_to_receive->Clone();
    copied_message = TCL_DYNAMIC_CAST(Message_Frame, frame_base_class);
    message_time = copied_message->Get_Message_Time() + message_delay;
    the_grant = TCL_DYNAMIC_CAST(Grant_Frame, copied_message);
    the_report = TCL_DYNAMIC_CAST(Report_Frame, copied_message);
    the_bid = TCL_DYNAMIC_CAST(Bid_Frame, copied_message);

    if(the_report || the_grant)
        _the_message_queue.Push_to_Front(copied_message);
```

```
        else
            _the_message_queue.Push_to_Back(copied_message);

        State    current_state = get_current_state();

        switch (current_state.get_state_id())
        {
            case IDLE:
                if(!_checking_qualifications)
                {
                    the_simulator->Add_to_Event_List_at_Time(message_time, this,
                                                                TASK_LAUNCHED);
                    _checking_qualifications = TRUE;
                };
                break;
            case WAITING_FOR_GRANT:
                if(the_grant)
                {
                    if(Received_the_Grant(the_grant))
                        the_simulator->Add_to_Event_List_at_Time(message_time,
                                                    this, TASK_GRANTED);
                    else
                        the_simulator->Add_to_Event_List_at_Time(message_time,
                                                    this, TASK_NOT_GRANTED);
                };
                break;
            case WAITING_FOR_BIDS:
                if(the_bid)
                    the_simulator->Add_to_Event_List_at_Time(message_time,
                                                this, SINGLE_BID_RECEIVED);
                break;
            case WAITING_FOR_PART:
                if(the_report)
                    the_simulator->Add_to_Event_List_at_Time(message_time,
                                                this, PART_POSE_RECEIVED);
                break;
            case WAITING_FOR_FREE_REGION:
                if(the_report)
                    the_simulator->Add_to_Event_List_at_Time(message_time,
                                                this, REGION_REQUEST_GRANTED);
                break;
        };
}

/*****************************************************************************
 *   Execute (public - override)
 *
 *   this method defines the execution of an assembling agent.
 *
 *****************************************************************************/

void Assembling_Agent::Execute(int type_of_event)
{
    State    current_state = get_current_state();

    switch (current_state.get_state_id())
    {
        case IDLE:
            if(type_of_event==TASK_LAUNCHED)
                Execute_Check_if_Qualified(current_state);
            else if(type_of_event==BID_FOR_TASK)
                Execute_Submit_Bid_for_Task(current_state);
            break;
        case WAITING_FOR_GRANT:
```

```
            if (type_of_event==TASK_NOT_GRANTED)
                Execute_Task_Not_Granted(current_state);
            else if(type_of_event==TASK_GRANTED)
                Execute_Request_Part(current_state);
            break;
        case WAITING_FOR_BIDS:
            if(type_of_event==SINGLE_BID_RECEIVED)
                Execute_Store_Single_Bid(current_state);
            else if(type_of_event==REQUEST_PART_TIMES_OUT)
                Execute_Check_Bids_for_Task(current_state);
            break;
        case BID_MADE_FOR_PART_REQUEST:
            if(type_of_event==BIDS_NOT_MADE)
                Execute_Bids_Not_Made_Transition(current_state);
            else if(type_of_event==BIDS_WERE_MADE)
                Execute_Grant_Part_Task(current_state);
            break;
        case WAITING_FOR_PART:
            if(type_of_event==PART_POSE_RECEIVED)
                Execute_Grasp_Part(current_state);
            break;
        case PART_GRASPED:
            if(type_of_event==PART_GRASPED_SUCCESSFULLY)
                Execute_Part_Grasped_Transition(current_state);
            else if(type_of_event==PART_NOT_GRASPED_SUCCESSFULLY)
                Execute_Part_Not_Grasped_Transition(current_state);
            break;
        case WAITING_FOR_FREE_REGION:
            if(type_of_event==REGION_REQUEST_GRANTED)
                Execute_Assemble_Part(current_state);
            break;
        case PART_SUCCESSFULLY_ASSEMBLED:
            if(type_of_event==PART_WAS_ASSEMBLED)
                Execute_Successful_Assembly_Transition(current_state);
            else if(type_of_event==PART_WAS_NOT_ASSEMBLED)
                Execute_Unsuccessful_Assembly_Transition(current_state);
            break;
        case TRY_PART_ACQUISITION_AGAIN:
            if(type_of_event==TRY_AGAIN)
                Execute_Request_Part_Again(current_state);
            break;
    }
}

/*****************************************************************************
 *  Assembling_Agent (default constructor - public)
 *
 *  the default constructor is needed for run-time type identification and
 *  is not meant to be used.
 *
 *****************************************************************************/

Assembling_Agent::Assembling_Agent(void) : Agent()
{
}

/*****************************************************************************
 *  Assembling_Agent (constructor - public)
 *
 *  the constructor initializes the ASA with the name specified
 *  by the user.
 *
 *****************************************************************************/
```

```
Assembling_Agent::Assembling_Agent(const char *name_of_agent) :
                    Agent(name_of_agent, TCL_CLASSNAME_FROM_POINTER(this))
{
    Setup_Knowledge_Base();
}

/*****************************************************************************
 *   ~Assembling_Agent (destructor)
 *
 *   this destructor has nothing to free.
 *
 *****************************************************************************/

Assembling_Agent::~Assembling_Agent(void)
{
}
```

Bibliography

[AARIA, 1997] Autonomous Agents for Rock Island Arsenal, "http://www.aaria.uc.edu/", Web Site, 1997.

[Aarsten and Brugali, 1997] Amund Aarsten and Davide Brugali, "From Object-Orientation to Agent-Orientation: Common Issues in the Development of System Architectures", IEEE International Conference on Robotics and Automation, *Workshop on Object Oriented Methods for Distributed Control Architectures*, Giuseppe Menga and Maria Gini, editors, April 1997.

[Abu-Hamdan and El-Gizawy, 1992] M. G. Abu-Hamdan and A. S. El-Gizawy, "An Error Diagnosis Expert System for Flexible Assembly Systems", *Information Control Problems in Manufacturing Technology*, IFAC-INCOM, 1992, pp. 451-456.

[AgentWeb, 1997] UMBC AgentWeb Resources, "http://www.cs.umbc.edu/agents/", Web Site, 1997.

[AIMS, 1997] Agile Infrastructure for Manufacturing Systems, "http://vet.parl.com/", Web Site, 1997.

[Akcadag et al., 1992] C. Akcadag, A. Shirkhodaie, and A. H. Soni, "Computer Control of a Flexible Robotic Cell", *4th Conference on Flexible Assembly Systems*, The Design Engineering Division, ASME 1992, Volume 48, pp. 77-82.

[Akella and Krogh, 1987] Ram Akella and Bruce Krogh, "Hierarchical Control Structures for Multi-Cell Flexible Assembly System Coordination", *IEEE International Conference Robotics and Automation*, 1987, pp. 295-301.

[Alander et al., 1990] Jarmo T. Alander, Matti Frisk, Juha Tuominen, and Merja Vuolteenaho, "An Outline of a Modular and Multiarm Assembly Cell with Object-Oriented Control", *16th Annual Conference of IEEE Industrial Electronics Society*, November 1990, Volume 1, pp. 592-597.

[Albayrak and Krallman, 1995] S. Albayrak, and H. Krallman, "Distributed Artificial Intelligence in Manufacturing Control", *Artificial Intelligence in Industrial Decision Making, Control and Automation*, Spyros G. Tzafestas and Henk B. Verbruggen, editors, 1995, Kluwer Academic Publishers, pp. 247-294.

[Albus et al., 1987] James S. Albus, Harry G. McCain, and Ronald Lumia, *NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)*, National Bureau of Standards Technical Note 1235, 1987.

[Al-Jaar and Desrochers, 1990] Robert Y. Al-Jaar and Alan A. Desrochers, "Performance Evaluation of Automated Manufacturing Systems Using Generalized Stochastic Petri Nets", *IEEE Transactions on Robotics and Automation*, December 1990, Volume 6, Number 6, pp. 621-639.

[Amamiya et al., 1993] Makoto Amamiya, Shigeru Kusakabe, and Shigeto Aramaki, "A Multi-Agent System Description Language *New Valid* and Its Application to Robot Control", *IEEE 2nd International Workshop on Emerging Technologies and Factory Automation*, September 1993, pp. 210-217.

[Ames et al., 1995] Arlo L. Ames, Terri L. Calton, Rondall E. Jones, Stephen G. Kaufman, Cathy A. Laguna, and Randall H. Wilson, "Lessons Learned from a Second Generation Assembly Planning System", *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, August 1995, pp. 41-47.

[Amori, 1988] R. D. Amori, "A Multiple Cooperating Intelligent Agents Project Status Report", *Computers in Engineering*, July 1988, Volume 2, pp. 317-323.

[Andrews and McCarroll, 1988] Phillip P. Andrews and Bruce McCarroll, "Integration, Robots and CIM", *International Encyclopedia of Robotics: Applications and Automation*, Richard D. Dorf, Editor-in-Chief, 1988, Volume 2, pp. 691-703.

[Archibald, 1995] Colin C. Archibald, *A Computational Model for Skills-Oriented Robot Programming*, Doctoral Dissertation, University of Ottawa, 1995.

[Archibald and Petriu, 1993] Colin Archibald and Emil Petriu, "Functional Abstraction for High-Level Online Robot Programming", *DND Workshop on Advanced Technologies in Knowledge Based Systems and Robotics*, November 1993, pp. 309-314.

[Arkin and Ali, 1994] Ronald C. Arkin and Khaled S. Ali, "Integration of Reactive and Telerobotic Control in Multi-agent Robotic Systems", *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, David Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors, 1994, pp. 473-478.

[Arkin and Hobbs, 1993] Ronald C. Arkin and J. David Hobbs, "Dimensions of Communication and Social Organization in Multi-agent Robotic Systems", *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, Jean-Arcady Meyer, Herbert L. Roitblat, and Stewart W. Wilson, editors, 1993, pp. 486-493.

[Asama et al., 1992] Hajime Asama, Yoshiki Ishida, Koichi Ozaki, Maki K. Habib, Akihiro Matsumoto, Hayato Kaetsu, and Isao Endo, "A Communication System between Multiple Robotic Agents", *Proceedings of the Japan-USA Symposium on Flexible Automation*, July 1992, Volume 1, pp. 647-654.

[Atabakhsh and Chan, 1992] H. Atabaksh and A. W. Chan, "Discrete Event Simulation using Object-Oriented Programming: Application to Manufacturing", *National Research Council of Canada*, 1992, NRC No. 33188.

[Aylett et al., 1995] R. S. Aylett, A. M. Coddington, R. A. Ghanea-Hercock, and D. P. Barnes, "Heterogeneous Agents for Multi-Robot Cooperation", *The Design and Development of Autonomous Agents*, IEE Computing and Control Division, November 1995, Digest Number 95/211, pp. 3/1 to 3/7.

[Bagchi et al., 1996] Sugato Bagchi, Gautam Biswas, and Kazuhiko Kawamura, "Interactive task planning under uncertainty and goal changes", *Robotics and Autonomous Systems*, 1996, Volume 18, pp. 157-167.

[Bagherzadeh et al., 1992] A. Bagherzadeh, A. K. M. Mostafa Kamal, and S. J. Steiner, "The Development of an Improved Control Methodology for a Robotics Assembly Cell using A.I.-Based Sensory Systems", *Third International Conference on Competitive Performance through Advanced Technology*, July 1992.

[Baker, 1992] Albert D. Baker, "Case Study Results with the Market-Driven Contract Net Production Planning and Control System", *Conference Proceedings of AUTOFACT*, November 1992, pp. 31-17 to 31-35.

[Balch and Arkin, 1994] Tucker Balch and Ronald C. Arkin, "Communication in Reactive Multiagent Robotic Systems", *Autonomous Robots*, 1994, Volume 1, Number 1, pp. 27-52.

[Banks and Carson, 1984] Jerry Banks and John S. Carson II, *Discrete-Event System Simulation*, Prentice-Hall,

1984.

[Barata and Steiger-Garçao, 1994] Manuel M. Barata and A. Steiger-Garçao, "Transputer-Based Multi-Agent Monitoring System in NC", *Robotics & Computer-Integrated Manufacturing*, 1994, Volume 11, Number 4, pp. 319-325.

[Barbuceanu and Fox, 1995] Mihai Barbuceanu and Mark S. Fox, "The Architecture of an Agent Building Shell", *Intelligent Agents II*, Agent Theories, Architectures, and Languages, M. Wooldridge, J. P. Muller, and M. Tambe, editors, August 1995, pp. 235-250.

[Barr et al., 1982] Avron Barr, Paul R. Cohen, and Edward A. Feigenbaum, editors, *The Handbook of Artificial Intelligence*, Heuristech Press, 1982, Volumes 1, 2, 3.

[Basran and Petriu, 1993] Jagdeep S. Basran and Emil M. Petriu, "Paradigms of Intelligent Robotics", *DND Workshop on Advanced Technologies in Knowledge Based Systems and Robotics*, November 1993, pp. 293-300.

[Basran and Petriu, 1997] Jagdeep S. Basran and Emil M. Petriu, "Translation of Task-Level Instructions to Sensing/Actuation Skills by a Robotic Assembling Agent", *Proceedings of IMTC/97 IEEE Instrumentation and Measurement Technology Conference*, May 1997, pp. 593-598.

[Basran et al., 1997a] Jagdeep S. Basran, Emil M. Petriu, and Dorina C. Petriu, "Agent-based Robotic Assembly Cell implemented with Object-based Technology", *Proceedings of the 12th International Conference on Computers and Their Applications*, March 1997, pp. 271-275.

[Basran et al., 1997b] Jagdeep S. Basran, Emil M. Petriu, and Dorina C. Petriu, "Flexible Agent-based Robotic Assembly Cell", *Proceedings of the IEEE International Conference on Robotics and Automation*, April 1997, pp. 3461-3466.

[Basran et al., 1997c] Jagdeep S. Basran, Emil M. Petriu, and Dorina C. Petriu, "Object-Oriented Implementation of an Assembling Agent", *ISCA International Journal of Computers and Their Applications*, accepted for publication in the December 1997 issue.

[Beckers et al., 1994] R. Beckers, O. E. Holland, and J. L. Deneubourg, "From Local Actions to Global Tasks: Stigmergy and Collective Robotics", *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, Rodney A. Brooks and Pattie Maes, editors, 1994, pp. 181-189.

[Bekey, 1996] George A. Bekey, "Trends in Robotics", *Communications of the ACM*, February 1996, Volume 39, Number 2, p. 62.

[Benameur et al., 1991] N. Benameur, N. Akhamlich, Kh. Ouirachi, and M. Bourton, "Hierarchical programming of assembly tasks", *6th Mediterranean Electrotehnical Conference*, May 1991, Volume II, pp. 923-926.

[Ben-Arieh et al., 1988] David H. Ben-Arieh, Colin L. Moodie, and Chi-Chung Chu, "Control Methodology for FMS", *IEEE Journal of Robotics and Automation*, February 1988, Volume 4, Number 1, pp. 53-59.

[Benjamin et al., 1995] Perakath C. Benjamin, Christopher P. Menzel, Richard J. Mayer, and Natarajan Padmanaban, "Toward a method for acquiring CIM ontologies", *International Journal of Computer Integrated Manufacturing*, 1995, Volume 8, Number 3, pp. 225-234.

[Berge-Cherfaoui and Vachon, 1994] Véronique Berge-Cherfaoui and Bertrand Vachon, "Dynamic configuration of mobile robot perceptual system", *Proceedings of the IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, October 1994, pp. 707-714.

[Blume and Jakob, 1986] Christian Blume, and Wilfried Jakob, *Programming Languages for Industrial Robots*, Springer-Verlag, 1986.

[Bohner, 1994] Petra Bohner, "A Multi-agent Approach to Distributed Control for Task-Level Programs for Cooperating Manipulators", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 1994, pp. 1088-1094..

[Bohner, 1995] Petra Bohner, "A multi-agent approach with distributed fuzzy logic control", *Computers in Industry*, 1995, Volume 26, pp. 219-227.

[Bonasso et al., 1995] R. Peter Bonasso, David Kortenkamp, David P. Miller, and Marc Slack, "Experiences with an Architecture for Intelligent, Reactive Agents", *Intelligent Agents II*, Agent Theories, Architectures, and Languages, M. Wooldridge, J. P. Muller, and M. Tambe, editors, August 1995, pp. 187-202.

[Bond and Gasser, 1988] Alan H. Bond and Les Gasser, editors, *Readings in Distributed Artificial Intelligence*, Morgan Kaufmann Publishers, 1988.

[Boneschanscher, 1990] N. Boneschanscher, "Task Assignment for a Small Batch Flexible Assembly Cell Incorporating Multiple Robots", *Annual Conference of IEEE Industrial Electronics Society*, November 1990, Volume I, pp. 746-750.

[Bonner and Shin, 1982] Susan Bonner and Kang G. Shin, "A Comparative Study of Robot Languages", *IEEE Computer*, December 1982, Volume 15, pp. 82-96.

[Booch, 1986] Grady Booch, "Object-Oriented Development", *IEEE Transactions on Software Engineering*, 1986, Volume 12, Number 2, pp. 211-221.

[Booch, 1994] Grady Booch, *Object-Oriented Analysis and Design with Applications*, Addison-Wesley Publishing Company, 1994.

[Booth, 1996] Rupert Booth, "Agile Manufacturing", *Engineering Management Journal*, 1996, Volume 6, Number 2, pp. 105-112.

[Borchelt and Alptekin, 1994] R. D. Borchelt and S. Alptekin, "Error recovery in intelligent robotic workcells", *International Journal of Production Research*, 1994, Volume 32, Number 1, pp. 65-73.

[Boucher et al., 1989] Thomas O. Boucher, Mohsen A. Jafari, and Glenn A. Meredith, "Petri Net Control of an Automated Manufacturing Cell", *Computers and Industrial Engineering*, 1989, Volume 17, Numbers 1-4, pp. 459-463.

[Bozinovski, 1994] Steve Bozinovski, "Parallel Programming for Mobile Robot Control: Agent-Based Approach", *Proceedings of the 14th International Conference on Distributed Computing Systems*, June 1994, pp. 202-208.

[Brachman and Levesque, 1985] R. J. Brachman and H. J. Levesque, editors, *Readings in Knowledge Representation*, Morgan Kaufmann Publishers, 1985.

[Brady, 1985] Michael Brady, "Artificial Intelligence and Robotics", *Artificial Intelligence*, 1985. Volume 26, pp. 79-121.

[Bratko, 1988] Ivan Bratko, "AI Tools and Techniques for Manufacturing Systems", *Robotics & Computer-Integrated Manufacturing*, 1988. Volume 4, Numbers 1-2, pp. 27-31.

[Brooks, 1986a] Rodney A. Brooks, "A Robust Layered Control System for a Mobile Robot", *IEEE Journal of Robotics and Automation*, March 1986, Volume 2, Number 1, pp. 14-23.

[Brooks, 1986b] Rodney A. Brooks, "Achieving Artificial Intelligence through Building Robots", Massachusetts Institute of Technology, *AI Memo 899*, May 1986.

[Brooks, 1990] Rodney A. Brooks, "Elephants Don't Play Chess", *Robotics and Autonomous Systems*, 1990, Volume 6, pp. 3-15.

[Brooks, 1991a] Rodney A. Brooks, "Intelligence without Representation", *Artificial Intelligence*, 1991. Volume 47, Numbers. 1-3, pp. 139-160.

[Brooks, 1991b] Rodney A. Brooks, "Intelligence without Reason", Massachusetts Institute of Technology, *AI Memo 1293*, April 1991.

[Brown and Jennings, 1995] Russel G. Brown and James S. Jennings, "A Pusher/Steerer Model for Strongly Cooperative Mobile Robot Manipulation", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, August 1995, Volume 3, pp. 562-568.

[Bruyninckx et al., 1995] H. Bruyninckx, S. Dutré, and J. De Schutter, "Peg-on-Hole: A Model Based Solution to Peg and Hole Alignment", *IEEE International Conference on Robotics and Automation*, May 1995, Volume 2, pp. 1919-1924.

[Burgess, 1994] Thomas F. Burgess, "Making the Leap to Agility: Defining and Achieving Agile Manufacturing through Business Process Redesign and Business Network Redesign", *International Journal of Operations & Production Management*, 1994, Volume 14, Number 11, pp. 23-34.

[Caloud et al., 1990] Philippe Caloud, Wonyun Choi, Jean-Claude Latombe, Claude Le Pape, and Mark Yim, "Indoor Automation with Many Mobile Robots", *IEEE International Workshop on Intelligent Robots and Systems*, July 1990, Volume 1, pp. 67-72.

[Camarinha-Matos et al., 1995] L. M. Camarinha-Matos, H. Pinheiro-Pita, R. Rabelo, and J. Barata, "Towards a taxonomy of CIM activities", *International Journal of Computer Integrated Manufacturing*, 1995, Volume 8, Number 3, pp. 160-176.

[Cao and Sanderson, 1995] Tiehua Cao and Arthur C. Sanderson, "Task Sequence Planning Using Fuzzy Petri Nets", *IEEE Transactions on Systems, Man, and Cybernetics*, May 1995, Volume 25, Number 5, pp. 755-768.

[Capizzi et al., 1993] Angelo Capizzi, Gaetano Messina, and Guido Tricomi, "Robot Programming Languages Standardization in Manufacturing Environment", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, July 1993, Volume 1, pp. 488-492.

[Caracciolo et al., 1995] R. Caracciolo, E. Ceresole, T. De Martino, and F. Giannini, "From CAD Models to

Assembly Planning", *Graphics and Robotics*, Wolfgang Straber and Friedrich Wahl, editors, 1995, 115-129.

[Cardarelli et al., 1995] Gino Cardarelli, Mario Palumbo, and Pacifico Marcello Pelagagge, "Assembly with co-operating robots", *Assembly Automation*, 1995, Volume 15, Number 2, pp. 36-40.

[Casavant and Kuhl, 1990] T. L. Casavant and J. G. Kuhl, "A Communicating Finite Automata Approach to Modeling Distributed Computation and its Application to Distributed Decision-Making", *IEEE Transactions on Computers*, May 1990, Volume 39, Number 5, pp. 628-639.

[Cash and Wilhelm, 1986] Charles E. Cash and Wilbert E. Wilhelm, "A Simulation Modeling Approach for Analyzing Robotic Assembly Cells", *Winter Simulation Conference Proceedings*, December 1986, pp. 594-596.

[Causey and Quinn, 1997] Greg C. Causey and Roger D. Quinn, "Design of a Flexible Parts Feeding System", *Proceedings of the International Conference on Robotics and Automation*, April 1997, pp. 1235-1240.

[Chaib-Draa et al., 1992] B. Chaib-Draa, B. Moulin, R. Mandiau, and P. Millot, "Trends in Distributed Artificial Intelligence", *Artificial Intelligence Review*, 1992, Volume 6, pp. 35-66.

[Chandrasekaran, 1981] B. Chandrasekaran, "Natural and Social System Metaphors for Distributed Problem Solving: Introduction to the Issue", *IEEE Transactions on Systems, Man, and Cybernetics*, January 1981, Volume 11, Number 1, pp. 1-4.

[Charalambous and Hindi, 1991] O. Charalambous and K. S. Hindi, "A review of artificial intelligence-based job-shop scheduling systems", *Information and Decision Technologies*, 1991, Volume 17, pp. 189-202.

[Chen and Pao, 1993] C. L. Philip Chen and Yoh-Han Pao, "An Integration of Neural Network and Rule-Based Systems for Design and Planning of Mechanical Assemblies", *IEEE Transactions on Systems, Man, and Cybernetics*, September 1993, Volume 23, Number 5, pp. 1359-1371.

[Chen and Trivedi, 1992] ChuXin Chen and Mohan M. Trivedi, "Architecture and Automatic Task Planning for Integrated Sensor-Based Robots", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, July 1992, pp. 925-932.

[Chen and Trivedi, 1995] ChuXin Chen and Mohan M. Trivedi, "Task Planning and Action Coordination in Integrated Sensor-Based Robots", *IEEE Transactions on Systems, Man, and Cybernetics*, 1995, Volume 25, Number 4, pp. 569-591.

[Cheng et al., 1991] X. Cheng, D. Kappey, and J. Schloen, "Elements of an Advanced Robot Control System for Assembly Tasks", *Fifth International Conference on Advanced Robotics*, June 1991, Volume 1, pp. 411-416.

[Chochon and Alami, 1986] H. Chochon and R. Alami, "NNS, A Knowledge-Based On-Line System for an Assembly Workcell", *IEEE International Conference on Robotics and Automation*, 1986, pp. 603-609.

[Chongstitvatana, 1994] Prabhas Chongstitvatana, "Vision-based Behavioural Modules for Robotic Assembly Systems", *Proceedings of the 6th International Conference on Tools with Artificial Intelligence*, November 1994, pp. 312-316.

[Chu and ElMaraghy, 1993] H. Chu and H. A. ElMaraghy, "Integration of Task Planning and Motion Control in a Multi-Robot Assembly Workcell", *Robotics & Computer Integrated Manufacturing*, 1993, Volume 10, Number 3, pp. 235-255.

[Chu et al., 1991] H. Chu, J. M. Rondeau, and H. A. ElMaraghy, "Multi-Robot Plan Generation and Control in a Mechanical Assembly System", *Proceedings of the International Symposium on Advanced Robot Technology*, March 1991, pp. 53-60.

[Chung and Malowany, 1988] T. C. A. Chung, and A. S. Malowany, "An Intelligent Robotic System for Circuit Board Assembly and Repair", *Computers in Engineering*, 1988, Volume 2, pp. 341-344.

[Chutima et al., 1992] P. Chutima, I. S. Fan, and S. J. Harrington, "Flexible Robotic Assembly Control Systems", *Third International Conference on Factory 2000*, 1992, pp. 71-76.

[Classe and Feldmann, 1986] D. Classe and K. Feldmann, "A Dual Armed Robot System for Assembly Tasks", *Proceedings of the 16th International Symposium on Industrial Robots*, October 1986, pp. 731-741.

[Clermont et al., 1986] G. Clermont, M. Hermant, and P. Gaspart, "Programming an assembly task with a hierarchical planning system", *Proceedings of the 7th International Conference on Assembly Automation*, February 1986, pp. 91-100.

[Collins et al., 1984] K. Collins, A. J. Palmer, and K. Rathmill, "The Development of a European Benchmark for the Comparison of Assembly Robot Programming Systems", *Proceedings of the 1st Robotics Europe Conference*, Robot Technology and Applications, June 1984, pp. 187-199.

[Connell, 1992] Jonathan H. Connell, "SSS: A Hybrid Architecture Applied to Robot Navigation", *IEEE International Conference on Robotics and Automation*, May 1992, pp. 2719-2724.

[Cook and Emrich, 1987] James H. Cook, and Mary Emrich, "Artificial Intelligence in Manufacturing", *Manufacturing High Technology Handbook*, 1987, pp. 741-764.

[Coupez et al., 1989] D. Coupez, A. Delchambre, and P. Gaspart, "The Supervision and Management of A Two Robots Flexible Assembly System", *Proceedings of the International Conference on Robotics and Automation*, 1989, Volume 1, pp. 540-550.

[Crocker et al., 1996] Mike Crocker, Larry Harada, and Ram Sriram, "AIMSNET: Technology Infrastructure to support Agile Electronic Commerce using the Internet", *Fifth National Agility Conference*, March 1996.

[Culbreth et al., 1994] C. Thomas Culbreth, Janette M. Hopkins, and Russell E. King, "An Object-Oriented Control Architecture for a Flexible Manufacturing Cell: A Case Study", *International Journal of Flexible Automation and Integrated Manufacturing*, 1994, Volume 2, Numbers 1-2, pp. 95-117.

[Cutkosky et al., 1993] Mark R. Cutkosky, Robert S. Engelmore, Richard E. Fikes, Michael R. Genesereth, Thomas R. Gruber, William S. Mark, Jay M. Tenenbaum, and Jay C. Weber, "PACT: An Experiment in Integrating Concurrent Engineering Systems", *Computer*, January 1993, Volume 26, Number 1, pp. 28-38.

[Czarnecki, 1995] Chris Czarnecki, "Automated Stripping: A Robotic Handling Cell for Garment Manufacture", *IEEE Robotics & Automation Magazine*, June 1995, Volume 2, Number 2, pp. 4-8.

[Davis and Smith, 1983] Randall Davis and Reid G. Smith, "Negotiation as a Metaphor for Distributed Problem

Solving". *Artificial Intelligence*, 1983, Volume 20, pp. 63-109.

[Davis et al., 1993] Randall Davis, Howard Shrobe, and Peter Szolovits, "What Is a Knowledge Representation?", *AI Magazine*, Spring 1993, Volume 14, Number 1, pp. 17-33.

[Deacon and Malcolm, 1994] Graham E. Deacon and Chris Malcolm, "A Robot System Designed for Task-Level Assembly", Department of Artificial Intelligence, *University of Edinburgh*, 1994, Research Paper Number 687.

[Decker, 1987] Keith S. Decker, "Distributed Problem-Solving Techniques: A Survey", *IEEE Transactions on Systems, Man, and Cybernetics*, September/October 1987, Volume 17, Number 5, pp. 729-740.

[Delchambre et al., 1991] A. Delchambre, P. Gaspart, D. Coupez, and P. R. Glibert, "Knowledge-based Process Planning, Scheduling and Error Recovery in Robotized Assembly", *Expert Systems and Robotics*, T. Jordanides and B. Torby, editors, 1991, NATO ASI Series, Volume F71, pp. 619-639.

[Demazeau and Müller, 1989] Yves Demazeau and Jean Pierre Müller, "Decentralized Artificial Intelligence", *Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Yves Demazeau and Jean Pierre Müller, editors, Elsevier Science Publishers, August 1989, pp. 3-13.

[Deneubourg et al., 1991] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chrétrien, "The Dynamics of Collective Sorting Robot-Like Ants and Ant-Like Robots", *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, Jean-Arcady Meyer and Stewart W. Wilson, editors, 1991, pp. 356-363.

[Desrochers and Silva, 1994] A. A. Desrochers and M. Silva, editors, Special Issue on Computer Integrated Manufacturing, *IEEE Transactions on Robotics and Automation*, April 1994, Volume 10, Number 2.

[Dilts et al., 1991] D. M. Dilts, N. P. Boyd, and H. H. Whorms, "The Evolution of Control Architectures for Automated Manufacturing Systems", *Journal of Manufacturing Systems*, 1991, Volume 10, Number 1, pp. 79-93.

[Draper Laboratory, 1984] *Flexible Manufacturing Systems Handbook*, Prepared by the staff of Automation and Management Systems Division - The Charles Stark Draper Laboratory Inc., Noyes Publications, 1984.

[Drogoul and Ferber, 1993] Alexis Drogoul and Jacques Ferber, "From Tom Thumb to the Dockers: Some Experiments with Foraging Robots", *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, Jean-Arcady Meyer, Herbert L. Roitblat, and Stewart W. Wilson, editors, 1993, pp. 451-459.

[D'Souza and Khator, 1993] Kelwyn A. D'Souza and Suresh K. Khator, "A Petri net approach for modelling controls of a computer-integrated assembly cell", *International Journal of Computer Integrated Manufacturing*, 1993, Volume 6, Number 5, pp. 302-310.

[Dubreuil, 1995] Christophe Dubreuil, "Towards a Testbed for Multi-Agents Robotics", *Proceedings of the First International Workshop on Decentralized Intelligent and Multi-Agent Systems*, November 1995, pp. II/132 to II/139.

[Dudek et al., 1993] G. Dudek, M. Jenkin, and D. Wilkes, "A Taxonomy for Swarm Robots", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, July 1993, Volume 1, pp. 441-

447.

[Duffie et al., 1988] Neil A. Duffie, Ramesh Chitturi, and Jong-I Mou, "Fault-tolerant Heterarchical Control of Heterogeneous Manufacturing System Entities", *Journal of Manufacturing Systems*, 1988, Volume 7, Number 4, pp. 315-328.

[Duffy and Herd, 1991] N. D. Duffy and J. T. Herd, "A societal architecture for robotic applications", *Computing & Control Engineering Journal*, November 1991, Volume 2, Number 6, pp. 269-274.

[Duffy et al., 1989] N. Duffy, D. Allan, and J. T. Herd, "Real-time collision avoidance system for multiple robots operating in shared work-space", *IEE Proceedings-E*, Computers and Digital Techniques, November 1989, Volume 136, Number 6, pp. 478-484.

[Duffy et al., 1991] N. D. Duffy, J. T. Herd, and N. J. Eccles, "Agents Get Framed In Novel Architecture", *Industrial Robot*, 1991, Volume 18, Number 2, pp. 23-26.

[Durfee, 1991] Edmund H. Durfee, "The Distributed Artificial Intelligence Melting Pot", *IEEE Transactions on Systems, Man, and Cybernetics*, November/December 1991, Volume 21, Number 6, pp. 1301-1306.

[Durfee and Rosenschein, 1994] Edmund H. Durfee and Jeffrey S. Rosenschein, "Distributed Problem Solving and Multi-Agent Systems: Comparisons and Examples", *Papers from the Thirteenth International Workshop on Distributed Artificial Intelligence*, July 1994, pp. 52-62.

[Durfee et al., 1989] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill, "Trends in Cooperative Distributed Problem Solving", *IEEE Transactions on Knowledge and Data Engineering*, March 1989, Volume 1, Number 1, pp. 63-83.

[Eccles et al., 1991] J. Eccles, J. T. Herd, and N. D. Duffy, "Multi-Sensor Integration for Robotic Assembly", *Proceedings of the Fifth International Conference on Advanced Robotics*, June 1991, pp. 1295-1298.

[Elfes, 1986] Alberto Elfes, "A distributed control architecture for an autonomous mobile robot", *The International Journal for Artificial Intelligence in Engineering*, October 1986, Volume 1, Number 2, pp. 99-108.

[Ellis and Stroustrup, 1990] M. A. Ellis and B. Stroustrup, *The Annotated C++ Reference Manual*, Addison-Wesley Publishing Co., 1990.

[ElMaraghy and Knoll, 1989] Hoda A. ElMaraghy and Larry Knoll, "Flexible Assembly of a Family of DC Motors", *Manufacturing Review*, December 1989, Volume 2, Number 4, pp. 250-256.

[ElMaraghy and Laperrière, 1992] Hoda A. ElMaraghy and Luc Laperrière, "Modelling and sequence generation for robotized mechanical assembly", *Robotics and Autonomous Systems*, 1992, Volume 9, pp. 137-147.

[Esteban and Courvoisier, 1991] P. Esteban and M. Courvoisier, "Multilevel Hierarchical Control Applied to a Demonstration Flexible Assembly Cell", *International Conference on Industrial Electronics, Control and Instrumentation*, October 1991, pp. 878-883.

[Etzioni and Weld, 1994] Oren Etzioni and Daniel Weld, "A Softbot-Based Interface to the Internet", *Communications of the ACM*, July 1994, Volume 37, Number 7, pp. 72-76.

[Etzioni et al., 1995] Oren Etzioni, Henry M. Levy, and Richard B. Segal, "The Softbot Approach to OS

Interfaces", *Software*, July 1995, Volume 12, Number 4, pp. 42-51.

[Fabian and Lennartson, 1992] M. Fabian and B. Lennartson, "Control of Manufacturing Systems: An Object-Oriented Approach", *7th Symposium on Information Control Problems in Manufacturing Technology*, Marek B. Zaremba, editor, May 1995, Volume 1, pp. 47-52.

[Farsi, 1993] Mohammad Farsi, "Flexible and Reliable Robotics Cell in Factory Automation", *IEEE International Conference on Systems, Man, and Cybernetics*, Systems Engineering in the Service of Humans, October 1993, Volume 1, pp. 520-525.

[Farsi, 1994] Mohammad Farsi, "Flexible Robotics Cell in Factory Automation: Communication Concept", *Proceedings of the Third IEEE Conference on Control Applications*, August 1994, Volume 3, pp. 1763-1768.

[Ferguson, 1992] Innes A. Ferguson, *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*, Doctoral Dissertation, University of Cambridge, October 1992.

[Ferullo and Billatos, 1992] David A. Ferullo and Samir B. Billatos, "Sensor-based Networking for Intelligent Assembly Cell", *Proceedings of the Japan-USA Symposium on Flexible Automation*, July 1992, Volume 2, pp. 1629-1635.

[Finin et al., 1994] Tim Finin, Richard Fritzson, Don McKay, and Robin McEntire, "KQML as an Agent Communication Language", *Proceedings of the Third International Conference on Information and Knowledge Management*, November 1994.

[Fischer, 1993] Klaus Fischer, "Knowledge-based Reactive Scheduling in a Flexible Manufacturing System", *Proceedings of the IFIP TC5/WG5.7 International Workshop on Knowledge-Based Reactive Scheduling*, October 1993, pp. 1-18.

[Fischer, 1994] Klaus Fischer, "The Design of an Intelligent Manufacturing System - A Multi-Agent System Approach", *Proceedings of the 2nd International Workshop on Cooperating Knowledge Based Systems*, S. M. Deen, editor, 1994, pp. 83-99.

[Freedman and Malowany, 1988] P. Freedman and A. Malowany, "SAGE: A Decision Support System for the Sequencing of Operations within a Robotic Workcell", *Decision Support Systems*, 1988, Volume 4, pp. 329-343.

[Frommherz and Werling, 1989] B. Frommherz and G. Werling, "A Graphical Implicit Programming System for Robot Action Planning", *Intelligent Autonomous Systems 2*, T. Kanade, F. C. A. Groen, and L. O. Hertzberger, editors, pp. 196-207.

[Fu and Hsu, 1993] Li-Chen Fu and Yung-Jen Hsu, "Fully Automated Two-Robot Flexible Assembly Cell", *Proceedings of the International Conference on Robotics and Automation*, May 1993, Volume 3, pp. 332-338.

[Fukuda et al., 1989] Toshio Fukuda, Seiya Nakagawa, Yoshio Kawauchi, and Martin Buss, "Structure Decision Method for Self Organizing Robots Based on Cell Structures-CEBOT", *IEEE International Conference on Robotics and Automation*, 1989, Volume 2, pp. 695-700.

[Gasmi et al., 1990] B. Gasmi, C. Castel, C. Reboulet, and R. Houdebert, "SAFIR: Flexible Multi-robots

Assembly System", *Proceedings of Rensselaer's Second International Conference on Computer Integrated Manufacturing*, May 1990, pp. 582-587.

[Gasser, 1992] Les Gasser, "An Overview of DAI", *Distributed Artificial Intelligence: Theory and Praxis*, Nicholas M. Avouris and Les Gasser, editors, Kluwer Academic Publishers, 1992, pp. 9-30.

[Gasser and Huhns, 1989] Les Gasser and Michael N. Huhns, "Themes in Distributed Artificial Intelligence Research", *Distributed Artificial Intelligence*, Les Gasser and Michael N. Huhns, editors, 1989, pp. vii-xv.

[Gauthier et al., 1987] David Gauthier, Paul Freedman, Gregory Carayannis, and Alfred S. Malowany, "Interprocess Communication for Distributed Robotics", *IEEE Journal of Robotics and Automation*, December 1987, Volume 3, Number 6, pp. 493-504.

[Genesereth and Fikes, 1992] Michael R. Genesereth and Richard E. Fikes, "Knowledge Interchange Format Version 3.0 Reference Manual", Computer Science Department, *Stanford University*, June 1992.

[Genesereth and Ketchpel, 1994] Michael R. Genesereth and Steven P. Ketchpel, "Software Agents", *Communications of the ACM*, July 1994, Volume 37, Number 7, pp. 48-53.

[Georgeff, 1983] Michael Georgeff, "Communication and Interaction in Multi-Agent Planning", *Proceedings of the National Conference on Artificial Intelligence*, 1983, pp. 125-129.

[Gini and Gini, 1991] Maria Gini and Giuseppina Gini, "Programming for Intelligent Robot Systems", *Intelligent Robotic Systems*, Spyros G. Tzafestas, editor, Marcel Dekker, 1991, pp. 165-208.

[Glibert et al., 1990] P. R. Glibert, D. Coupez, Y. M. Peng, and A. Delchambre, "Scheduling of a multi-robot assembly cell", *Computer-Integrated Manufacturing Systems*, November 1990, Volume 3, Number 4, pp. 236-245.

[Goetz, 1989] Philip W. Goetz, editor in chief, "Automation", *The New Encylopædia Britannica*, 1989, Volume 14, pp. 529-538.

[Gosling and McGilton, 1995] James Gosling and Henry McGilton, *The Java Language Environment: A White Paper*, Sun Microsystems Inc., October 1995.

[Goto et al., 1980] Tatsuo Goto, Kiyoo Takeyasu, and Tadao Inoyama, "Control Algorithm for Precision Insert Operation Robots", *IEEE Transactions on Systems, Man, and Cybernetics*, May 1993, Volume 10, Number 1, pp. 19-25.

[Gottschlich and Kak, 1989] Susan N. Gottschlich and Avinash C. Kak, "A Dynamic Approach to High-Precision Parts Mating", *IEEE Transactions on Systems, Man, and Cybernetics*, July/August 1989, Volume 19, Number 4, pp. 797-810.

[Gottschlich and Kak, 1990] S. N. Gottschlich and A. C. Kak, "Assembly Knowledge Representation for Assembly Motion Planning and Execution", *Proceedings of 5th IEEE International Symposium on Intelligent Control*, September 1990, pp. 948-956.

[Gottschlich et al., 1994] Susan Gottschlich, Carlos Ramos, and Damian Lyons, "Assembly and Task Planning: A Taxonomy", *IEEE Robotics and Automation Magazine*, September 1994, Volume 1, Number 3, pp. 4-12.

[Grant, 1992] T. J. Grant, "A review of Multi-Agent Systems techniques, with application to Columbus User Support Organization", *Future Generation Computer Systems*, 1992, Volume 7, pp. 413-437.

[Graves, 1987] Robert J. Graves, "Hierarchical Scheduling Approach in Flexible Assembly Systems", *IEEE International Conference on Robotics and Automation*, March 1987, pp. 118-123.

[Gruber, 1992] Thomas R. Gruber, "A Translation Approach to Portable Ontology Specifications", *Knowledge Systems Laboratory*, Computer Science Department, Stanford University, Technical Report KSL 92-71.

[Gunasekaran et al., 1995] A. Gunasekaran, T. Martikainen, and P. Yli-Olli, "Flexible Manufacturing Systems: An Investigation For Research and Application", *Flexible Manufacturing Systems: Recent Developments*, A. Raouf and M. Ben-Daya, editors, 1995, pp. 3-44.

[Gürocak and Lazaro, 1995] H. B. Gürocak and A. de Sam Lazaro, "Fuzzy Logic and Position Sensing for Precision Assembly", *Journal of Robotic Systems*, February 1995, Volume 12, Number 2, pp. 135-146.

[Gutsche et al., 1995] R. Gutsche, F. Rohrdanz, and F. M. Wahl, "Assembly Planning Using Symbolic Spatial Relationships", *Graphics and Robotics*, Wolfgang Straber and Friedrich Wahl, editors, 1995, 87-114.

[Hahndel and Levi, 1994a] Stefan Hahndel and Paul Levi, "A Distributed Task Planning Method for Autonomous Agents in a FMS", *Proceedings of the International Conference on Intelligent Robots and Systems*, 1994, pp. 1285-92.

[Hahndel and Levi, 1994b] S. Hahndel and P. Levi, "Optimized Distributed Production Planning", *Second International Conference on Intelligent Systems Engineering*, September 1994, pp. 419-424.

[Hara and Azuma, 1988] S. Hara and K. Azuma, "Cell Production System for Assembly", *Robotics & Computer-Integrated Manufacturing*, 1988, Volume 4, Numbers 3-4, pp. 379-385.

[Hara and Nagata, 1993] Isao Hara and Tadashi Nagata, "Robot Assembly Planning using Contract Nets", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, July 1993, Volume 3, pp. 1971-1976.

[Hardy et al., 1987] N. W. Hardy, D. P. Barnes, and M. H. Lee, "Declarative Sensor Knowledge in a Robot Monitoring System", *Languages for Sensor-Based Control in Robotics*, Ulrich Rembold and Klaus Hörmann, editors, 1987, NATO ASI Series, Volume F29, pp. 169-187.

[Harmon et al., 1984] S. Y. Harmon, D. W. Gage, W. A. Aviles, and G. L. Bianchini, "Coordination of Intelligent Subsystems in Complex Robots", *The First Conference on Artificial Intelligence Applications*, December 1984, pp. 64-69.

[Harmon et al., 1986] S. Y. Harmon, W. A. Aviles, and D. W. Gage, "A Technique for Coordinating Autonomous Robots", *Proceedings of the IEEE International Conference on Robotics and Automation*, April 1986, pp. 2029-2034.

[Hasegawa et al., 1991] Tsutomu Hasegawa, Suehiro Takashi, and Kunikatsu Takase, "A Model-Based Manipulation System with Skill-Based Execution in Unstructured Environment", *Fifth International Conference on Advanced Robotics*, June 1991, Volume 2, pp. 970-975.

[Hasegawa et al., 1992] Tsutomu Hasegawa, Suehiro Takashi, and Kunikatsu Takase, "A Model-Based

Manipulation System with Skill-Based Execution", *IEEE Transactions on Robotics and Automation*, 1992, Volume 8, Number 5, pp. 535-544.

[Hasegawa et al., 1994] Tetsuo Hasegawa, Ling Gou, Shinsuke Tamura, Peter B. Luh, and John M. Oblak, "Holonic Planning and Scheduling Architecture for Manufacturing," *Proceedings of the 2nd International Workshop on Cooperating Knowledge Based Systems*, S. M. Deen, editor, 1994, pp. 125-139.

[Hatvany, 1985] J. Hatvany, "Intelligence and Cooperation in Heterarchic Manufacturing Systems", *Robotics & Computer-Integrated Manufacturing*, 1985, Volume 2, Number 2, pp. 101-104.

[Hawker et al., 1986] J. Scott Hawker, R. N. Nagel, Richard Roberts, and Nicholas G. Odrey, "Multiple Robotic Manipulators", *Byte*, January 1986, pp. 203-219.

[He et al., 1990] Shulin He, Li Lu, and A. H. Soni, "Configuration Design and Operation Planning of Multi-Robot-Assembly Work Cells", *2nd Conference on Flexible Assembly Systems*, The Design Engineering Division, ASME 1990, Volume 28, pp. 121-126.

[Herd et al., 1990] J. T. Herd, N. D. Duffy, G. P. Philip, A. C. Davidson, and N. J. Eccles, "A design/implementation architecture for complex multi-robot systems", *Cooperative Intelligent Robotics in Space*, SPIE proceedings, November 1990, Volume 1387, pp. 194-201.

[Hern, 1988] Luis Eduardo Castillo Hern, "On distributed artificial intelligence", *The Knowledge Engineering Review*, March 1988, Volume 3, Part 1, pp. 21-57.

[Hewitt, 1977] Carl Hewitt, "Viewing Control Structures as Patterns of Passing Messages", *Artificial Intelligence*, 1977, Volume 8, Number 3, pp. 323-364.

[Hewitt and Inman, 1991] Carl Hewitt and Jeff Inman, "DAI Betwixt and Between: From 'Intelligent Agents' to Open Systems Science", *IEEE Transactions on Systems, Man, and Cybernetics*, November/December 1991, Volume 21, Number 6, pp. 1409-1419.

[Hewitt and Jong, 1983] Carl Hewitt and Peter de Jong, "Analyzing the Roles of Descriptions and Actions in Open Systems", *Proceedings of the National Conference on Artificial Intelligence*, 1983, pp. 162-167.

[Hindi and Singh, 1995] K. S. Hindi and M. G. Singh, "A Survey of Knowledge-based Industrial Scheduling", *Artificial Intelligence in Industrial Decision Making, Control and Automation*, S. G. Tzafestas and H. B. Verbruggen, editors, Kluwer Academic Publishers, 1995, pp. 663-685.

[Holm et al., 1993] Hans Holm, Hans Peter Boss, Haraldur Petursson, and Jorgen A. Nielson, "Information System Structure for a Task Level Robot Assembly Language for On-line Program Generation", *Robotics & Computer-Integrated Manufacturing*, 1993, Volume 10, Numbers 1-2, pp. 77-88.

[Homem de Mello and Sanderson, 1991] Luiz S. Homem de Mello and Arthur C. Sanderson, "Representations of Mechanical Assembly Sequences", *IEEE Transactions on Robotics and Automation*, April 1991, Volume 7, Number 2, pp. 211-227.

[Hörmann, 1989] A. Hörman, "A Petri net based control architecture for a multi-robot system", *Proceedings of 4th IEEE International Symposium on Intelligent Control*, 1989, pp. 493-498.

[Hörmann, 1992] Andreas Hörmann, "On-line Planning of Action Sequences for a Two-Arm Manipulator System",

Proceedings of the International Conference on Robotics and Automation, May 1992, pp. 1109-1114.

[Hörmann and Rembold, 1991] Andreas Hörmann and Ulrich Rembold, "Development of an Advanced Robot for Autonomous Assembly", *Proceedings of the International Conference on Robotics and Automation*, April 1991, pp. 2452-2457.

[Hörmann et al., 1989] A. Hörmann, W. Meier, and J. Schloen, "A Control Architecture for an Advanced Fault-Tolerant Robot System", *Intelligent Autonomous Systems 2*, T. Kanade, F. C. A. Groen, and L. O. Hertzberger, editors, pp. 576-585.

[Hoska, 1988] David Hoska, "Assembly Robots", *International Encyclopedia of Robotics: Applications and Automation*, Richard D. Dorf, Editor-in-Chief, 1988, Volume 1, pp. 117-128.

[Hou, 1989] E. S. H. Hou, "Model-based reasoning of sensory data for robotic assembly", *Sensor Fusion II: Human and Machine Strategies*, SPIE proceedings, November 1989, Volume 1198, pp. 490-493.

[Howell, 1988] Vincent W. Howell, "Integration, Systems of", *International Encyclopedia of Robotics: Applications and Automation*, Richard D. Dorf, Editor-in-Chief, 1988, Volume 2, pp. 679-691.

[Hsu and Fu, 1995] Yung-Jen Hsu and Li-Chen Fu, "Fully Automated Robotic Assembly Cell: Scheduling and Simulation", *Proceedings of the International Conference on Robotics and Automation*, May 1995, Volume 1, pp. 208-214.

[Hu and Brady, 1996] Huosheng Hu and Michael Brady, "A parallel processing architecture for sensor-based control of intelligent mobile robots", *Robotics and Autonomous Systems*, 1996, Volume 17, pp. 235-257.

[Hu et al., 1995] H. Hu, J. M. Brady, J. Grothusen, F. Li, and P. J. Probert, "LICAs: A Modular Architecture for Intelligent Control of Mobile Robots", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, August 1995, pp. 471-476.

[Huang and Lee, 1991] Y. F. Huang and C. S. G. Lee, "A Framework of Knowledge-based Assembly Planning", *Proceedings of the International Conference on Robotics and Automation*, April 1991, pp. 599-604.

[Huang et al., 1997] Tz-Shian Huang, Li-Chen Fu, and Yung-Yu Chen, "Design and Analysis of a Dynamic Scheduler for a Flexible Assembly System", *Proceedings of the International Conference on Robotics and Automation*, April 1997, pp. 3334-3339.

[Hutchison and Kak, 1989] Seth A. Hutchison and Avinash C. Kak, "Planning Sensing Strategies in a Robot Work Cell with Multi-Sensor Capabilities", *IEEE Transactions on Robotics and Automation*, December 1989, Volume 5, Number 6, pp. 765-783.

[Hutchison and Kak, 1990] Seth A. Hutchison and Avinash C. Kak, "SPAR: A Planner That Satisfies Operational and Geometric Goals in Uncertain Environments", *AI Magazine*, Spring 1990, Volume 11, Number 1, pp. 30-61.

[Hutt, 1994a] *Object Analysis and Design: Description of Methods*, Andrew T. F. Hutt, editor, John Wiley & Sons, 1994.

[Hutt, 1994b] *Object Analysis and Design: Comparison of Methods*, Andrew T. F. Hutt, editor, John Wiley & Sons, 1994.

[Inoue, 1979] Hirochika Inoue, "Force Feedback in Precise Assembly Tasks", *Artificial Intelligence: An MIT Perspective*, Volume 2: Understanding Vision, Manipulation, Computer Design, and Symbol Manipulation, Patrick Henry Winston and Richard Henry Winston, editors, pp. 219-241.

[Ishida et al., 1991] Y. Ishida, H. Asama, I. Endo, K. Ozaki, and A. Matsumoto, "Communication and Cooperation in an Autonomous an Decentralized Robot System", *Distributed Intelligence Systems*, International Federation of Automatic Control, H. E. Stephanou and A. H. Levis, editors, pp. 299-304.

[Jagannathan and Evans, 1994a] S. Jagannathan and M. Evans, "Intelligent Control of Flexible Autonomous Robots Part I: Architectural Considerations", *The IEEE International Conference on Neural Networks*, June 1994, Volume V, pp. 2837-2841.

[Jagannathan and Evans, 1994b] S. Jagannathan and M. Evans, "Intelligent Control of Flexible Autonomous Robots Part II: Implementation", *The IEEE International Conference on Neural Networks*, June 1994, Volume V, pp. 2831-2836.

[James and Guptill, 1993] Paul A. James and Rick Guptill, "Using Simulation to Evaluate Robot Programming Decision Making", *Industrial Robot*, 1993, Volume 20, Number 4, pp. 26-31.

[Jennings, 1994] Nick Jennings, *Cooperation in Industrial Multi-Agent Systems*, Doctoral Dissertation, World Scientific Series in Computer Science, Volume 43, 1994.

[Jennings and Wooldridge, 1995] Nicholas R. Jennings and Michael J. Wooldridge, "Applying Agent Technology", *Applied Artificial Intelligence*, 1995, Volume 9, pp. 357-369.

[Kaiser et al., 1994] M. Kaiser, A. Giordana, and M. Nuttin, "Integrated Acquisition, Execution, Evaluation, and Tuning of Elementary Skills for Intelligent Robots", *Artificial Intelligence in Real Time Control*, International Federation of Automatic Control, October 1994, pp. 117-122.

[Kak et al., 1986] A. C. Kak, K. L. Boyer, C. H. Chen, R. J. Safranek, and H. S. Yang, "A Knowledge-Based Robotic Assembly Cell", *IEEE Expert*, Spring 1986, Volume 1, Number 1, pp. 63-83.

[Kamel and Syed, 1988] M. Kamel and A. Syed, "An Automated Multiagent Planning System", *Expert Systems and Intelligent Manufacturing*, Michael D. Oliff, editor, 1988, pp. 367-384.

[Kamel and Syed, 1994] M. Kamel and A. Syed, "A Multiagent Task Planning Method for Agents with Disparate Capabilities", *International Journal of Advanced Manufacturing Technology*, 1994, Volume 9, Number 6, pp. 408-420.

[Kang and Wenhan, 1993] Li Kang and Qian Wenhan, "Fuzzy Expert System in Robotic Assembly Workcell", *Proceedings of IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering*, October 1993, pp. 738-741.

[Kantrowitz, 1994] Mark Kantrowitz, editor, *Prime Time Freeware for AI*, Issue 1-1, 1994.

[Kao et al., 1996] Ju-Hsien Kao, James S. Hemmerle, and Friedrich B. Prinz, "Collision Avoidance Using Asynchronous Teams", *Proceedings of the International Conference on Robotics and Automation*, April 1996, Volume 2, pp. 1093-1099.

[Kapitanovksy and Maimon, 1993] A. Kapitanovsky and O. Maimon, "Robot Programming System for Assembly:

Conceptual Graph-Based Approach", *Journal of Intelligent and Robotic Systems*, 1993, Volume 8, pp. 35-62.

[Karp, 1993] Peter D. Karp, "The Design Space of Frame Knowledge Representation Systems", *SRI AI Center Technical Note #520*, 1993.

[Kautz et al., 1994] Henry A. Kautz, Bart Selman, and Michael Coen, "Bottom-Up Design of Software Agents", *Communications of the ACM*, July 1994, Volume 37, Number 7, pp. 143-146.

[Kawauchi et al., 1993] Yoshio Kawauchi, Makoto Inaba, and Toshio Fukuda, "A Principle of Distributed Decision Making of Cellular Robotic System (CEBOT)", *Proceedings of the International Conference on Robotics and Automation*, May 1993, Volume 3, pp. 833-838.

[Kelley, 1991] Robert B. Kelley, "Hierarchy for Intelligent On-Line Execution", *Intelligent Robotic Systems*, Spyros G. Tzafestas, editor, Marcel Dekker, 1991, pp. 85-107.

[Kelley et al., 1990] Robert B. Kelley, Deepak Sood, and Michael C. Repko, "Integrated Sensing for Circuit Board Insertion", *Journal of Robotic Systems*, 1990, Volume 7, Number 3, pp. 487-505.

[Kempf, 1988] Karl Kempf, "Practical Applications of Artificial Intelligence in Manufacturing", *Artificial Intelligence in Manufacturing, Assembly and Robotics*, Horst O. Bunke, editor, 1988, pp. 1-26.

[Kennedy and Neville, 1974] John B. Kennedy and Adam M. Neville, *Basic Statistical Method for Engineers & Scientists*, Harper & Row Publishers, 1974.

[Khoshafian and Abnous, 1995] Setrag Khoshafian and Razmik Abnous, *Object Orientation*, John Wiley & Sons, 1995.

[Kidd, 1994] Paul T. Kidd, "Agile Manufacturing: Key Issues", *Advances in Agile Manufacturing - Integrating Technology, Organization, and People*, Paul T. Kidd and Waldemar Karwowoski, editors, 1994, IOS Press, pp. 29-32.

[Kim et al., 1989] Dae-Won Kim, Myoung-Sam Ko, and Bum-Hee Lee, "An Approach to Modelling for Operation Management of Robotic Assembly Cells through Knowledge Base", *Proceedings of the 28th SICE Annual Conference*, July 1989, pp. 961-964.

[Kim et al., 1991] Dae-Won Kim, Bum-Hee Lee, and Myoung-Sam Ko, "A knowledge-based approach to modelling of robotic assembly cells", *Robotica*, 1991, Volume 9, pp. 31-42.

[Kim et al., 1997] Yoohwan Kim, Ju-Yeon Jo, Virgilio B. Velasco Jr., Nicholas A. Barendt, Andy Podgurski, Gultekin Ozsoyoglu, and Frank L. Merat, "A Flexible Software Architecture for Agile Manufacturing", *Proceedings of the International Conference on Robotics and Automation*, April 1997, pp. 3043-3047.

[Kimbler and Malave, 1989] D. L. Kimbler and Cesar O. Malave, "Design of a knowledge-based generic control protocol generator for automatic assembly operations", *International Journal of Computer Integrated Manufacturing*, 1989, Volume 2, Number 1, pp. 36-47.

[Ko and Lee, 1991] Woosung Ko and Sukhan Lee, "Robot Task Planning based on Resource Reasoning", *Proceedings of the International Conference on Robotics and Automation*, April 1991, pp. 756-761.

[Kokkinaki and Valavanis, 1995] A. I. Kokkinaki and K. P. Valavanis, "On the Comparison of AI and DAI Based Planning Techniques for Automated Manufacturing Systems", *Artificial Intelligence in Industrial Decision Making, Control and Automation*, Spyros G. Tzafestas and Henk B. Verbruggen, editors, 1995, Kluwer Academic Publishers, pp. 573-629.

[Korson and McGregor, 1990] Tim Korson and John D. McGregor, "Understanding Object-Oriented: A Unifying Paradigm", *Communications of the ACM*, 1990, Volume 33, Number 9, pp. 40-60.

[Kountouris and Stephanou, 1990] Vasilios G. Kountouris and Harry E. Stephanou, "Task decomposition, distribution, and localization for intelligent robot coordination", *Cooperative Intelligent Robotics in Space*, SPIE proceedings, November 1990, Volume 1387, pp. 169-180.

[KQML, 1997] KQML Resource Page, "http://www.cs.umbc.edu/kqml/", Web Site, 1997.

[Kube and Zhang, 1993] C. Ronald Kube and Hong Zhang, "Collective Robotic Intelligence", *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, Jean-Arcady Meyer, Herbert L. Roitblat, and Stewart W. Wilson, editors, 1993, pp. 460-468.

[Kube et al., 1993] C. Ronald Kube, Hong Zhang, and Xiaohuan Wang, "Controlling Collective Tasks with an ALN", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, July 1993, Volume 1, pp. 289-293.

[Kuokka and Harada, 1995] Daniel R. Kuokka and Larry T. Harada, "Communication infrastructure for concurrent engineering", *Artificial Intelligence for Engineering Design, Analysis & Manufacturing*, September 1995, Volume 9, Number 4, pp. 283-297.

[Laengle and Lueth, 1994] T. H. Laengle and T. C. Lueth, "Decentralized Control of Distributed Intelligent Robots and Subsystems", *Artificial Intelligence in Real Time Control*, International Federation of Automatic Control, September 1994, Volume 19, pp. 281- 286.

[Lashkari et al., 1994] Yezdi Lashkari, Max Metral, and Pattie Maes, "Collaborative Interface Agents", *Proceedings of the Twelfth National Conference on Artificial Intelligence*, July 1994, Volume 1, pp. 444-449.

[Lee et al., 1987] C. N. Lee, M. Y. Chiu, P. Liu, amd S. J. Clark, "Model-based hierarchical diagnosis of robotic assembly cell", *Intelligent Robots and Computer Vision VI*, SPIE Proceedings, 1987, pp. 182-189.

[Lee et al., 1994] M. H. Lee, Y. F. Li, J. J. Rowland, and M. A. Rodrigues, "Structured Supervisory Control and Grasp Determination for Product Packing by Robot", *Factory 2000 — Advanced Factory Automation*, October 1994, pp. 496-501.

[Lefrancois and Montreuil, 1994] Pierre Lefrancois and Benoit Montreuil, "An Object-Oriented Knowledge Representation for Intelligent Control of Manufacturing Workstations", *IIE Transactions*, January 1994, Volume 26, Number 1, pp. 11-26.

[Lesser, 1995] Victor R. Lesser, "Multiagent Systems: An Emerging Subdiscipline of AI", *ACM Computing Surveys*, September 1995, Volume 27, Number 3, pp. 340-342.

[Lesser and Erman, 1980] Victor R. Lesser and Lee D. Erman, "Distributed Interpretation: A Model and Experiment", *IEEE Transactions on Computers*, December 1980, Volume 29, Number 12, pp. 1144-1163.

[Lim, 1994] William Lim, "An Agent-Based Approach for Programming Mobile Robots", *Proceedings of the IEEE International Conference on Robotics and Automation*, May 1994, pp. 3584-3589.

[Lin and Hsu, 1995a] Fang-Chang Lin and Jane Yung-jen Hsu, "A Decentralized Cooperation Protocol for Autonomous Robotic Agents", *Proceedings of the Second International Symposium on Autonomous Decentralized Systems*, April 1995, pp. 420-426.

[Lin and Hsu, 1995b] Fang-Chang Lin and Jane Yung-jen Hsu, "Cooperation and Deadlock-Handling for an Object-Sorting Task in a Multi-agent Robotic System", *Proceedings of the International Conference on Robotics and Automation*, May 1995, Volume 3, pp. 2580-2585.

[Lin and Hsu, 1996a] Fang-Chang Lin and Jane Yung-jen Hsu, "Coordination-based Cooperation Protocol in Multi-agent Robotic Systems", *Proceedings of the International Conference on Robotics and Automation*, April 1996, Volume 2, pp. 1632-1637.

[Lin and Hsu, 1996b] Fang-Chang Lin and Jane Yung-jen Hsu, "Cost-balanced Cooperation Protocol in Multi-agent Robotic Systems", *Proceedings of the International Conference on Robotics and Automation*, June 1996, pp. 72-79.

[Lin and Solberg, 1992] Grace Yuh-Jiun Lin and James J. Solberg, "Integrated Shop Floor Control using Autonomous Agents", *IIE Transactions*, Volume 24, Number 3, pp. 57-71.

[Lin et al., 1994] Li Lin, Masatoshi Wakabayashi, and Sadashiv Adiga, "Object-Oriented Modeling and Implementation of Control Software for a Robotic Flexible Manufacturing Cell", *Robotics & Computer-Integrated Manufacturing*, 1994, Volume 11, Number 1, pp. 1-12.

[Liu and Sadler, 1992] Jian Liu and J. P. Sadler, "Robotic Assembly Cell Simulation", *4th Conference on Flexible Assembly Systems*, The Design Engineering Division, ASME 1992, Volume 48, pp. 125-130.

[López-Mellado, 1995] Ernesto López-Mellado, "Design of Knowledge-based Real-Time Controllers for Assembly Systems", *IEEE International Conference on Systems, Man, and Cybernetics*, Humans, Information and Technology, October 1995, pp. 1763-1767.

[Lotter, 1986] Bruno Lotter, "Planning and implementation of flexible assembly cells", *Proceedings of the 7th International Conference on Assembly Automation*, February 1986, pp. 1-9.

[Lozano-Pérez, 1983] Tomás Lozano-Pérez, "Robot Programming", *Proceedings of the IEEE*, 1983, Volume 71, Number 7, pp. 821-841.

[Lozano-Pérez et al., 1989] Tomás Lozano-Pérez, Joseph L. Jones, Emmanuel Mazer, and Patrick A. O'Donnell, "Task-Level Planning of Pick-and-Place Robot Motions", *Computer*, March 1989, pp. 21-29.

[Lozano-Pérez et al., 1992] Tomás Lozano-Pérez, Joseph L. Jones, Emmanuel Mazer, and Patrick A. O'Donnell, *Handey - A Robot Task Planner*, The MIT Press, 1992.

[Lumia et al., 1989] Ronald Lumia, John Fiala, and Albert Wavering, "The NASREM Robot Control System Standard", *Robotics & Computer-Integrated Manufacturing*, 1989, Volume 6, Numbers 4, pp. 303-308.

[Maimon, 1984] Oded Zvi Maimon, *Activity Controller for a Multiple Robot Assembly Cell*, Doctoral Dissertation, Purdue University, May 1984.

[Maimon and Fisher, 1988] Oded Z. Maimon and Edward L. Fisher, "An object-based representation method for a manufacturing cell controller", *The International Journal for Artificial Intelligence in Engineering*, 1988, Volume 3, Number 1, pp. 2-11.

[Maimon and Kapitanovsky, 1992] O. Maimon and A. Kapitanovsky, "Conceptual graph-based system for assembly program synthesis", *Robotica*, 1992, Volume 10, pp. 329-338.

[Maimon and Nof, 1985] O. Z. Maimon and S. Y. Nof, "Coordination of Robots Sharing Assembly Tasks", *Journal of Dynamic Systems, Measurement and Control*, Transactions of the ASME, December 1985, Volume 107, Number 4, pp. 299-307.

[Makino and Arai, 1994] Hiroshi Makino, and Tamio Arai, "New Developments in Assembly Systems", *Manufacturing Technology*, CIRP Annals 1994, Volume 43, Number 2, pp. 501-511.

[Malcolm and Smithers, 1990] Chris Malcolm and Tim Smithers, "Symbol Grounding via a Hybrid Architecture in an Autonomous Assembly System", *Robotics and Autonomous Systems*, 1990, Volume 6, pp. 123-144.

[Malcolm et al., 1989] Chris Malcolm, Tim Smithers, and John Hallam, "An Emerging Paradigm in Robot Architecture", *Intelligent Autonomous Systems 2*, T. Kanade, F. C. A. Groen, and L. O. Hertzberger, editors, pp. 545-564.

[Malowany and Malowany, 1988] M. E. Malowany and A. S. Malowany, "A Rule-Based Framework for Controlling a Robotic Workcell", *Proceedings of the Seventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, June 1988, pp. 191-198.

[Manivannan, 1993] S. Manivannan, "Robotic Collision Avoidance in a Flexible Assembly Cell Using a Dynamic Knowledge Base", *IEEE Transactions on Systems, Man, and Cybernetics*, May 1993, Volume 23, Number 3, pp. 766-782.

[Mascle, 1995] Christian Mascle, "Features Modeling in Assembly Sequences and Resource Planning", *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, August 1995, pp. 232-237.

[Masini et al., 1991] Gerald Masini, Amedeo Napoli, Dominique Colnet, Daniel Leonard, and Karl Tombre, *Object Oriented Languages*, Academic Press, 1991.

[Mason, 1981] Matthew T. Mason, "Compliance and Force Control for Computer Controlled Manipulators", *IEEE Transactions on Systems, Man, and Cybernetics*, June 1981, Volume 11, Number 6, pp. 418-433.

[Mataric, 1993] Maja J. Mataric, "Designing Emergent Behaviors: From Local Interactions to Collective Intelligence", *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, Jean-Arcady Meyer, Herbert L. Roitblat, and Stewart W. Wilson, editors, 1993, pp. 432-441.

[Mataric, 1994] Maja J. Mataric, "Learning to Behave Socially", *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, David Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors, 1994, pp. 453-462.

[Mataric, 1995] Maja J. Mataric, "Issues and approaches in the design of collective autonomous agents", *Robotics and Autonomous Systems*, 1995, Volume 16, pp. 321-331.

[Mayer and Wood, 1986] Gordon Mayer and Elaine Ide Wood, "Multiple-Arm Control and Assembly Operation", *The Journal of Intelligent Machines*, Robotics Engineering, April 1986, Volume 8, Number 4, pp. 18-25.

[Mayfield et al., 1995] James Mayfield, Yannis Labrou, and Tim Finin, "Evaluation of KQML as an Agent Communication Language", *Intelligent Agents II*, Agent Theories, Architectures, and Languages, M. Wooldridge, J. P. Muller, and M. Tambe, editors, August 1995, pp. 408-431.

[Mazouz and Han, 1989] A. Kader Mazouz and Chingping Han, "Flexible Automated Material Handling System", *Journal of Mechanical Working Technology*, 1989, Volume 20, pp. 433-440.

[McFarland, 1994] David McFarland, "Towards Robot Cooperation", *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, David Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors, 1994, pp. 440-444.

[Meier, 1987] Claire D. Meier, "Command Verification: An Appraisal System for Mobile Robot Task Commands", *Robotics and Expert Systems*, June 1987, Volume 3, pp. 245-250.

[Merat et al., 1997] Frank L. Merat, Nicholas A. Barendt, Roger D. Quinn, Greg C. Causey, Wyatt S. Newman, Virgilio B. Velasco Jr., Andy Podgurski, Yoohwan Kim, Gultekin Ozsoyoglu, and Ju-Yeon Jo, "Advances in Agile Manufacturing", *Proceedings of the International Conference on Robotics and Automation*, April 1997, pp. 1216-1222.

[Meyer et al., 1988] Wolfgang Meyer, Randolf Isenberg, and Martin Hübner, "Knowledge-based factory supervision – The CIM shell", *International Journal of Computer Integrated Manufacturing*, 1988, Volume 1, Number 1, pp. 31-43.

[Mina, 1987] I. Mina, "KMPR: An Experimental Knowledge-Based Modeling Prototype for Robots", *IEEE International Conference on Robotics and Automation*, March 1987, pp. 2011-2020.

[Minsky, 1985] Marvin Minsky, "A Framework for Representing Knowledge", *Readings in Knowledge Representation*, R. J. Brachman and H. J. Levesque, editors, Morgan Kaufmann Publishers, 1985, pp. 246-262.

[Missiaen, 1991] Lode Missiaen, "Localized Abductive Planning for Robot Assembly", *Proceedings of the International Conference on Robotics and Automation*, April 1991, pp. 605-610.

[Mitrani, 1982] I. Mitrani, *Simulation techniques for discrete event systems*, Cambridge University Press, 1982.

[Miura and Ikeuchi, 1995] Jun Miura and Katsushi Ikeuchi, "Generating Visual Sensing Strategies in Assembly Tasks", *IEEE International Conference on Robotics and Automation*, May 1995, Volume 2, pp. 1912-1918.

[Mochizuki et al., 1987] Jun Mochizuki, Michio Takahashi, and Seiji Hata, "Unpositioned Workpieces Handling Robot with Visual and Force Electronics," *IEEE Transactions on Industrial Electronics*, February 1987, Volume 34, Number 1, pp. 1-4.

[Morrow and Khosla, 1995] J. Daniel Morrow and Pradeep K. Khosla, "Sensorimotor Primitives for Robotic Assembly Skills", *Proceedings of IEEE International Conference on Robotics and Automation*, May 1995 Volume 2, pp. 1894-1899.

[Morrow and Khosla, 1997] J. Daniel Morrow and Pradeep K. Khosla, "Manipulation Task Primitives for Composing Robot Skills", *Proceedings of the International Conference on Robotics and Automation*, April 1997, pp. 3354-3359.

[Müller, 1996] Jörg P. Müller, *An Architecture for Dynamically Interacting Agents*, Doctoral Dissertation, Der Technischen Fakultät der Universität des Saarlandes, 1996.

[Musser, 1995] David R. Musser, *Rationale for Adding Hash Tables to the C++ Standard Template Library*, February 20, 1995, available by anonymous ftp from ftp.cs.rpi.edu as pub/stl/hashrationale.ps.

[Nagata et al., 1988] Tadashi Nagata, Kunihiko Honda, and Yoshiaki Teramoto, "Multirobot Plan Generation in a Continuous Domain: Planning by Use of Plan Graph and Avoiding Collisions Among Robots", *IEEE Journal of Robotics and Automation*, February 1988, Volume 4, Number 1, pp. 2-13.

[Najjari and Steiner, 1996] H. Najjari and S. J. Steiner, "An Intelligent Software System for a Robotics Assembly Cell", *Journal of Materials Processing Technology*, 1996, Volume 61, pp. 18-23.

[Nakamura and Xu, 1988] Yoshihiko Nakamura and Yingti Xu, "An Architecture of Intelligent Controller for Multi-Sensor Robotic Systems", *Proceedings of the International Symposium and Exposition on Robots*, November 1988, pp. 550-592.

[Neches et al., 1991] Robert Neches, Richard Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator, and William S. Swartout, "Enabling Technology for Knowledge Sharing", *AI Magazine*, Fall 1991, Volume 12, Number 3, pp. 36-56.

[Neves and Gray, 1995] C. Neves and J. O. Gray, "System Architectures for Advanced Robotics", *The Design and Development of Autonomous Agents*, IEE Computing and Control Division, November 1995, Digest Number 95/211, pp. 6/1 to 6/8.

[Nevins and Whitney, 1977] J. L. Nevins and D. E. Whitney, "Research on Advanced Assembly Automation", *Computer*, December 1977, pp. 24-38.

[Nevins and Whitney, 1978] J. L. Nevins and D. E. Whitney, "Computer-controlled Assembly", *Scientific American*, February 1978, pp. 62-74.

[Nevins and Whitney, 1980] J. L. Nevins and D. E. Whitney, "Assembly Research", *Automatica*, 1980, Volume 16, pp. 595-613.

[Nevins et al., 1989] James L. Nevins, Daniel E. Whitney, Thomas L. De Fazio, Alexander C. Edsall, Richard E. Gustavson, Richard W. Metzinger, and William A. Dvorak, *Concurrent Design of Products and Processes: A Strategy for the Next Generation in Manufacturing*, McGraw-Hill Publishing Company, 1989.

[Nilsson, 1969] N. G. Nilsson, "A mobile automaton: An application of artificial intelligence techniques", *Proceedings of 1st International Joint Conference on AI*, 1969, pp. 509-520.

[Nnaji, 1993] B. O. Nnaji, *Theory of Automatic Robot Assembly and Programming*, Chapman & Hall, 1993.

[Noaker, 1994] Paula M. Noaker, "The Search for Agile Manufacturing", *Manufacturing Engineering*, 1994, Volume 113, Number 5, pp. 40-43.

[Nof and Hanna, 1989] S. Y. Nof and D. Hanna, "Operational characteristics of multi-robot systems with cooperation", *International Journal of Production Research*, 1989, Volume 27, Number 3, pp. 477-492.

[Nof and Rajan, 1993] Shimon Y. Nof and N. Rajan, "Automatic Generation of Assembly Constraints and Cooperation Task Planning", *Manufacturing Technology*, CIRP Annals 1993, Volume 42, Number 1, pp. 13-16.

[Nolfi et al., 1994] Stefano Nolfi, Dario Floreano, Orazio Miglino, and Francesco Mondada, "How to Evolve Autonomous Robots: Different Approaches in Evolutionary Robotics", *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, Rodney A. Brooks and Pattie Maes, editors, 1994, pp. 190-197.

[Noorhosseini and Malowany, 1995] S. M. Noorhosseini and A. S. Malowany, "State Matrix Representation of Assembly and Robot Planning", *Robotica*, 1995, Volume 13, pp. 259-272.

[Noreils, 1993] Fabrice R. Noreils, "Toward a Robot Architecture Integrating Cooperation between Mobile Robots: Application to Indoor Environment", *The International Journal of Robotics Research*, February 1993, Volume 12, Number 1, pp. 79-98.

[Numaoka and Takeuchi, 1993] Chisato Numaoka and Akikazu Takeuchi, "Collective Choice of Strategic Type", *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, Jean-Arcady Meyer, Herbert L. Roitblat, and Stewart W. Wilson, editors, 1993, pp. 469-477.

[Nwana, 1996] Hyacinth S. Nwana, "Software Agents: An Overview", *The Knowledge Engineering Review*, 1996, Volume 11, Number 3, pp. 1-40.

[O'Grady and Lee, 1990] Peter O'Grady and Kwan H. Lee, "A Hybrid Actor and Blackboard Approach to Manufacturing Cell Control", *Journal of Intelligent and Robotic Systems*, 1990, Volume 3, pp. 67-72.

[O'Grady et al., 1987] P. J. O'Grady, H. Bao, and K. H. Lee, "Issues in Intelligent Cell Control for Flexible Manufacturing Systems", *Computers in Industry*, Volume 9, Number 1, pp. 25-36.

[O'Hare, 1990] G. M. P. O'Hare, "Designing Intelligent Manufacturing Systems: A Distributed Artificial Intelligence Approach", *Computers in Industry*, 1990, Volume 15, Number 1, pp. 17-25.

[Oliveira, 1994] Eugénio Oliveira, "Cooperative Multi-agent System for an Assembly Robotics Cell", *Robotics & Computer-Integrated Manufacturing*, 1994, Volume 11, Number 4, pp. 311-317.

[Oliveira and Ramos, 1996] Eugénio Oliveira and Carlos Ramos, "A cooperative multi-agent system for an assembly robotic cell", *Advanced Robotics*, 1996, Volume 10, Number 1, pp. 15-50.

[Oliveira and Rocha, 1994] Eugénio Oliveira and Ana Paula Rocha, "Negotiation and Conflict Resolution in a Multi-Agent System for an Assembly Robotics Cell", *IEEE International Conference on Systems, Man, and Cybernetics*, Humans, Information and Technology, October 1994, pp. 605-611.

[Oliveira et al., 1991] Eugénio Oliveira, R. Camacho, and C. Ramos, "A multi-agent environment in robotics", *Robotica*, 1991, Volume 9, pp. 431-440.

[Osato, 1993] Nobuyasu Osato, "An Action Interpreter of a Robot Control Agent", *Proceedings of the IEEE/RSJ*

International Conference on Intelligent Robots and Systems, July 1993, pp. 1126-1133..

[Ota et al., 1995] Jun Ota, Tamio Arai, Eiichi Yoshida, Daisuke Kurabayashi, and Tomoyuki Mori, "Real Time Planning for Multiple Mobile Robots", *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, August 1995, pp. 406-411.

[Pan and Tenenbaum, 1991] Jeff Y. C. Pan and Jay M. Tenenbaum, "An Intelligent Agent Framework for Enterprise Integration", *IEEE Transactions on Systems, Man, and Cybernetics*, November/December 1991, Volume 21, Number 6, pp. 1391-1408.

[Panceralla et al., 1995] Carmen M. Pancerella, Andrew J. Hazelton, and H. Robert Frost, "An autonomous agent for on-machine acceptance of machined components", *Proceedings of Modeling, Simulation, and Control Technologies for Manufacturing*, SPIE's International Symposium on Intelligent Systems and Advanced Manufacturing, October 1995.

[Pant et al., 1994] Somendra Pant, Laurie Rattner, and Cheng Hsu, "Manufacturing Information Integration Using a Reference Model", *International Journal of Operations & Production Management*, 1994, Volume 14, Number 11, pp. 52-72.

[Pao and Jelinek, 1988] Yoh-Han Pao and Mariann Jelinek, "Flexible Manufacturing Cells and Systems", *International Encyclopedia of Robotics: Applications and Automation*, Richard D. Dorf, Editor-in-Chief, 1988, Volume 1, pp. 530-551.

[Pape, 1990] Claude Le Pape, "A Combination of Centralized and Distributed Methods for Multi-Agent Planning and Scheduling", *IEEE International Conference on Robotics and Automation*, May 1990, pp. 488-493.

[Park and Cho, 1993] Yong K. Park and Hyung S. Cho, "A Fuzzy Rule-based Assembly Algorithm for Precision Parts Mating", *Mechatronics: Mechanics ~ Electronics ~ Control*, Volume 3, Number 4, pp. 433-450.

[Park and Chung, 1993] Jong Hun Park and Myung Jin Chung, "Automatic Generation of Assembly Sequences for Multi-Robot Workcell", *Robotics & Computer-Integrated Manufacturing*, 1993, Volume 10, Numbers 5, pp. 355-363.

[Parker, 1993] Lynne E. Parker, "Adaptive Action Selection for Cooperative Agent Teams", *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, Jean-Arcady Meyer, Herbert L. Roitblat, and Stewart W. Wilson, editors, 1993, pp. 442-450.

[Parker, 1994] Lynne E. Parker, "An Experiment in Mobile Robotic Cooperation", *Proceedings of the ASCE Specialty Conference on Robotics for Challenging Environments*, 1994, pp. 131-139.

[Parthasarathy and Kim, 1990] Sanjay Parthasarathy and Steven H. Kim, "Manufacturing systems: parallel system models and some theoretical results", *International Journal of Computer Applications in Technology*, 1990, Volume 3, Number 4, pp. 225-238.

[Parunak, 1985] H. Van Dyke Parunak, "Manufacturing Experience with the Contract Net", *Distributed Artificial Intelligence*, Michael H. Huhns, editor, 1985, pp. 285-310.

[Parunak, 1990] H. Van Dyke Parunak, "Distributed AI and Manufacturing Control: Some Issues and Insights", *Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Yves Demazeau and Jean Pierre Müller, editors, Elsevier Science Publishers, August 1989, pp. 81-101.

[Parunak, 1994] H. Van Dyke Parunak, "Deploying Autonomous Agents on the Shop Floor: A Preliminary Report", *Papers from the Thirteenth International Workshop on Distributed Artificial Intelligence*, July 1994, pp. 259-261.

[Parunak et al., 1997] H. Van Dyke Parunak, Albert D. Baker, and Steven J. Clark, "The AARIA Agent Architecture: An Example of Requirements-Driven Agent-Based System Design", *Agents*, Marina del Ray, 1997.

[Patarinski and Botev, 1993] Simeon P. Patarinski and Roumen G. Botev, "Robot Force Control: A Review", *Mechatronics: Mechanics ~ Electronics ~ Control*, Volume 3, Number 4, pp. 377-398.

[Patil et al., 1994] Ramesh S. Patel, Richard E. Fikes, Peter F. Patel-Schneider, Don Mckay, Tim Finin, Thomas Gruber, and Robert Neches, "The DARPA Knowledge Sharing Effort: Progress Report", *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, 1992..

[Pelagagge et al., 1995] Pacifico M. Pelagagge, Gino Cardarelli, and Mario Palumbo, "Design Criteria for Cooperating Robots Assembly Cells", *Journal of Manufacturing Systems*, 1995, Volume 14, Number 4, pp. 219-229.

[Pelagagge et al., 1996] Pacifico M. Pelagagge, Gino Cardarelli, and Mario Palumbo, "Some Criteria to Help the Experimental Setup of Assembly Cells with Cooperating Robots", *Robotics & Computer-Integrated Manufacturing*, 1996, Volume 12, Numbers 2, pp. 125-133.

[Petriu et al., 1993] Dorina C. Petriu, Colin C. Archibald, and Emil M. Petriu, "Petri Net Model of a Resource Management System for a Multi-Robot Assembly Cell", *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 1993, Volume 5, pp. 409-414.

[Petriu et al., 1994] Dorina C. Petriu, Thomas R. Jones, and Emil M. Petriu, "Communicating State-Machine Model of Resource Management in a Multi-Robot Assembly Cell", *ISCA International Journal of Computers and Their Applications*, August 1994, Volume 1, Number 1, pp. 43-48.

[Petriu et al., 1996] E. M. Petriu, D. C. Petriu, and J. S. Basran, "Modeling and Verification of a Multirobot Assembly System." *Proceedings of the Third AMAST Workshop on Real-time Systems*, March 1996, pp. 175-189.

[Petropoulakis and Malcolm, 1990] Lykourgos Petropoulakis and Chris Malcolm, "Programming Autonomous Assembly Agents: Functionality and Robustness", Department of Artificial Intelligence, *University of Edinburgh*, 1990, Research Paper Number 468.

[Pfeiffer, 1996] Friedrich Pfeiffer, "Assembly processes with robotic systems", *Robotics and Autonomous Systems*, 1996, Volume 19, pp. 151-166.

[Picardat, 1986] Jean-François Picardat, "An A.I. robot programming architecture: MUCAR", *Proceedings of the 16th International Symposium on Industrial Robots*, October 1986, pp. 765-774.

[Pooch and Wall, 1993] Udo W. Pooch and James A. Wall, *Discrete Event Simulation: A Practical Approach*, CRC Press, 1993.

[Popplestone et al., 1978] R. J. Popplestone, A. P. Ambler, and I. Bellos, "RAPT: A language for describing assemblies", *Industrial Robot*, September 1978, Volume 15, Number 3, pp. 131-137.

[Popplestone et al., 1980] R. J. Popplestone, A. P. Ambler, and I. M. Bellos, "An Interpreter for a Language for Describing Assemblies", *Artificial Intelligence*, 1980, Volume 14, pp. 79-107.

[Popplestone et al., 1990] Robin J. Popplestone, Yanxi Liu, and Rich Weiss, "A Group Theoretic Approach to Assembly Planning", *AI Magazine*, Spring 1990, Volume 11, Number 1, pp. 82-97.

[Premvuti and Yuta, 1992] Suparerk Premvuti and Shinichi Yuta, "An Experiment in Realizing Autonomous and Cooperative Behaviours between Multiple Mobile Robots", *IEEE International Workshop on Emerging Technologies and Factory Automation*, August 1992, pp. 301-304.

[Qiao et al., 1993] H. Qiao, B. S. Dalay, and R. M. Parkin, "Robotic peg-hole insertion operations using a six-component force sensor", *Journal of Mechanical Engineering Science*, Part C, Volume 207, pp. 289-306.

[Rabelo and Camarinha-Matos, 1994] Ricardo J. Rabelo and L. M. Camarinha-Matos, "Negotiation in Multi-Agent based Dynamic Scheduling", *Robotics & Computer-Integrated Manufacturing*, 1994, Volume 11, Number 4, pp. 303-309.

[Raczkowsky and Seucken, 1991] Jörg Raczkowksy and Kerstin Seucken, "Sensor Planning for the Error Diagnosis of Robot Assembly Tasks", *Expert Systems and Robotics*, T. Jordanides and B. Torby, editors, 1991, NATO ASI Series, Volume F71, pp. 127-134.

[Raibert and Craig, 1981] M. H. Raibert and J. J. Craig, "Hybrid Position/Force Control of Manipulators", *Transactions of the ASME*, June 1981, Volume 102, pp. 126-133.

[Rajan, 1994] Venkat N. Rajan, "Assembly Product and Cooperating Multi-Robot Cell Design based on Constraints Interaction Analysis", *Flexible Assembly Systems*, The Design Engineering Division, ASME 1994, Volume 73, pp. 107-116.

[Ramos, 1992] Carlos Ramos, "Closing the Loop of Task Planning, Action and Sensing", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, July 1992, pp. 909-916.

[Ramos, 1995] Carlos Ramos, "Task Negotiation for Distributed Manufacturing Systems", *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, August 1995, pp. 259-264.

[Ramos and Oliveira, 1992] Carlos Ramos and Eugénio Oliveira, "Sensor-Based Reactive Planning and Execution for Robotic Assembly Tasks", *IEEE International Conference on Systems Engineering*, September 1992, pp. 135-138.

[Ramos and Oliveira, 1993] Carlos Ramos and Eugénio Oliveira, "CIARC: A Multi-Agent Community for Intelligent Assembly Robotic Systems with Real-Time Capabilities", *IEEE International Conference on Systems, Man, and Cybernetics*, Humans, Information and Technology, 1993, pp. 617-622.

[Rampersad, 1995a] Hubert K. Rampersad, "State of the art in robotic assembly", *Industrial Robot*, 1995, Volume 22, Number 2, pp. 10-13.

[Rampersad, 1995b] Hubert K. Rampersad, "A Case Study in the Design of Flexible Assembly Systems", *The International Journal of Flexible Manufacturing Systems*, 1995, Volume 7, pp. 255-286.

[Rampersad, 1996] Hubert K. Rampersad, "Robotic assembly system design for total productivity", *International Journal of Production Research*, 1996, Volume 34, Number 1, pp. 71-93.

[Ranky, 1983] Paul Ranky, *The Design and Operation of FMS*, IFS Publications, 1983.

[Ranky, 1986a] Paul G. Ranky, *Computer Integrated Manufacturing*, Prentice-Hall, 1986.

[Ranky, 1986b] Paul G. Ranky, "Program Prospectus for the Simulation, Design and Implementation of a Flexible Assembly and Inspection Cell", *Center for Research on Integrated Manufacturing*, The University of Michigan, February 1986, pp. 1-29.

[Ranky, 1991] Paul G. Ranky, "Flexible Robot Work Cell Design by Simulation", *Intelligent Robotic Systems*, Spyros G. Tzafestas, editor, Marcel Dekker, 1991, pp. 673-713.

[Ranky, 1994] Paul G. Ranky, *Concurrent / Simultaneous Engineering (Methods, Tools & Case Studies*, CIMware Limited, 1994.

[Rao et al., 1996] Anil Rao, David J. Kriegman, and Kenneth Y. Goldberg, "Complete Algorithms for Feeding Polyhedral Parts Using Pivot Grasps", *IEEE Transactions on Robotics and Automation*, April 1996, Volume 12, Number 2, pp. 331-342.

[Rathmill, 1985] Keith Rathmill, 1985, *Robotic Assembly*, International Trends in Manufacturing Technology, IFS Publications, 1985.

[RATIONAL, 1997] *Unified Modeling Language: Notation Guide*, RATIONAL Software Corporation, Version 1.0, January 1997, "http://www.rational.com/uml".

[Rausch et al., 1995] Alexander Rausch, Norbert Oswald, and Paul Levi, "Cooperative crossing of traffic intersections in a distributed robot system", *Sensor Fusion and Networked Robotics VIII*, SPIE proceedings, October 1995, Volume 2589, pp. 218-229.

[Ravani, 1988] Bahram Ravani, editor, *CAD Based Programming for Sensor Robots*, Springer-Verlag, 1988, NATO ASI Series, Volume F50.

[Reinhardt, 1994] Andy Reinhardt, "The Network with Smarts", *Byte*, October 1994, pp. 51-64.

[Rembold and Hörmann, 1987] Ulrich Rembold and Klaus Hörmann, editors, *Languages for Sensor-Based Control in Robotics*, Springer-Verlag, 1987.

[Rembold and Soetadji, 1987] U. Rembold and T. Soetadji, "The Development of an Autonomous Assembly Robot", *Robotics & Computer-Integrated Manufacturing*, 1987, Volume 3, Numbers 1, pp. 23-37.

[Rizzi et al., 1997] Alfred A. Rizzi, Jay Gowdy, and Ralph L. Hollis, "Agile Assembly Architecture: An Agent Based Approach to Modular Precision Assembly Systems", *Proceedings of the International Conference on Robotics and Automation*, April 1997, pp. 1511-1516.

[Roach and Boaz, 1987] John W. Roach and Michael N. Boaz, "Coordinating the Motions of Robot Arms in a Common Workspace", *IEEE Journal of Robotics and Automation*, October 1987, Volume 3, Number 5, pp. 437-444.

[Rodighiero and Canciani, 1987] F. Rodighiero and A. Canciani, "An Experience in Task Level Robot Programming", *IEEE Workshop on Languages for Automation*, 1987, pp. 86-89.

[Rogers and Williams, 1988] P. Rogers, and D. J. Williams, "Knowledge-based control in manufacturing and automation", *International Journal of Computer Integrated Manufacturing*, 1988, Volume 1, Number 1, pp. 21-30.

[Rosenschein, 1992] Jeffrey S. Rosenschein, "Teaching Distributed Artificial Intelligence", *Distributed Artificial Intelligence: Theory and Praxis*, Nicholas M. Avouris and Les Gasser, editors, Kluwer Academic Publishers, 1992, pp. 215-227.

[Ross, 1994] E. M. Ross, "Flexible Parts Feeders", *Assembly*, October 1994, Volume 37, Number 9, pp. 24-28.

[Rowland and Nicholls, 1995] J. J. Rowland and H. R. Nicholls, "A virtual sensor implementation for a flexible assembly machine", *Robotica*, 1995, Volume 13, pp. 195-199.

[Rubin and Dilworth, 1992] Stuart H. Rubin and David S. Dilworth, "On Automating Goal-to-Task Translation in a Futuristic Robotic Factory", *Artificial Intelligence for Engineering, Design, and Manufacturing*, ISA 1992, Volume 31, Number 2, pp. 135-150.

[Rumbaugh, 1995] James Rumbaugh, "OMT: The object model", *Journal of Object-Oriented Programming*, January 1995, Volume 7, Number 8, pp. 21-27.

[Russell and Norvig, 1995] S. J. Russell and P. Norvig, *Artificial Intelligence - A Modern Approach*, Prentice-Hall, 1995.

[Russell et al., 1989] G. T. Russell, J. T. Herd, N. D. Duffy, J. B. C. Davies, and W. J. Finlay, "An integrated control and error interpretation system for collaborating robots", *International Journal of Production Research*, 1989, Volume 27, Number 3, pp. 493-514.

[Saad et al., 1995] A. Saad, K. Kawamura, G. Biswas, M. E. Johnson, and A. Salama, "Evaluating a Contract Net-Based Heterarchical Scheduling Approach for Flexible Manufacturing", *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, August 1995, pp. 147-152.

[Sakane et al., 1993] Shigeyuki Sakane, Tomomasa Sato, Hiroaki Okoshi, and Masayoshi Kakikura, "Distributed Sensing System with 3D Model-Based Agents", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, July 1993, pp. 1157-1163.

[Sanderson et al., 1990] Arthur C. Sanderson, Luiz S. Homem de Mello, and Hui Zhang, "Assembly Sequence Planning", *AI Magazine*, Spring 1990, Volume 11, Number 1, pp. 62-81.

[Sato and Maciejewski, 1994] Akihiro Sato and Anthony A. Maciejewski, "A Virtual Object Manipulation Interface for Automated Assembly Programming", *IEEE International Conference on Systems, Man, and Cybernetics*, Humans, Information and Technology, October 1994, pp. 1826-1831.

[Saveriano, 1988] Jerry W. Saveriano, "Pioneers of Robotics", *International Encyclopedia of Robotics: Applications and Automation*, Richard C. Dorf, editor in chief, Volume 2, pp. 1078-1092.

[Schneider, 1989] Stanley Aaron Schneider, *Experiments in the Dynamic and Strategic Control of Cooperating Manipulators*, Doctoral Dissertation, Stanford University, September, 1989.

[Sedgewick, 1983] R. Sedgewick, *Algorithms*, Addison-Wesley Publishing Company, Chapter 16, 1983.

[Segre, 1988] Alberto Maria Segre, *Machine Learning of Robot Assembly Plans*, Doctoral Dissertation, Cornell University, Kluwer Academic Publishers, 1988.

[Segre, 1991] Alberto Segre, "Learning how to plan", *Robotics and Autonomous Systems*, 1991, Volume 8, pp. 93-111.

[Selke and Pugh, 1986] K. Selke and A. Pugh, "Sensor-guided generic assembly", *Proceedings of the 6th International Conference on Robot Vision and Sensory Controls*, M. Briot, editor, June 1986, pp. 11-20.

[Selke et al., 1987] K. Selke, K. G. Swift, G. E. Taylor, A. Pugh, S. N. Davey, and G. E. Deacon, "Knowledge-based robotic assembly — a step further towards flexibility", *Computer Aided Engineering Journal*, February 1987, Volume 4, Number 1, pp. 62-67.

[Selke et al., 1991] Klaus K. Selke, Helen C. Shen, Graham E. Deacon, and A. Pugh, "A strategy for sensors and rules in flexible robotic assembly", *International Journal of Production Research*, 1991, Volume 29, Number 2, pp. 277-291.

[Shapiro, 1992] Stuart C. Shapiro, Editor in Chief, [1992], "Manufacturing, AI in", *Encyclopedia of Artificial Intelligence*, John Wiley & Sons, Second Edition, Volume 2, pp. 903-9.

[Shaum, 1986] Loren Shaum, "Robotic Work Cell enhanced by Programmable Part Presentation System", *Proceedings of the Second International IMS Conference*, October 1986, pp. 270-282.

[Shaw, 1987] Michael J. Shaw, "A distributed scheduling method for computer integrated manufacturing: the use of local area networks in cellular systems", *International Journal of Production Research*, 1987, Volume 25, Number 9, pp. 1285-1303.

[Shaw and Whinston, 1985] Michael J. Shaw and Andrew B. Whinston, "Task Bidding and Distributed Planning in Flexible Manufacturing", *The 2nd Conference on Artificial Intelligence Applications*, December 1985, pp. 184-189.

[Shibata and Fukuda, 1993] Takanori Shibata and Toshio Fukuda, "Coordinative Behavior by Genetic Algorithm and Fuzzy in Evolutionary Multi-Agent System", *Proceedings of the International Conference on Robotics and Automation*, May 1993, Volume 1, pp. 760-765.

[Shih and Srihari, 1995] Wurong Shih and K. Srihari, "Distributed Artificial Intelligence in Manufacturing Systems Control", *Proceedings of the 17th International Conference on Computers and Industrial Engineering*, 1995, pp. 199-203.

[Shih et al., 1991] Ching-Long Shih, Peter J. Sadler, and William A. Gruver, "Collision Avoidance for Two SCARA Robots", *Proceedings of the International Conference on Robotics and Automation*, April 1991, pp. 674-679.

[Shin and Epstein, 1987] Kang K. Shin and Mark E. Epstein, "Intertask Communication in an Integrated Multirobot System", *IEEE Journal of Robotics and Automation*, April 1987, Volume 3, Number 2, pp. 90-100.

[SIAM, 1997] Sandia Intelligent Agents for Manufacturing, "http://nittany.ca.sandia.gov:8001/", Web Site, 1997.

[Sikora and Shaw, 1994] Riyaz Sikora and Michael J. Shaw, "Manufacturing Information Coordination and System

Integration By A Multi-Agent Framework", *Papers from the Thirteenth International Workshop on Distributed Artificial Intelligence*, July 1994, pp. 314-340.

[Smith, 1979] Reid G. Smith, *A Framework for Distributed Problem Solving*, Doctoral Dissertation, Stanford University, UMI Research Press, 1979.

[Smith, 1980] Reid G. Smith, "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", *IEEE Transactions on Computers*, 1980, Volume 29, Number 12, pp. 1104-1113.

[Smith and Davis, 1981] Reid G. Smith and Randall Davis, "Frameworks for Cooperation in Distributed Problem Solving", *IEEE Transactions on Systems, Man, and Cybernetics*, January 1981, Volume 11, Number 1, pp. 61-70.

[Smithers, 1994] Tim Smithers, "On Why Better Robots Make it Harder", *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, David Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors, 1994, pp. 64-72.

[Smithers and Malcolm, 1987] Tim Smithers and Chris Malcolm, "A Behavioral Approach to Robot Task Planning and Off-line Programming", *First Workshop on Manipulators, Sensors, and Steps toward Mobility*, May 1987, pp. 313-326.

[Spiegel, 1975] Murray R. Spiegel, *Theory and Problems of Probability and Statistics*, Schaum's Outline Series, 1975.

[Spur and Specht, 1987] G. Spur and D. Specht, "Computer Integrated Manufacturing in Future Factories", *Robotics & Computer-Integrated Manufacturing*, 1987, Volume 3, Number 2, pp. 147-155.

[Sridharan, 1987] N. S. Sridharan, "1986 Workshop on Distributed AI", *AI Magazine*, Fall 1987, Volume 8, Number 3, pp. 75-85.

[Sriram and Candadai, 1996] Ram Sriram and Arun Candadai, "Agile Infrastructure for Manufacturing Systems (AIMS) - A Pilot Program", *Fifth National Agility Conference*, March 1996.

[Steels, 1994a] Luc Steels, "A case study in the behavior-oriented design of autonomous agents", *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, David Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors, 1994, pp. 445-452.

[Steels, 1994b] Luc Steels, "Emergent Functionality in robotic agents through on-line evolution", *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, Rodney A. Brooks and Pattie Maes, editors, 1994, pp. 445-452.

[Steels, 1995] Luc Steels, "When are robots intelligent autonomous agents?", *Robotics and Autonomous Systems*, 1995, Volume 15, pp. 3-9.

[Stepanov and Lee, 1995] Alexander Stepanov and Meng Lee, *The Standard Template Library*, Hewlett-Packard Laboratories, 1995.

[Stevenson, 1987] C. N. Stevenson, "Distributed Control of a Robot-based Assembly Cell", *The Winter Annual Meeting of the American Society of Mechanical Engineers*, Intelligent and Integrated Manufacturing

Analysis and Synthesis, December 1987, pp. 333-340.

[Strip, 1989] David R. Strip, "Primitives for Robotic Mechanical Assembly: Force Directed Insertions", *Robotics & Computer-Integrated Manufacturing*, 1989, Volume 6, Number 4, pp. 283-286.

[Suehiro and Kitagaki, 1995] Takashi Suehiro and Kosei Kitagaki, "A Multi-agent Based Implementation of Task Coordinate Servo for the DD Manipulator:ETA3", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, August 1995, pp. 459-464.

[Suehiro and Kitagaki, 1996] Takashi Suehiro and Kosei Kitagaki, "Multi-Agent Based Implementation of Robot Skills", *Proceedings of the International Conference on Robotics and Automation*, April 1996, Volume 4, pp. 2976-2981.

[Swyt, 1989] Dennis A. Swyt, "AI in Manufacturing: The NBS*AMRF as an Intelligent Machine", *Robotics and Autonomous Systems*, 1989, Volume 4, pp. 327-332.

[Taylor and Grossman, 1983] Russell H. Taylor and David D. Grossman, "An Integrated Robot System Architecture", *Proceedings of the IEEE*, Volume 71, Number 7, pp. 842-856.

[Tchako et al., 1994] J. F. N. Tchako, B. Beldjilali, D. Trentesaux, and C. Tahon, "Modelling with coloured timed Petri nets and simulation of dynamic and distributed management system for a manufacturing cell", *International Journal of Computer Integrated Manufacturing*, 1994, Volume 7, Number 6, pp. 323-339.

[Temmes et al., 1986] J. Temmes, P. Ruusunen, and J. Lempiäinen, "Assembly cell controls using flexible software tools and high level control configuration language", *Proceedings of the 5th International Conference on Flexible Manufacturing Systems*, November 1986, pp. 53-62.

[Thomas et al., 1993] M. C. Thomas, M. Occello, and J. Y. Tigli, and H. Medromi, "Real Time Constraints for Decision Making in Mobile Robotics", *IEEE International Conference on Systems, Man, and Cybernetics*, Systems Engineering in the Service of Humans, October 1993, pp. 640-645.

[Tigli and Thomas, 1994] J. Y. Tigli and M. C. Thomas, "Use of Multi Agent Systems for Mobile Robotics Control", *IEEE International Conference on Systems, Man, and Cybernetics*, Humans, Information and Technology, October 1994, pp. 588-592.

[Toguyeni et al., 1995] A. K. A. Toguyeni, E. Craye, and J. C. Gentina, "Knowledge-based Supervision of Flexible Manufacturing Systems", *Artificial Intelligence in Industrial Decision Making, Control and Automation*, S. G. Tzafestas and H. B. Verbruggen, editors, Kluwer Academic Publishers, 1995, pp. 631-662.

[Tönshoff et al., 1992] H. K. Tönshoff, E. Menzel, and H. S. Park, "A Knowledge-Based System for Automated Assembly Planning", *Manufacturing Technology*, CIRP Annals 1992, Volume 41, Number 1, pp. 19-2 (briefly discuss task planning).

[Toukal et al., 1995] Zakaria Toukal, Yacine Amirat, Said Babaci, and Jean Pontnau, "Programming Language for Robotic Assembly Tasks", *International IEEE/IAS Conference on Industrial Automation and Control: Emerging Technologies*, May 1995, pp. 612-619.

[Tracy et al., 1994a] Michael J. Tracey, James N. Murphy, Robert W. Denner, Bruce W. Pince, F. Robert Joseph, Allen R. Pilz, and Michael B. Thompson, "Achieving agile manufacturing in the automotive industry",

Automotive Engineering, November 1994, Volume 102, Number 11, pp. 19-24.

[Tracy et al., 1994b] Michael J. Tracey, James N. Murphy, Robert W. Denner, Bruce W. Pince, F. Robert Joseph, Allen R. Pilz, and Michael B. Thompson, "Achieving agile manufacturing: Part II", *Automotive Engineering*, December 1994, Volume 102, Number 12, pp. 13-17.

[TRIO, 1997] *Mobile Agents: Basics, Technology and Applications*, Seminar Program, Telecommunications Research Institute of Ontario, March 1997.

[Tsuda and Takahashi, 1995] Masayuki Tsuda and Tomoichi Takahashi, "A Method for Changing Contact States for Robotic Assembly by using Some Local Models in a Multi-Agent System", *Proceedings of the International Conference on Robotics and Automation*, May 1995, Volume 3, pp. 2713-2719.

[Tsukada and Shin, 1994] Thomas Tsukada and Kang G. Shin, "Polite Rescheduling: Responding to Local Schedule Disruptions in Distributed Manufacturing Systems", *IEEE International Conference on Robotics and Automation*, May 1994, pp. 1986-1991.

[Tsukada and Shin, 1996] Thomas Tsukada and Kang G. Shin, "PRIAM: Polite Rescheduler for Intelligent Automated Manufacturing", *IEEE Transactions on Robotics and Automation*, April 1996, Volume 12, Number 2, pp. 235-245.

[Tung and Kak, 1994] C. P. Tung and A. C. Kak, "Integrated Sensing, Task Planning and Execution", *Proceedings of IEEE International Conference on Robotics and Automation*, May 1994, pp. 2030-2037.

[Tung and Kak, 1996] Chao-Ping Tung and Avinash C. Kak, "Integrated Sensing, Task Planning, and Execution of Robotic Assembly", *IEEE Transactions on Robotics and Automation*, April 1996, Volume 12, Number 2, pp. 187-201.

[Tzafestas, 1990] Spyros G. Tzafestas, "AI Techniques in Computer-Aided Manufacturing Systems", *Knowledge Engineering: Applications*, Hojjat Adeli, editor, Volume II, pp. 161-212.

[Tzafestas, 1994] Elpida S. Tzafestas, "Agentifying the Process: Task-Based or Robot-Based Decomposition?", *IEEE International Conference on Systems, Man, and Cybernetics*, Humans, Information and Technology, October 1994, pp. 582-587.

[Tzafestas and Tzafestas, 1991] Spyros Tzafestas and Elpida Tzafestas, "The Blackboard Architecture in Knowledge-Based Robotic Systems", *Expert Systems and Robotics*, T. Jordanides and B. Torby, editors, 1991, NATO ASI Series, Volume F71, pp. 285-317.

[Tzafestas and Tzafestas, 1995] Elpida S. Tzafestas and Spyros G. Tzafestas, "Incremental Design of a Flexible Robotic Assembly Cell using Reactive Robots", *Artificial Intelligence in Industrial Decision Making, Control and Automation*, S. G. Tzafestas and H. B. Verbruggen, editors, Kluwer Academic Publishers, 1995, pp. 555-571.

[Tzafestas and Verbruggen, 1995] Spyros Tzafestas and Henk Verbruggen, "Artificial Intelligence in Industrial Decision Making, Control, and Automation: An Introduction", *Artificial Intelligence in Industrial Decision Making, Control and Automation*, Spyros G. Tzafestas and Henk B. Verbruggen, editors, 1995, Kluwer Academic Publishers, pp. 1-39.

[Vijaykumar and Arbib, 1987] Rukmini Vijaykumar and Michael A. Arbib, "Problem Decomposition for Assembly

Planning", *Proceedings of the International Conference on Robotics and Automation*, March 1987, pp. 1361-1366.

[Viswanadham and Narahari, 1988] N. Viswanadham and Y. Narahari, "Stochastic Petri net models for performance evaluation of automated manufacturing systems", *Information and Decision Technologies*, 1988, Volume 14, pp. 125-142.

[Volpe and Khosla, 1993] Richard Volpe and Pradeep Khosla, "A Theoretical and Experimental Investigation of Explicit Force Control Strategies for Manipulators", *IEEE Transactions on Automatic Control*, November 1993, Volume 38, Number 11, pp. 1634-1650.

[Walker et al., 1994] H. Fred Walker, Eugene Wallingford, and Ronald Meier, "Benchmarking the Transition to Agile Manufacturing: A Knowledge-Based Systems Approach", *International Journal of Flexible Automation and Integrated Manufacturing*, 1994, Volume 2, Number 3, pp. 197-208.

[Wang et al., 1996] Z. Y. Wang, K. P. Rajurkar, and A. Kapoor, "Architecture for Agile Manufacturing and its Interface with Computer Integrated Manufacturing", *Journal of Materials Processing Technology*, 1996, Volume 61, pp. 99-103.

[Watt, 1985] Douglas A. Watt, "Networking Software Support for FMS", *Manufacturing Engineering*, 1985, Volume 95, Number 3, pp. 103-106.

[Werling, 1991] Gerhard Werling, "Planning of Sensing Tasks in an Assembly Environment", *Journal of Intelligent and Robotic Systems*, 1991, Volume 4, pp. 221-254.

[Werling and Wild, 1994] Gerhard Werling and Harald Wild, "HOM — A hybrid object model for automatic assembly planning", *Journal of Intelligent Manufacturing*, 1994, Volume 5, pp. 153-163.

[Whitney, 1987] Daniel E. Whitney, "Historical Perspective and State of the Art in Robot Force Control", *The International Journal of Robotics Research*, Spring 1987, Volume 6, Number 1, pp. 3-14.

[Williams et al., 1994] T. J. Williams, P. Bernus, J. Brosvic, D. Chen, G. Doumeingts, L. Nemes, J. L. Nevins, B. Vallespir, J. Vlietstra, and D. Zoetekouw, "Architectures for integrating manufacturing activities and enterprises", *Computers in Industry*, Special Issue: CIM Architectures, 1994, Volume 24, pp. 111-139.

[Wolter, 1991] Jan D. Wolter, "A Combinatorial Analysis of Enumerative Data Structures for Assembly Planning", *Proceedings of the International Conference on Robotics and Automation*, April 1991, pp. 611-618.

[Wolter et al., 1992] Jan D. Wolter, Sugato Chakrabarty, and Jungfu Tsao, "Mating Constraint Languages for Assembly Sequence Planning", *Proceedings of the International Conference on Robotics and Automation*, May 1992, pp. 2367-2374.

[Wooldridge and Jennings, 1995] Michael J. Wooldridge and Nicholas R. Jennings, "Intelligent agents: theory and practice", *The Knowledge Engineering Review*, 1995, Volume 10, Number 2, pp. 115-152.

[Wooldridge et al., 1995] Michael Wooldridge, Jörg P. Muller, and Milind Tambe, "Agent Theories, Architectures, and Languages: A Bibliography", *Intelligent Agents II*, Agent Theories, Architectures, and Languages, M. Wooldridge, J. P. Muller, and M. Tambe, editors, August 1995, pp. 408-431.

[Wozny and Regli, 1996] Michael J. Wozny and William C. Regli, "Computer Science in Manufacturing",

Communications of the ACM, February 1996, Volume 39, Number 2, p. 33.

[Yamada et al., 1995] Y. Yamada, S. Nagamatsu, and Y. Sato, "Development of Multi-Arm Robots for Automobile Assembly", *Proceedings of the International Conference on Robotics and Automation*, May 1995, Volume 3, pp. 2224-2229.

[Yanco and Stein, 1993] Holly Yanco and Lynn Andrea Stein, "An Adaptive Communication Protocol for Cooperating Mobile Robots", *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, Jean-Arcady Meyer, Herbert L. Roitblat, and Stewart W. Wilson, editors, 1993, pp. 8-14.

[Yourdon, 1989] E. Yourdon, *Modern Structured Analysis*, Prentice Hall, Chapter 13, 1989.

[Yun, 1993] Xiaoping Yun, "Object Handling Using Two Arms Without Grasping", *The International Journal of Robotics Research*, February 1993, Volume 12, Number 1, pp. 99-106.

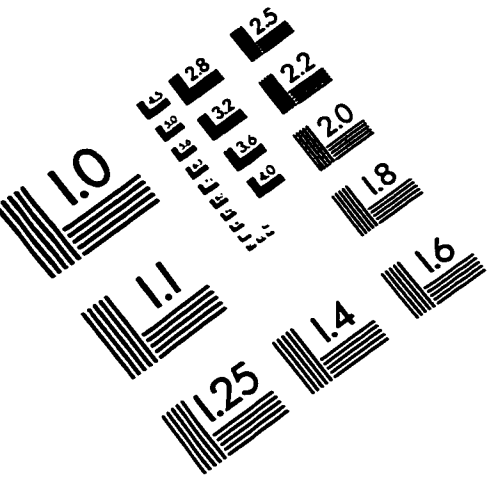
[Zanichelli et al., 1994] Francesco Zanichelli, Stefano Caselli, Antonio Natali, and Andrea Omicini, "A Multi-Agent Framework and Programming Environment for Autonomous Robotics", *Proceedings of the International Conference on Robotics and Automation*, May 1994, pp. 3501-3507.

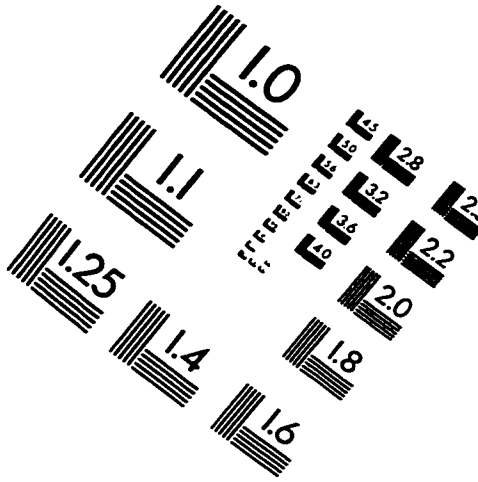
[Zhang et al., 1997] Jianwei Zhang, Yorck von Collani, and Alois Knoll, "On-line Learning of B-Spline Fuzzy Controller To Acquire Sensor-Based Assembly Skills", *Proceedings of the International Conference on Robotics and Automation*, April 1997, pp. 1418-1423.

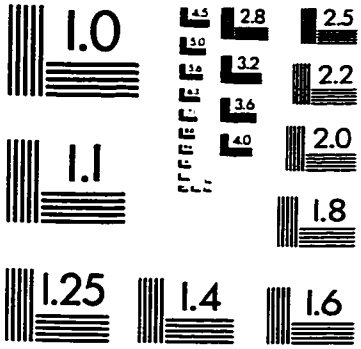
[Zhou et al., 1993] MengChu Zhou, Kevin McDermott, and Paresh A. Patel, "Petri Net Synthesis and Analysis of a Flexible Manufacturing System Cell", *IEEE Transactions on Systems, Man, and Cybernetics*, March 1993, Volume 23, Number 2, pp. 523-531.

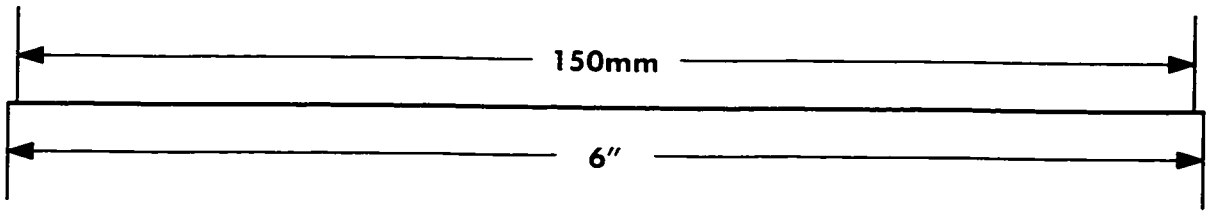
[Zussman et al., 1994] E. Zussman, H. Schuler, and G. Seliger, "Analysis of the Geometrical Features Detectability Constraints for Laser-Scanner Sensor Planning", *The International Journal of Advanced Manufacturing Technology*, 1994, Volume 9, Number 1, pp. 56-64.

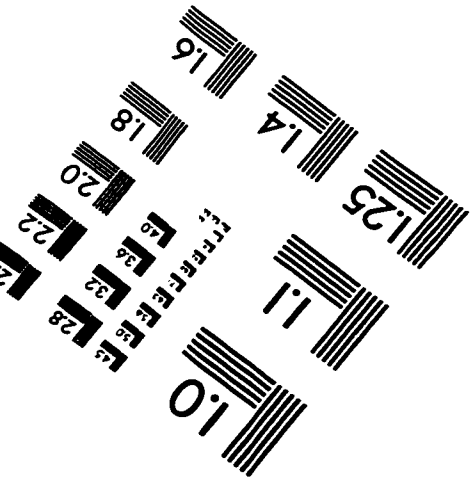
IMAGE EVALUATION TEST TARGET (QA-3)

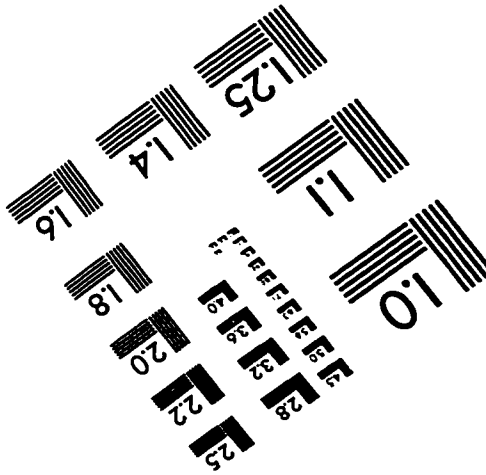












APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989