A Knowledge-Based Model Of Distributed Problem Solving

by

© Mark Evans

A thesis
presented to the University of Manitoba
in fulfillment of the
thesis requirement for the degree of
Doctor Of Philosophy
in
Computer Science

ISBN 0-315-47877-2

A KNOWLEDGE-BASED MODEL OF DISTRIBUTED PROBLEM SOLVING

BY

MARK EVANS

A thesis submitted to the Faculty of Graduate Studies of
the University of Manitoba in partial fulfillment of the requirements
of the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

The majority of artificial intelligence research has concentrated on applying knowledge-based programming techniques to program individual problem-solving agents with large amounts of knowledge thus enabling them to solve difficult problems. These agents work well with problems that are within the scope of their knowledge, but their performance degrades swiftly when they encounter problems that stretch past the boundaries of their expertise. A distributed problem-solving system consists of a collection of agents that are capable of performing knowledgeable, if not expert, problem solving and, more importantly, are capable of interacting to solve problems co-operatively. As a result, the agents are able to function as a team, resulting in a synergy which enables them to solve problems that could not be solved by any one individual agent.

This dissertation proposes a knowledge-based model of distributed problem-solving which provides a conceptual framework for constructing co-operative problem-solving systems. In this framework, each agent is separated into two components: a problem-solving component which embodies the agent's own problem-solving knowledge and skill; and a planning component which embodies a knowledge-based model of the agent's own abilities and those of other agents in the environment and which uses this knowledge to plan and co-ordinate co-operative activities with other agents. The two components are tightly coupled, but each addresses a particular aspect of an agent's operation. The planning component determines when the agent should perform problem-solving functions for other agents and when the agent should have other agents perform problem-solving functions for it as well as which agents should perform this problem solving. The problem-solving component is only concerned with performing the tasks that the planning component determines should be carried out by the agent itself. The separation of an agent's distributed problem-solving functions from its basic problem-solving functions, coupled with a knowledge-based approach to representing each set of functions produces a coherent, extensible, and maintainable framework for facilitating co-operative problem-solving among groups of agents.

This dissertation addresses the conceptual principles that a planning component must embody in order to facilitate co-operative problem solving. The types of knowledge that a planning component needs and the basic control functions required to select, interpret, and apply that knowledge are examined. The engineering issues required to support the construction of co-operative problem-solving systems based on this conceptual model of distributed problem solving also are examined.

The application of the model to the problem of cancer diagnosis and treatment is also presented. This domain illustrates the need for

co-operative problem solving by teams of health-care professionals. By applying the model to this domain, the types of knowledge and control functions used by health-care professionals during co-operative activities is uncovered.

Finally, an evaluation and analysis of the effectiveness of the model is presented and the issues to be addressed by future research are examined.

ACKNOWLEDGEMENTS

I would like to acknowledge the time, patience, and commitment of my advisor, David Scuse. Words cannot express the admiration I have for him. David is an amazing man and I am proud to consider him a colleague and, most importantly, a friend. I look forward to future times when I can somehow repay him for his efforts, without which I surely would have become a basket-case. I would also like to thank the other members of my examination committee: Dr. W. Pedrycz, Dr. M. Doyle, and Dr. R. Goebel. Furthermore, I would like to thank Dr. R.G. Stanton for his generous support and guidance over the past eight years.

I would like to apologize to my friends for being grumpy, sharp, and distant at times. I only wish you could know how much you really mean to me. I would especially like to thank Linda, Ken, Lisa, Dave, and Geoff for their their support over the last year.

It is not an easy task living with someone working on an advanced degree, and I wish to express my deepest love to my brother, David and my sister, Karen. I may not tell you enough, but I am very proud of you and I love you very much. Mom, what can I say? You have always been there for me and I know you are very proud, but always remember that I am a product of your kindness, selflessness, and love. I love you very much and I will always be your little boy. I would also like to thank Brandy for her love and wet kisses, and for listening to my worries when I couldn't talk to anyone else.

There is one other family that I must thank, the Miller clan: Dale, Anne, Brenna, Leslie, Jason, Sparky, and Muffin. For the last four years you have put up with my smart-ass comments and treated me like a member of your family. I love you guys very much.

Finally, there are two people for whom I live my life - my wife, Sharon, and my late father, Lloyd. Dad, thanks for all of the lessons you taught me, the discipline you instilled in me, and the love you gave me. I am proud to say that I have the greatest father who ever lived. You may not be with me now, but your influence will remain for as long as I live. I love you Dad. Sharon, you are the most thoughtful, loving, and giving person I know. You've helped me through the deepest depressions when I didn't think I would make it, and you've shared in my joys. You are the love of my life. Please forgive me for being moody and short with you at times. Without you, my life would be empty.

CONTENTS

ABSTRACT ii

ACKNOWLEDGEMENTS iv

Chapter page

I. INTRODUCTION 1

Overview 1
Individual Agents Are Too Weak 4
Co-operative Problem Solving Offers A Solution 7
Distributed Artificial Intelligence 7
An Example: Corporate Synergy 8
Advantages of Distributed Problem Solving 10
Summary 11

II. INTELLIGENT AGENTS 13

Overview 13
Knowledge-Based Problem-solving Models 13
Physical Symbol Systems 13
What is a Problem-Solving Model? 15
What is Knowledge 16
Conventional Models vs AI Models 16
The Water Jug Problem 18
Knowledge Manipulation 21
Basic Search 23
Heuristic Search 27
Constraint Satisfaction 29
Means-Ends Analysis 31
Agenda-Based Reasoning 32
Knowledge Representation 35
Propositional Representations 37
Structured Representations 42
Multiple Representation Paradigms 48
The Role of Metaknowledge in Knowledge Systems 50
Parallel Processing 53
Summary 56

III. DISTRIBUTED PROBLEM SOLVING 57

Overview 57
Inadequacy of the Closed-World Assumption 57
Distributed Problem Solving Offers a Solution 58
Open Systems Are Ideal Distributed Environments 61
Synergy in Open Systems 61

Requirements of An Open System 62
Interaction Is The Central Theme In Distributed Systems . . 65
Overview 65
Communication 65
Interaction Through Communication 67
Factors Influencing Interaction Among Agents 68
Overview 68
Grain Size 69
Explicitness 70
Focus 72
Structuring of Interactions 74
Flexibility 76
Survey of Distributed Problem-Solving Paradigms 79
Overview 79
Transactional Blackboards 80
Contract Nets 82
Actors 85
Functionally Accurate, Co-operative Distributed Systems 87
Summary 89

IV. A MODEL OF DISTRIBUTED PROBLEM SOLVING 91

Overview 91
Requirements of Distributed Problem Solving 91
Requirement 1: Ability to Construct Individual Agents . . 92
Requirement 2: Arbitrary Topology 93
Requirement 3: An Explicit, Knowledge-Based Planning Mechanism 95
A Model of Distributed Problem Solving 96
Overview of the Model 96
Physical-Level Assumptions 99
Planning Components 101
Meta-Level Knowledge Base 104
Overview 104
Problem Descriptions 106
Plan Knowledge 110
Task Knowledge 113
Agent Knowledge 115
Reasoning With The Knowledge 118
Activity Blackboard 121
Problems Partition 122
Plan Partition 123
Agent Activity Partition 124
General Partition 125
Managing Distributed Problem Solving 126
Overview 126
Introspection 128
Interrogation 131
Negotiation 132
Plan Execution 134
Integration 136
Co-ordinating Control 137
Summary 138

V. APPLYING THE MODEL 140

Overview 140
The Domain 141
Agents 143
Activities 147
Overview 147
General Examination 149
Diagnosis at a Cancer Centre 150
Establishing a Treatment Plan 151
Treatment Administration and Management 152
Snapshots of Distributed Problem Solving in the Domain . . 153
Overview 153
General Examination 156
Cancer Diagnosis 168
Establishing a Treatment Plan 173
Administering Treatment 177
Control of Distributed Problem Solving 183
Overview 183
Introspection 184
Interrogation 185
Negotiation 186
Integration 188
Plan Execution 189
Co-ordinating Control 191
Summary 192

VI. CONCLUSIONS 194

Summary of Distributed Problem-Solving Characteristics . . 194
The Model of Distributed Problem Solving 196
Evaluating The Model 200
Analysis and Future Work 207

Appendix page

A. SPECIFICATIONS AND GUIDELINES FOR IMPLEMENTING THE MODEL . . 212

Overview 212
Knowledge representation 213
Overview 213
Representing Problem-Solving Requests 214
Representing the Knowledge 217
Organization of the Knowledge Base 222
Basic Control Issues 223
Overview 223
Basic Control Algorithm 227
Additional Issues to Consider 229
Increasing Robustness Through Negotiation 230
Representing Relaxation Methods 231
Manipulating Relaxation Methods 233
Co-ordinating Efforts During Negotiation 237
Summary 239

B. A SAMPLE DIAGNOSIS SESSION 240

Co-operative Problem-Solving Activities 240
Summary 251

BIBLIOGRAPHY 252

Chapter I

INTRODUCTION

1.1 OVERVIEW

Intelligence is man's most powerful attribute; it enables him to react to his environment and to evolve as that environment changes. It is man's ability to reason intelligently that has enabled him to become the dominant creature in our world. Why then does man have such difficulty in understanding the core of his being - the mechanisms that underlie intelligence? For centuries man has pondered questions regarding intelligence and the principles of intelligent behaviour [Sternberg 82]. The greatest thinkers of the past and present have studied this phenomena and have constructed theories which they construed as explanations for this enigma; however, these theories remain only conjectures, their proofs eluded their creators. Why then do those involved in artificial intelligence (AI) research think that they will be able to solve such a difficult problem? Armed with perhaps man's greatest creation, the computer, AI researchers are attempting to solve the mysteries surrounding intelligence by building machines capable of exhibiting intelligent behavior.

AI research is not a new endeavor. The roots of AI can be traced back to philosophers such as Socrates, Aristole, and Descartes among others, but it is only recently that enough groundwork and advances in computer technology have emerged to enable empirical studies of intelligence to

be undertaken successfully. Such studies have produced significant advances, but much of the mystery still remains. Nevertheless, AI researchers have made the important realization that understanding general intelligence requires a multi-discipline approach; to gain a true understanding of intelligence requires research from several perspectives, each representing a piece of the overall puzzle. Philosophy, psychology, engineering, physiology, neuroscience, linguistics, and computer science all must make major contributions towards breaking the barrier that surrounds AI. It is only through the co-operative contributions of many areas of research that a more complete understanding of general intelligence can be achieved.

What, if anything, do we know about intelligence? We know that no single element is solely responsible for producing intelligence. Instead, intelligence is a complex, multi-faceted phenomena. Perhaps the most promising theory summarizing the basis of intelligence is stated by Minsky:

> "A mind is made up of many small processes called agents. Each mental agent by itself can only do some simple thing that needs no mind or thought at all. It is the joining of these agents into societies - in certain very special ways - that leads to true intelligence." [Minsky 86]

These statements propose that intelligence is a product of the interactions among many small, unintelligent agents. Individually, the agents are mindless, yet when many of these agents are organized into a group and allowed to interact, intelligent behaviour is attained.

What effect does this conjecture have on research in AI? Much of AI research to date has concentrated on creating individual agents capable

of solving difficult problems. These agents work well with problems that are within the scope of their knowledge, but their performance degrades swiftly when they encounter problems that stretch past the boundaries of their expertise. Recently, researchers have begun to examine mechanisms that will enable several such agents to interact and work together to produce more substantial forms of intelligent behavior [Davis 80],[Sridharan 87]. This research embodies Minsky's theory, although many of the approaches involve co-operation among several minds rather than co-operation among many (mindless) entities that make up an individual mind. Throughout all of the approaches, however, a fundamental theme prevails: co-operation is necessary to solve complex problems.

This dissertation proposes a knowledge-based model of distributed problem-solving which provides a conceptual framework for constructing co-operative problem-solving systems. In this framework, each agent is separated into two components: a problem-solving component which embodies the agent's own problem-solving knowledge and skill; and a planning component which embodies a knowledge-based model of the agent's own abilities and those of other agents in the environment and which uses this knowledge to plan and co-ordinate co-operative activities with other agents. The two components are tightly coupled, but each addresses a particular aspect of an agent's operation. The planning component determines when the agent should perform problem-solving functions for other agents, and when the agent should have other agents perform problem-solving functions for it as well as which agents should perform this problem solving. The problem-solving component is only

concerned with performing the tasks that the planning component determines should be carried out by the agent itself. The separation of an agent's distributed problem-solving functions from its basic problem-solving functions coupled with a knowledge-based approach to representing each set of functions produces a coherent, extensible, and maintainable framework for facilitating co-operative problem-solving among groups of agents.

1.2 INDIVIDUAL AGENTS ARE TOO WEAK

Although there are many definitions of artificial intelligence, the most encompassing defines AI as the science of making computers more useful [Winston 84]. AI attempts to provide techniques and methodologies that satisfy the requirements of applications that resist conventional data processing techniques. In effect, AI concentrates on applications for which algorithmic, data-oriented methods are inappropriate [Rauch-Hindin 88]. In recent years, AI applications have begun emerging in many areas of society that previously resisted the influence of the computer [O'Shea 87]. For example, researchers have developed computer programs known as expert systems which embody the knowledge and skills of human experts and which are capable of achieving levels of performance comparable to their human counterparts [Liebowitz 88]. Man-machine interfaces have also been developed to enable users to communicate with the computer and its resources using a subset of a natural language (e.g. English), making the power of the computer accessible to people who are unfamiliar with computer languages [Rich 84].

Each of these endeavors, and many others, takes advantage of the techniques that have evolved from AI research. The success of these applications is largely due to the realization that representing and manipulating large amounts of domain-specific knowledge are the keys to solving complex problems [Waterman 86]. The consensus is that, whether it is expert knowledge encoded in an expert system or knowledge about valid sentence meanings in a natural language understanding system, in order to create systems that exhibit intelligence of one form or another we must acquire, organize, represent, and manipulate large amounts of domain-specific knowledge. Hence, much of the research in AI has concentrated on developing techniques for representing and manipulating knowledge.

Consider the hijacker example in Figure 1.1. The use of an algorithmic problem-solving method is infeasible because numerous people must be searched which is time-consuming, costly, and inconvenient. This application requires another problem-solving method in which a more "intelligent" approach is used. An alternative approach might use a hijacker profile (knowledge base) to guide the search and thus prune the number of people who must be searched. This approach is not infallible, but it does provide a more feasible alternative. In effect, the method balances the acceptability of a solution with the effort that must be expended to derive it. It is domain-specific knowledge (in this case the hijacker profile and metal detector) that provides the key to achieving acceptable solutions [Waterman 86].

Although much success has been enjoyed in the development of these so-called knowledge-based systems (KBS), there remain limitations in

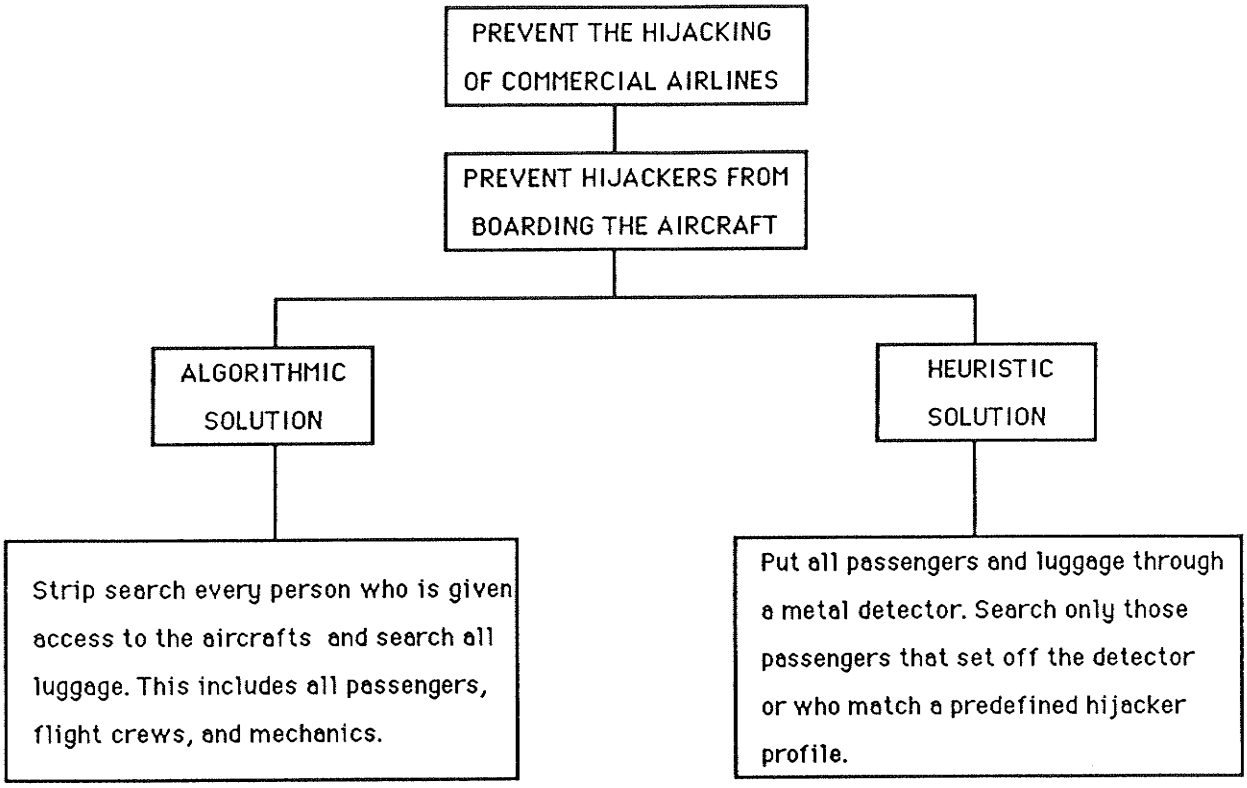


Figure 1.1: The Hijacker Example

their use and in their ability to solve complex problems. These systems require very narrow scopes in order to function effectively. As a result, the systems are capable of exhibiting intelligent action for only a relatively small set of problems [Hayes-Roth 83]. Furthermore, the systems can only perform those actions which their knowledge bases have been programmed to handle. The systems are extremely susceptible to errors and poor performance as problems stretch past the boundaries of their knowledge [Turban 88]. Moreover, creating and extending the set of problems that a system is capable of solving is a long and tedious process [Hayes-Roth 83]. Consequently, AI research has focused on developing numerous special-purpose systems for a variety of

applications, each capable of exhibiting very narrow forms of intelligent behaviour. Experience has shown that it is impossible, using current AI techniques, to build an individual system that is capable of exhibiting a wide variety of intelligent action [Tsotsos 85].

1.3 CO-OPERATIVE PROBLEM SOLVING OFFERS A SOLUTION

1.3.1 Distributed Artificial Intelligence

One way to create more powerful intelligent systems is to network special-purpose systems and enable them to interact and work co-operatively. It is not difficult today to build a network facility (physical or logical) that can provide the basis for a society of special-purpose systems [Chang 88]. The difficulty lies in co-ordinating the interactions among the individual agents in the network [Cammarata 83]. A subfield of AI research known as Distributed Artificial Intelligence (DAI) has evolved that studies co-operative problem solving among groups of intelligent problem solvers [Sridharan 87],[Jagannathan 86]. DAI research examines problem-solving methodologies that enable the creation of distributed environments in which several agents co-operate to produce a synergy of their abilities. The principle of synergy states that the combined effects of the collection of agents will be greater than the sum of their individual effects [Fowler 64]. Distributed AI research attempts to develop problem-solving techniques that embody this principle in order to facilitate the creation of co-operative problem-solving environments. Problem-solving techniques of this sort enable a developer to organize individual agents into distributed environments and allow the agents to interact in order to share their abilities and perform tasks co-operatively.

1.3.2 An Example: Corporate Synergy

Figure 1.2 illustrates the organization of a corporate entity, an example of a distributed problem-solving system in which many play an active part everyday. A company is typically divided into a hierarchy of departments, each of which is assigned a specific set of roles and functions. No single department is capable of performing all of the functions that the company is responsible for. Instead, the departments typically work together to take advantage of their combined resources. Synergy is evident throughout the corporate structure because the departments are actually composite entities (agencies) which are comprised of groups of employees. Employees are intelligent agents that are aware of their own roles and functions within a specific department. Employees may also be aware of some of the roles and functions of other employees or other departments. This partial organizational knowledge enables an employee to interact with other employees to obtain assistance in carrying out tasks. Employees can communicate with one another using such mechanisms as memos, conversations, conferences, and meetings in order to co-ordinate the sharing and exchanging of their abilities. Each employee has a role and contributes some resource (however small) to the overall operation of the departments and the company. The roles and functions of the company are only realized by the synergy of the abilities of the individual employees and departments.

The size and complexity of the company will dictate the complexity of the techniques used to co-ordinate co-operative problem solving among the members of the organization. Management research has studied and

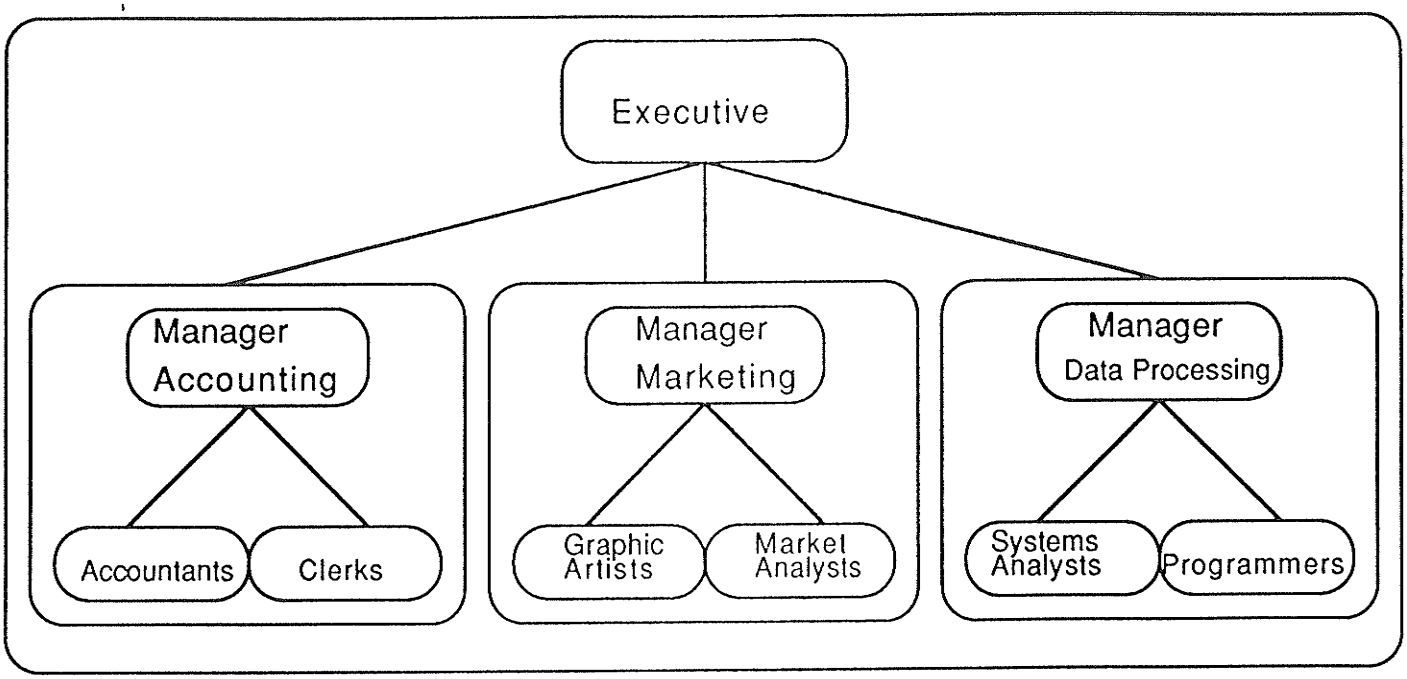


Figure 1.2: A Corporate Hierarchy

developed many models of co-operative problem solving in human organizations including: the Rational Model [Allison 71], the Garbage Can Model [Cohen 72], the Process Model [Cyert 63], and the Political Model [Allison 71]. Each of these models examines particular types of organizational decision-making strategies that can be used to co-ordinate co-operative problem solving among members of an organization. Organizational decision making is the foundation of co-operative problem solving. Thus, DAI includes the study of organizational decision-making techniques that can be used to facilitate co-operation among groups of computerized agents (and in some cases, among groups of humans and groups of computer systems [Chang 86]).

1.3.3 Advantages of Distributed Problem Solving

According to Huhns, there are five primary reasons why one would want to study and utilize DAI [Huhns 87]:

- DAI can provide insights and understanding about interactions among humans, who organize themselves into various groups, committees, and societies in order to solve problems.
- DAI can provide a means for interconnecting multiple expert systems that have different, but possibly overlapping expertise, thereby enabling the solution of problems whose domains are outside that of any one expert system.
- DAI can potentially solve problems that are too large for a centralized system, because of resource limitations induced by a given level of technology. Limiting factors such as communication bandwidths, computing speed, and reliability result in classes of problems that can be solved only by a distributed system.
- DAI can potentially provide a solution to a current limitation of knowledge engineering: the use of only one expert. If there are several experts, or several nonexperts that together have the ability of an expert, there is no established way to engineer a successful system.
- DAI is the most appropriate solution when the problem itself is inherently distributed, such as in distributed sensor nets and distributed information retrieval.

Huhns goes on to suggest that DAI also provides the next step beyond current expert systems. He suggests the following approach to the development of expert systems using DAI techniques: build a separate subsystem for each problem domain based on the ability of each expert, and then make these subsystems co-operate. This approach potentially has the following additional advantages:

- Modularity: The complexity of an expert system increases rapidly as the size of its knowledge base increases. Partitioning the system into N subsystems reduces the complexity by significantly more than a factor of N. The resultant system is easier to develop, test, and maintain.

- Speed: The subsystems can operate in parallel.
- Reliability: The system can continue to operate even if part of it fails.
- Knowledge acquisition: It is easier to find experts in narrow domains. Also, many problem domains are already partitioned or hierarchical - why not take advantage of this?
- Reusability: A small, independent expert system could be part of many distributed expert systems - its expertise would not have to be reimplemented for each. [Huhns 87]

1.4 SUMMARY

DAI research is concerned with the development of computer systems consisting of groups of individual agents that co-operate to solve complex problems. In the remaining chapters, I describe the fundamental principles involved in the development of intelligent agents and I propose a knowledge-based model of distributed problem solving which facilitates co-operation among groups of intelligent agents. Chapter 2 contains a general overview of the principles embodied in current AI techniques; these techniques provide the ability to construct individual computerized agents that can subsequently be combined to form distributed problem-solving environments. Chapter 3 examines the issues involved in distributed problem solving and discusses several models that have been developed to facilitate co-operative problem solving. Chapter 4 presents a knowledge-based model of distributed problem solving. The model is designed primarily for distributed expert system applications in which several subsystems are developed individually and then placed in a problem-solving network where they can solve problems co-operatively. Chapter 5 examines the application of the model to problem-solving in a particular domain, that of cancer diagnosis and

treatment. Chapter 6 presents a summary of the contributions of this research, including a discussion of the properties of the model, the ramifications that these properties have on the effectiveness of the model, and the issues that need to be addressed in the future. Appendix A provides a detailed description of the engineering specifications required to create an operational system based on the conceptual model outlined in Chapter 5. Appendix B provides an example of the application of the model to enable several agents to diagnose a particular patient's health problems co-operatively.

Chapter II

INTELLIGENT AGENTS

2.1 OVERVIEW

This chapter examines the organizational principles and techniques that enable the representation and manipulation of knowledge in computer programs. I concentrate on describing the general principles and methodologies that have been developed to facilitate the creation of knowledge-based programs capable of intelligent action, albeit within very narrow and specialized application domains. I examine the concepts inherent in knowledge programming, followed by an analysis of some of the basic knowledge representation and manipulation paradigms that have been developed. The remainder of the chapter discusses advanced forms of knowledge-based programming such as the role of metaknowledge and parallel processing in knowledge-based systems. Readers familiar with knowledge-based systems may omit this chapter.

2.2 KNOWLEDGE-BASED PROBLEM-SOLVING MODELS

2.2.1 Physical Symbol Systems

In order to create intelligent agents, one must have an understanding of the requirements of intelligence. We know that no single element accounts for all aspects of intelligence; intelligence is a composite phenomena. However, our understanding of intelligence is by no means complete. Much of our knowledge about what intelligence is and about the

requirements for intelligent action is only speculative. Nevertheless, this does not deter us from creating theories of intelligence. One such theory is the physical symbol system hypothesis:

> "A physical symbol system consists of a set of entities, called symbols, which are physical patterns that can occur as components of another type of entity called an expression (or symbol structure) ... At any instant of time the system will contain a collection of these symbol structures. Besides these structures, the system also contains a collection of processes that operate on the structures to produce other structures... A physical symbol system has the necessary and sufficient means for general intelligent action." [Newell 76]

This hypothesis is rather overbearing in that "necessary and sufficient" implies that any intelligent agent upon analysis must prove to be a physical symbol system. This implies that the basis of human intelligence is a physical symbol system. Although this may in fact prove to be true, to date there is no concrete evidence to support or refute this implication [Rich 83]. AI research deals with subjecting this hypothesis to empirical study using the computer as a medium for experimentation. AI researchers study the creation of computer programs that can perform particular tasks that are regarded as requiring intelligence. These computer programs are created by representing and manipulating symbols and symbol structures within a physical symbol system.

This chapter examines the fundamental principles surrounding the creation of special-purpose intelligent systems using physical symbol systems. I define a knowledge system as a computational entity that is capable of representing and manipulating symbols and symbol structures that denote domain knowledge in order to perform tasks or solve problems

within a specified domain. In essence, a knowledge system is a physical symbol system that provides the computational basis for creating intelligent systems.

2.2.2 What is a Problem-Solving Model?

A problem-solving model is a scheme for organizing reasoning steps and domain knowledge to construct a solution to a problem [Nii 86]. In theory, every problem-solving model consists of three components: objects, operators, and a control strategy [Nilsson 80]. The objects represent facts and assumptions about the domain entities, while the operators are the available operations that can be applied to manipulate the objects. The control strategy determines the order in which the operators are to be applied to move from a given problem state (initial state) to a desired problem state (goal state). A problem state is the configuration of a problem at a given point in time after a particular set of operators has been applied (operators are applied to a problem state to produce new problem states). The problem space is the set of all valid problem states. A solution is a set of operators (typically ordered) which when applied to the initial state generates a goal state.

A problem-solving model provides a conceptual framework for organizing and representing knowledge (objects and operators) and for controlling the manipulation of that knowledge in order to solve problems (control strategy). The study of knowledge representation involves the examination of methods of representing knowledge symbolically, while the study of knowledge manipulation involves investigating methods of manipulating knowledge in order to solve

problems. Problem-solving models therefore provide frameworks for constructing physical symbol systems.

2.2.3 What is Knowledge

The goal of every problem-solving model is to represent and manipulate knowledge so that the knowledge can be used to solve difficult problems. But what is knowledge? When we talk about people who solve problems, we always talk about what they have to **know** in order to solve them [Barr 81]. The knowledge that a person possesses is what we use to describe that person's ability to behave intelligently. In simple terms, knowledge consists of domain objects, facts about the objects, and operations for manipulating the objects and facts in order to produce new objects and new facts. The collection of domain objects, facts, and operators form a knowledge base describing what is known about a specific domain. AI involves the study of the representation and manipulation of various types of knowledge in order to develop methodologies for creating intelligent systems.

2.2.4 Conventional Models vs AI Models

In the conventional model of problem solving, the control strategy and the domain knowledge (object and operators) are combined in the form of an algorithm that specifies "how" to solve a problem. Problem solving is performed by executing the steps of the algorithm, a sequence of operator applications that produces a solution. This approach is very efficient because an algorithm specifies a direct and complete path to a solution. Furthermore, it is possible to handle the special cases of an application by tailoring the algorithm accordingly. However, systems

developed using the conventional model are often difficult to modify and maintain because the objects, operators, and control strategy are intertwined. This is especially true of large, complex systems in which much of the description of the problem is hidden and distributed throughout the algorithm. The domain knowledge is implicitly represented in an algorithm which is often an extremely complex and cumbersome entity [Evans 87].

In contrast, the AI model of problem solving enforces a strict separation of the knowledge from the control strategy as illustrated in Figure 2.1. The knowledge base represents a description of "what" the system knows about a given application, while the control strategy represents a general problem-solving method that searches the knowledge base for a set of operator applications that produces a solution to a given problem. In effect, the control strategy is responsible for determining an algorithm based on the characteristics of a given problem and the available knowledge in the knowledge base.

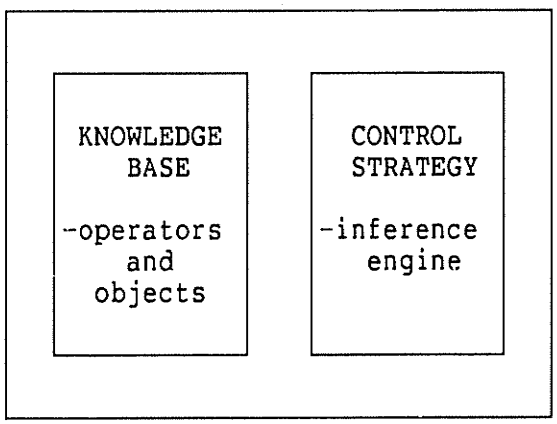


Figure 2.1: AI Model of Problem Solving

The separation of the knowledge from the control strategy produces a more flexible and modifiable model that is ideally suited to incremental development techniques required to tackle complex, ill-understood applications; the performance and competence of the system can be modified by adding new knowledge or by refining existing knowledge without having to modify the control strategy. Consequently, the developer can extend and refine the knowledge describing the problem without having to specify how the new knowledge is to be applied [Keller 87]. Moreover, we can also apply the same control strategy to a different set of operators from another application domain; one need only construct another knowledge base which describes the new domain [Barr 81a]. If the control strategy is improved, the improvements are applied to each knowledge base that is manipulated by the control strategy. On the other hand, this approach tends to be less efficient than its conventional counterpart because of the search involved in selecting the knowledge at each problem-solving step [Evans 87a]. In addition, it is often difficult to handle exceptions or special cases without affecting the complexity of the knowledge base or modifying the control strategy [Rich 83].

2.2.5 The Water Jug Problem

The classic waterjug problem, illustrated in Figure 2.2, demonstrates the fundamental differences between the two models. The problem is straightforward: there are two water jugs, each containing a specified amount of water; in this case a 5 litre jug containing 5 litres of water, and a 2 litre jug that is empty. This configuration represents

the initial problem state. The two jugs are the domain objects that must be manipulated. The operations available to manipulate the jugs are also illustrated in Figure 2.2. Each operation has a set of preconditions that must be satisfied before the operator is applicable. The desired solution to this particular problem is to have 1 litre of water in the 2 litre jug (this represents the goal state). The solution must be derived using the available operations. There are no measuring devices and one must not pour more water into a jug than it can contain.

```
Initial State: 5L jug = 5;   2l jug = 0

Goal State:    5l jug = x;   2l jug = 1

Available Operators:

  R1: IF    5L jug is not empty   and
            2L jug is not full
      THEN  pour the contents of the
            5L jug into the 2L jug

  R2: IF    5L jug is not full   and
            2L jug is not empty
      THEN  pour the contents of the
            2L jug into the 5L jug

  R3: IF    5L jug is not empty
      THEN  empty the 5L jug

  R4: IF    2L jug is not empty
      THEN  empty the 2L jug
```

Figure 2.2: The Waterjug Problem

The conventional solution to the waterjug problem is illustrated in Figure 2.3. Here we have an algorithm that specifies the exact sequence of operator applications that generates a solution. In contrast, an AI solution illustrated in Figure 2.4 utilizes a nondeterministic method in

```
INITIAL STATE:   5L = 5; 2L = 0

1) apply R1; pour the contents of
   the 5L jug into the 2L jug.
     new problem state: 5L = 3; 2L = 2

2) apply R4; empty the 2L jug.
     new problem state: 5L = 3; 2L = 0

3) apply R1; pour the contents of
   the 5L jug into the 2L jug.
     new problem state: 5L = 1; 2L = 2

4) apply R4; empty the 2L jug.
     new problem state: 5L = 1; 2L = 0

5) apply R1; pour the contents of
   the 5L jug into the 2L jug.
     new problem state: 5L = 0; 2L = 1
        *** goal state  ***
```

Figure 2.3: Algorithm for Solving the Waterjug Problem

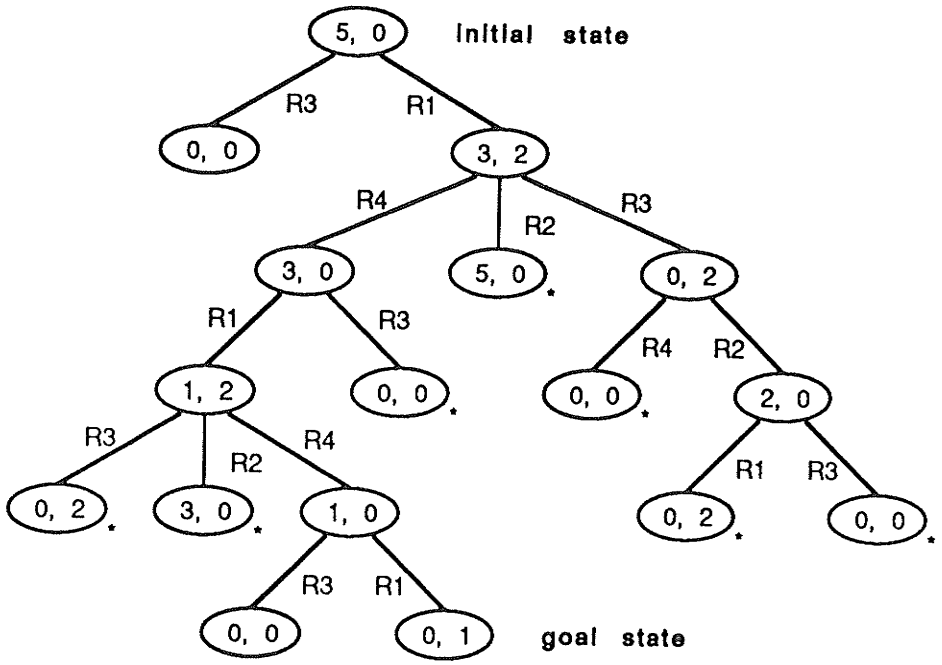


Figure 2.4: Nondeterministic Solution to the Waterjug Problem

which a control strategy exhaustively applies the available operators until a goal state is generated. Once a goal state is generated, the solution is the sequence of operator applications that leads from the initial state to the goal state.

The conventional method will generate a solution much more quickly than the AI approach because it represents a direct path to a goal state. However, what happens if we modify the problem description, either by modifying the initial or goal states or by modifying or adding operators? In the conventional approach, we are forced to refine or rewrite the algorithm to reflect the changes made to the problem description. When the AI approach is used, we can modify the knowledge base to reflect the changes in the problem description, and the control strategy will search for a sclution using the new description. Hence, the two approaches are intuitively different. The conventional model is geared towards algorithmic applications which require a large and relatively stable body of data manipulations, while the AI model is geared towards applications which require the manipulation of a large and changing body of knowledge. Each model provides a trade-off of some sort and it is the underlying application characteristics that dictate the model that is the most appropriate.

2.3 KNOWLEDGE MANIPULATION

This section examines the principles involved in knowledge manipulation and discusses five types of knowledge manipulation techniques. This is not intended to be an exhaustive examination of the area, rather it serves as an overview of the concepts involved in knowledge manipulation

and of some of the strategies that have been developed to facilitate the manipulation of knowledge by computer programs.

When a system is required to do something that it has not been told explicitly how to do, it must reason - it must determine what it needs to know from what it already knows [Fischler 86]. Reasoning involves manipulating knowledge to make explicit that which is implicit. For example, from the information (1) all men are mortal, and (2) John is a man, the system should be able to obtain the explicit statement (3) John is mortal. Statement (3) is implicit in statements (1) and (2). It is the manipulation of statements (1) and (2) that enables the system to obtain statement (3).

The study of knowledge manipulation involves determining the fundamental principles involved in the various forms of reasoning as well as techniques for developing strategies (referred to as control strategies) to incorporate reasoning into computer systems. A control strategy must determine the knowledge and the manipulations of that knowledge that are required to solve a given problem. A control strategy specifies a mechanism that controls systematically the application of domain knowledge in order to solve problems or perform tasks. These strategies perform problem solving by manipulating the domain objects and operators represented by symbols and symbol structures in a knowledge base. In effect, they specify ways in which the symbols and symbol structures can be manipulated within the constraints imposed by the representation conventions [Newell 76]. A formal discussion of the logical foundations of control can be found in [Nilsson 80].

2.3.1 Basic Search

AI methodologies employ a declarative model of problem solving in which the problem specification (representation) defines the permissible operators, but does not define the sequence of operators that will lead to a solution. The problem space is represented conceptually as a tree or graph in which each node denotes a problem state, such as the example of Figure 2.4. Problem solving becomes a search for a sequence of operator applications that solves a given problem. A solution is a path through the problem space from the initial state to a valid goal state.

The search of the problem space can be carried out using either a state-space search or a problem reduction search [Rich 83]. In a state-space search, the problem space is represented as an OR graph or tree in which each node represents a complete state of the problem (see Figure 2.5). The search proceeds by applying valid operators to each problem state in order to generate new problem states until a goal state is produced. On the other hand, some problems can be subdivided into two or more smaller subproblems which are frequently easier to solve than the entire problem as a whole [Rich 83]. This is the principle used in problem reduction search. A node in the problem space represents either a complete state of the problem or a state of a subproblem. The problem space can be represented by nodes in an AND graph or tree or an AND/OR graph or tree as illustrated in Figure 2.6. In order to solve an AND graph or tree, the initial problem state must be immediately solvable or its successors (subproblems) must **all** be solvable. On the other hand, a node in an AND/OR graph or tree is solvable if the node itself is solvable or it can be broken down into

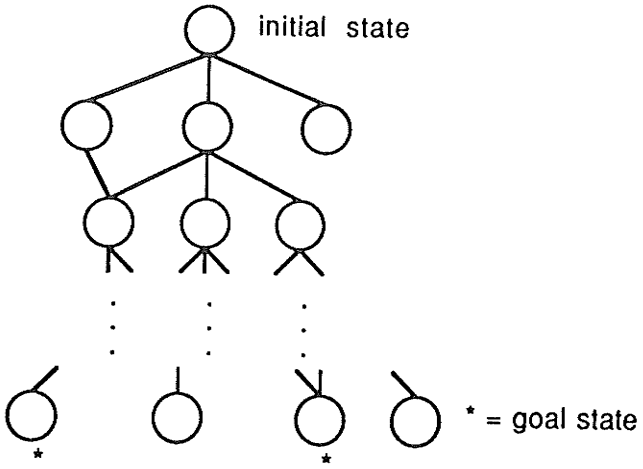


Figure 2.5: An OR Graph

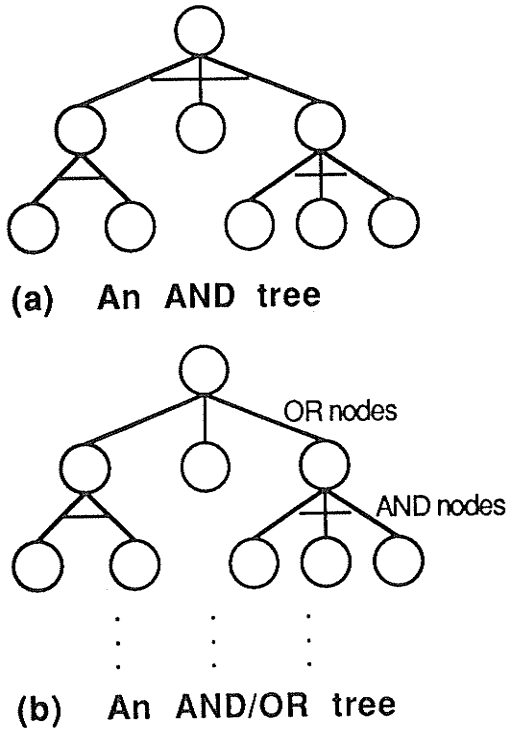


Figure 2.6: An AND Tree and an AND/OR Tree

subproblem states which are AND nodes that are **all** solvable or OR nodes of which **at least one** is solvable [Barr 81].

The objective of a search strategy is to find a path in the problem space from a given initial state to a goal state. Search can proceed in one of two directions: forwards, from the initial state towards a goal state; or backwards, from goal states towards the initial state [Charniak 85]. Both a state-space and problem reduction search can be applied in either the forwards or backwards direction. A state-space search in the forwards direction would start with the initial state, apply one or more valid operators in order to generate new problem states, and continue selecting and applying operators to these new states until a goal state is generated. The water jug problem in Figure 2.4 is an example of a state-space search in the forwards direction. A state-space search in the backwards would begin with the goal state, select an operator that can produce the goal state, apply the inverse of the operator to the goal state to generate new states, and continue selecting and applying operators to the new problem states until the initial state is produced. Forwards and backwards problem reduction searchs are carried out in similar fashion [Rich 83].

In many problems it is possible to search the problem space in either the forwards or backwards direction, while other problems can only be solved using one of the search directions [Winston 84]. For example, the water jug problem cannot be solved using a backwards state-space search because some of the operators do not have valid inverses (the inverse of the empty operator is not definable because one cannot determine the amount of water that was in a jug before it was emptied). When it is

possible to use either search direction, one must determine which direction is most appropriate. According to Rich [Rich 83], three factors influence the choice of search direction:

- Are there more possible start states or goal states? We would like to move from the smaller set to the larger (and thus easier to find) set of states.
- In which direction is the branching factor (i.e. fan-out of the problem space) greater? We would like to proceed in the direction with the lowest branching factor.
- Will the program be asked to explain or justify its actions to a user? If so, it is important to use a search direction that corresponds to the way the user understands the problem.

For some problems, it is possible (and often desirable) to use a mixture of the two search directions [Nilsson 80]. Bidirectional search is one such approach in which the search proceeds both forwards and backwards concurrently, and a solution is attained when the two searches meet. Once again, the characteristics of a given problem will dictate whether bidirectional search is applicable.

Most often, basic search strategies attempt to systematically examine the problem space to determine a solution (as opposed to randomly applying operators). Depth-first and breadth-first search are examples of systematic search strategies [Winston 84]. In depth-first search, a single branch of the tree representing the problem space is explored until it yields a solution and other branches are only examined after the initial branch is exhausted or a predetermined depth bound is reached. Conversely, a breadth-first search explores each node in each level of the tree representing the problem space before proceeding to the next level. Each of strategies is therefore better suited to

particular problem space topologies [Winston 84]. Both techniques, however, are susceptible to a combinatorial explosion of possible problem states that must be explored and thus are usually too inefficient to be useful when dealing with the complex and expansive problem spaces of real-world problems.

2.3.2 Heuristic Search

Most hard problems have too many problem states for it to be possible to exhaustively search the entire problem space [Rich 83]. Such problems are often solvable using heuristic techniques in which the nodes of the problem space are rated using a heuristic function and only the most promising nodes (those with the highest rating) are examined. This technique is commonly referred to as a best-first search strategy. As a result, problems with very large problem spaces can be solved because the search is able to avoid blind alleys and dead-ends thus reducing the amount of unfruitful effort that must be expended to generate a solution [Hayes-Roth 83].

Consider the 8-puzzle problem in Figure 2.7. The problem consists of a tray with eight numbered tiles and one blank tile. In order to solve the problem, one must start with a given configuration of the tiles (initial state) and rearrange the tiles by moving the blank tile up, down, right, or left until the desired goal configuration is achieved. Solving this problem using an exhaustive search involves applying the valid operators to the initial state and each of its successors until the goal state is generated. This approach, however, is susceptible to combinatorial explosion because the search proceeds through the problem

space in an uniformed manner (it does not consider which moves are better than others). A heuristic function that can guide the search in the 8-puzzle problem is to rate the problem states according to the number of tiles that are out of place in comparison with the desired goal state. The heuristic search uses this rating information to select the most promising new problem states to examine during the search; these are the states which are estimated to be closest to the goal state. As a result, the search is able to focus on the best potential solution paths and avoid those paths that are judged to have little chance of leading to a solution.

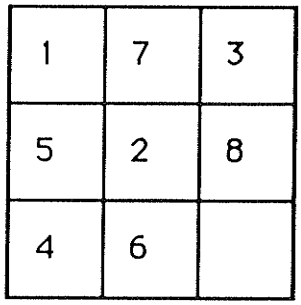


Figure 2.7: The 8-Puzzle Problem

Although heuristic search is a viable means of combating combinatorial explosion in hard problems, there are several difficulties surrounding its use. The quality of the heuristic function used to guide the search directly influences the effectiveness of the search; good heuristics prune the problem space and help avoid unfruitful effort, while poor heuristics often lead to much expenditure of time and

resources in examining dead-end paths because the goodness of paths is poorly estimated. In addition, some heuristics are able to ensure that the optimal solution will be found, while others (often the best ones) may occasionally cause optimal paths to be overlooked [Rich 83]. Hence, a heuristic search is only as good as the heuristic function used to rate the nodes of the problem space [Hayes-Roth 83].

Another problem surrounding heuristic search is that it is often impossible to find a single heuristic function that can be used in real-world problems. The complexity and uncertainty inherent in these problems usually necessitates several attributes be considered when evaluating a problem state, and more often than not it is impossible to find a heuristic function that can combine all of the relevant factors into a single rating. In such problems, the concept of heuristic search is employed by introducing heuristics into the domain operators (as was done in the hijacker profile in Figure 1.1). This approach leads to feasible heuristic solutions, but in a less structured and less mathematically-based manner [Rich 83].

2.3.3 Constraint Satisfaction

Many problems in AI can be viewed as problems of constraint satisfaction in which the goal is to discover some problem state that satisfies a given set of constraints [Rich 83]. Problems of this sort include map coloring, database retrieval for conjunctive queries, and edge labelling in computational vision [Shapiro 87, p. 209]. Many design tasks which must construct solutions within fixed limits of time, cost, and materials can also be viewed as constraint-satisfaction problems.

Constraint-satisfaction problems can be solved by extending basic and heuristic search strategies to include mechanisms for maintaining and manipulating lists of constraints associated with each problem state. As the search strategy examines nodes in the problem space, it must ensure that the problem state described by each node satisfies the constraints imposed on the problem. A solution is attained when a problem state is generated which satisfies all of the constraints.

The general algorithm for constraint-satisfaction is stated by Rich [Rich 83]:

- Until a solution is found or until all paths have been exhausted:
  1. Select the next node in the problem space to examine (beginning with the initial problem state)
  2. Apply the constraint rules to the current node to generate all possible new constraints (state-specific constraints)
  3. If the set of constraints contains a contradiction, then eliminate the path (dead-end)
  4. If the constraints are all satisfied then a solution has been found
  5. Otherwise, apply the problem space rules to generate a set of one or more new problem states that are consistent with the current set of constraints

This algorithm illustrates that constraint-satisfaction requires two sets of operators: constraint rules which generate new constraints based on current constraints and the characteristics of a given problem state; and problem space rules which are applied to problem states to generate the next set of states to be examined. The problem space rules, and any heuristic functions that are used to guide the search, will make reference to the constraints associated with a problem state as well as

the basic problem description information that a state contains. Thus, problem solving is a process of moving through the problem space by generating new states and new constraints, ruling out those states which do not satisfy the constraints, until a solution is found.

2.3.4 Means-Ends Analysis

The search strategies that I have examined so far can reason forwards or backwards. Another approach, known as means-ends analysis, compares the initial state to the goal state and chooses its next action based on how it can reduce the differences between the two states [Shapiro 87, p. 578]. Reasoning can proceed either forwards from the initial state or backwards from the goal state, the direction that is used depends on the operations that can reduce the differences between the two states.

Means-ends analysis is essentially a form of problem reduction search in which a problem is reduced to a set of differences between the initial state and the goal state. The search strategy maintains a list of the current differences and selects operators that can reduce the differences until all such differences have been resolved. Problem solving centres around solving a set of subproblems (reducing differences) which may in turn be broken down into smaller subproblems until each subproblem can be solved.

Many of the details involved in means-ends analysis have been omitted in this discussion. For instance, the order in which differences are reduced is often critical, as is the choice of operators to be used to reduce each difference [Charniak 85]. In most cases, it is necessary to

reduce significant differences first, but this requires some domain-specific mechanism that can rank each difference appropriately. Furthermore, a similar ranking mechanism is required to rate the operators which are capable of reducing each difference. Further complicating the process is the fact that the differences may interact; working on one difference may interfere with reducing another. Interactions of this sort can require the system to perform extensive backtracking during the search or require the addition of planning mechanisms that can detect and resolve interactions [Barr 81a]. Finally, the complexity inherent in real-world applications often results in numerous permutations of differences which the basic means-ends analysis strategy cannot cope with effectively [Winston 84].

2.3.5 Agenda-Based Reasoning

The control strategies presented in the preceding sections all perform search in a systematic manner, carrying out steps in a well-defined order. Frequently, it is advantageous to use a less systematic approach in which the search strategy maintains a list of the tasks it should perform, and determines its next action based on this task list and the current state of problem solving [Aikins 83]. Control strategies of this sort employ what is referred to as opportunistic reasoning; the control strategy determines opportunistically the actions it should perform in order to move closer to solution.

An agenda-based system is one of the most popular methods used in AI systems to incorporate opportunistic control. An agenda is a data structure, maintained by the control strategy, whose entries are

referred to as tasks. Each task represents some piece of work that needs to be accomplished during problem solving. In addition, a task entry can include a source for the task, reasons for executing the task, and a priority indicating the usefulness of executing the task. The agenda usually begins with one or more initial task entries and new tasks are placed on the agenda as side effects of executing other tasks.

The control strategy manipulates the agenda in order to determine the best sequence for executing pending tasks. Hence, control is carried out using a best-first search. This is accomplished by selecting tasks from the agenda based on the reasons (justifications) and priorities associated with each entry [Shapiro 87, p. 4]. Finding the most promising task on each cycle can be accomplished by maintaining the agenda sorted by a rating factor (based on justifications and priorities). When a new task is created, it is inserted in its proper place according to its rating. When justifications for a task in the agenda change, the rating associated with the entry is adjusted and the entry is moved to its correct place in the list.

An agenda-based control strategy becomes significantly more powerful if both positive and negative evidence are accumulated and stored as justification for each task [Aikins 83]. Maintaining evidence for and against executing tasks enables the control strategy to determine the best order of task execution. Moreover, as tasks are executed, they can add both positive and negative evidence to the justifications for other tasks which enables the system to adjust its behaviour (focus its attention) more effectively [Rich 83]. In complex problems, the ability to focus the attention of the problem solver dynamically is imperative

because it is often impossible to specify the focus of attention a priori in these domains.

There are, however, several problems associated with agenda-based reasoning [Rich 83]. One such problem is the choice of the proper grain size for the division of tasks. If each task is very small, then the system will spend little time on productive work and a great deal of time deciding which task should be done next. Conversely, if the tasks are too large, then executing a task will require a large amount of effort while other more promising tasks may remain idle. This problem is inherent in all control strategies, but is particularly cumbersome in agenda-based systems because of the amount of resources expended by the control component of these systems. A balance in task size is required in order to enable the problem solver to make the best use of its time and resources. This often requires experimentation, and is a problem-dependent decision.

In addition, agenda-based reasoning is inappropriate for some problems for which the justifications for executing tasks changes very rapidly. In such problems, the elements of the agenda are applicable for very short periods of time, after which they must be purged. The problem in these domains is that it is often very difficult to determine when a task is no longer useful because explicit negative justifications are not prevalent [Shapiro 87, p. 4].

2.4 KNOWLEDGE REPRESENTATION

This section examines the principles involved in knowledge representation and discusses three basic types of knowledge representation techniques. This is not intended to be an exhaustive examination of the area, rather it serves as an overview of the concepts involved in knowledge representation and of some of the strategies that have been developed to facilitate the representation of knowledge in computer programs.

A representation consists of a language for describing things and a formalism for physically encoding the descriptions [Fischler 87]. Consider a natural language such as English. The language is a representation in which words are used to describe things. The vocabulary of words and the grammatical rules of the language enforce constraints on the ways in which things may be described. The meaning of a statement in the language is determined by the meaning of the individual words in the vocabulary and the rules of the language that define how the meanings of words are combined to form composite meanings. The written and spoken forms of the language constitute the formalisms in which the descriptions can be encoded.

A problem-solving model must include a representation formalism that enables the knowledge of a domain to be encoded. In effect, a model must provide a mechanism for representing problem descriptions in a computerized form. The study of knowledge representation involves examining the characteristics of various types of knowledge and developing formal methods for organizing, structuring, representing, and interpreting them. In conjunction with knowledge manipulation it provides the basis for computerized problem solving.

A knowledge representation is a set of syntactic and semantic conventions that make it possible to describe things [Shapiro 87, p. 882]. The syntax of the representation specifies the symbols that may be used and the ways in which those symbols may be arranged. The semantics of the representation specify how meaning is embodied in the symbols and symbol arrangements allowed by the syntax. Together, these conventions enable the representation of domain objects and operators in a physical symbol system.

Symbols and symbol structures may be used to designate specific objects and specific facts about those objects. The semantic conventions specify how to determine the objects and facts that are designated by symbols and symbol structures. Symbol structures can also be used to denote operators that can manipulate symbols or symbol structures (representing objects and facts) through creation, modification, reproduction, and destruction. Again, the semantic conventions must specify how to interpret the symbols and symbol structures representing operators in order to determine the operations to be performed. The process of creating a knowledge base therefore consists of using representation conventions to construct a collection of symbols and symbol structures that represent the objects and operators of a domain. The characteristics of the representation component of a problem-solving model influences significantly the types of knowledge bases that can be created using the model.

There are several standard representation paradigms, any of which may be used alone or in conjunction with others to construct knowledge systems [Evans 87]. In the following sections I briefly describe three

of the more popular techniques, including a brief examination of the advantages and disadvantages of each. The reader should note that this is by no means a complete survey of all the paradigms available or in use today, nor is it an attempt to describe all of the details of the paradigms presented.

2.4.1 Propositional Representations

The typical basis for propositional representations is first-order predicate logic [Nilsson 80]. Knowledge is represented in predicate logic using constants, variables, predicates, and connectives. Constants are used to represent specific domain objects, while variables represent sets of objects belonging to some group. Predicates are used to associate properties with constants and variables, and connectives enable composite statements to be expressed (e.g. implications, conjunctions, etc.). In addition, logic provides conventions for representing universally and existentially quantified statements (e.g. all birds have feathers; some birds can fly). Reasoning in predicate logic is carried out through applications of rules of inference to a set of statements in order to generate new statements (i.e. deduce or infer facts). Examples of statements represented in predicate logic and of deduction using rules of inference are shown in Figure 2.8. An implementation of a predicate logic typically provides a general-purpose control strategy (e.g. resolution refutation), a set of operators (rules of inference), and conventions for representing knowledge bases of user-defined domain objects, facts, and implications.

Fred is a bird = **bird (fred)**

Fred is green = **color (fred, green)**

Fred can fly = **canfly (fred)**

(a) representing facts in Predicate Logic

forall (X): bird (X) --> has-feathers (X)
- x is a bird implies x has feathers
(Universal quantification)

exists (X): bird (X) and canfly (X)
- there exists at least one bird which can fly
(Existential quantification)

(b) representing quantified statements

A, A --> B therefore B (modus ponens)

given: **bird (fred)**
forall (X): bird (X) --> has-feathers (X)

deduce: **has-feathers (fred)**

(c) Deducing new facts using a rule of inference

Figure 2.8: Representation in Predicate Logic

Although predicate logic provides a representation that is applicable to many problems, it suffers from computational inefficiency [Hayes-Roth 83]. As a result, predicate logic is useful

for theoretical research where illustrating that a given problem can be solved is more important than constructing a viable implementation of the solution. Furthermore, the syntax and control strategies employed by predicate logic produce a representation that is very unnatural when viewed from a human's perspective. As a result, envisioning how a problem should be represented and actually implementing it become very tedious tasks.

Although predicate logic has major flaws computationally, restrictions may be imposed on it in order to produce a more feasible paradigm while retaining much of its expressive power and generality. The rule-based paradigm is a restricted, yet more natural, form of predicate logic. A rule-based paradigm consists of a rule set (IF condition THEN action statements), a rule interpreter that describes how and when to apply which rules, and a working memory that holds data, goals, and intermediate results [Jackson 86]. Knowledge is encoded by creating rules that represent the actions to be performed when specific conditions are met (see Figure 2.9). Each rule theoretically represents a single piece of knowledge in the problem domain. The current problem state is represented as facts in the working memory.

The reasoning process consists of having the rule interpreter determine the rules to apply based on the current facts. When the IF part of a rule is satisfied (matches the facts in working memory), then the action specified by the THEN part of the rule can be performed. This process is repeated until either a goal is satisfied or the rule interpreter determines that a conclusion cannot be reached (this is the algorithm for forward chaining, another search strategy known as

IF X is a bird
THEN X has feathers

(a) a rule representing the birds-have-feathers implication

IF temp (X) > 100
THEN fever (X)

IF fever (X) and
nauseous (X)
THEN flu (X)

(b) rules for inferring fever and flu conditions

Figure 2.9: Sample Rules

backward chaining moves from the goal state towards the initial state [Brownston 85]).

The rule-based paradigm provides a natural and flexible means for describing domains that are highly data-driven; a set of rules can specify how the system should react to changing data without requiring detailed knowledge in advance about the flow of control. Rather than predetermining when a rule should be applied, the rule interpreter can use the rules to examine the state of the problem at each step and react appropriately.

In addition, because the domain knowledge is represented separately from the technique used to solve problems, the control strategy (rule interpreter) is general purpose. Therefore, tackling a new domain only

requires the creation of a ruleset describing the new domain, while the control strategy remains unchanged. Furthermore, because each rule, in theory, forms an individual piece of knowledge, the knowledge base is more modifiable and easier to extend - in order to add new knowledge one simply creates new rules [Waterman 86].

Rule-based representations are also well-suited to representing heuristic knowledge. Many rule-based representations provide mechanisms for including confidence or certainty factors in the rules, representing the certainty associated with the knowledge which the rules represent [Buchanan 84]. In addition, certainty factors often can be associated with working memory facts to enable the rules to manipulate uncertain data [Buchanan 84].

The disadvantages of the rule-based paradigm are inherent in the flexibility of the representation. One of the major disadvantages is inefficient program execution. The interpretive nature of the control strategy results in high overhead during problem solving. Because rule-based systems perform every action by means of a match-action cycle, much of their time is spent determining which rules match the working memory. This overhead increases exponentially as the number of rules in the knowledge base increases. Moreover, because of the rigid match-action cycle, it is difficult to make rule-based systems take larger steps in their reasoning when the situation warrants it [Jackson 86].

Another disadvantage of the rule-based formalism is that it is often very difficult to follow the flow of control during problem solving -

algorithms are implicitly structured in the rules and are therefore less apparent than if expressed in a procedural representation. Because each rule contributes to the overall flow of control depending on the current problem state, it is extremely difficult to predict all the possible control situations that a given rule may contribute towards [Evans 87]. Consequently, although each rule, in theory, represents an independent piece of knowledge, the modularity of rules is reduced because the application of a rule can have significant effects on other rules and on the overall reasoning process. In other words, the rules are really not independent, and the addition of new knowledge is not as simple as creating new rules. One must take into account the global effects that new rules might have on the overall system. The use of explanation facilities which provide traces of the rules used to solve a particular problem helps to alleviate this difficulty to some degree. However, the onus is still on the developer to test the system thoroughly in order to validate each rule and each control situation that a rule can contribute to. This caveat is a direct result of the lack of an explicit contextual mechanism in the rule-based formalism. The capability to incorporate contexts must be built on top of the formalism such as in the rule-based expert systems MYCIN [Buchanan 84] and XCON [McDermott 81].

2.4.2 Structured Representations

Structured representations provide a means for explicitly linking related information together (objects, attributes, and operators). A semantic network is a popular structured representation scheme which facilitates the representation of objects, facts describing objects,

and, most importantly, relationships among the objects [Quillian 68]. The term semantic network is used to describe a knowledge representation scheme based on a network structure. In general, a semantic network consists of points called nodes connected by links that describe the relations between nodes. Knowledge is encoded as graphs in which the nodes represent entities (objects, concepts, events) and the links represent the properties of entities and relations among the entities (see Figure 2.10).

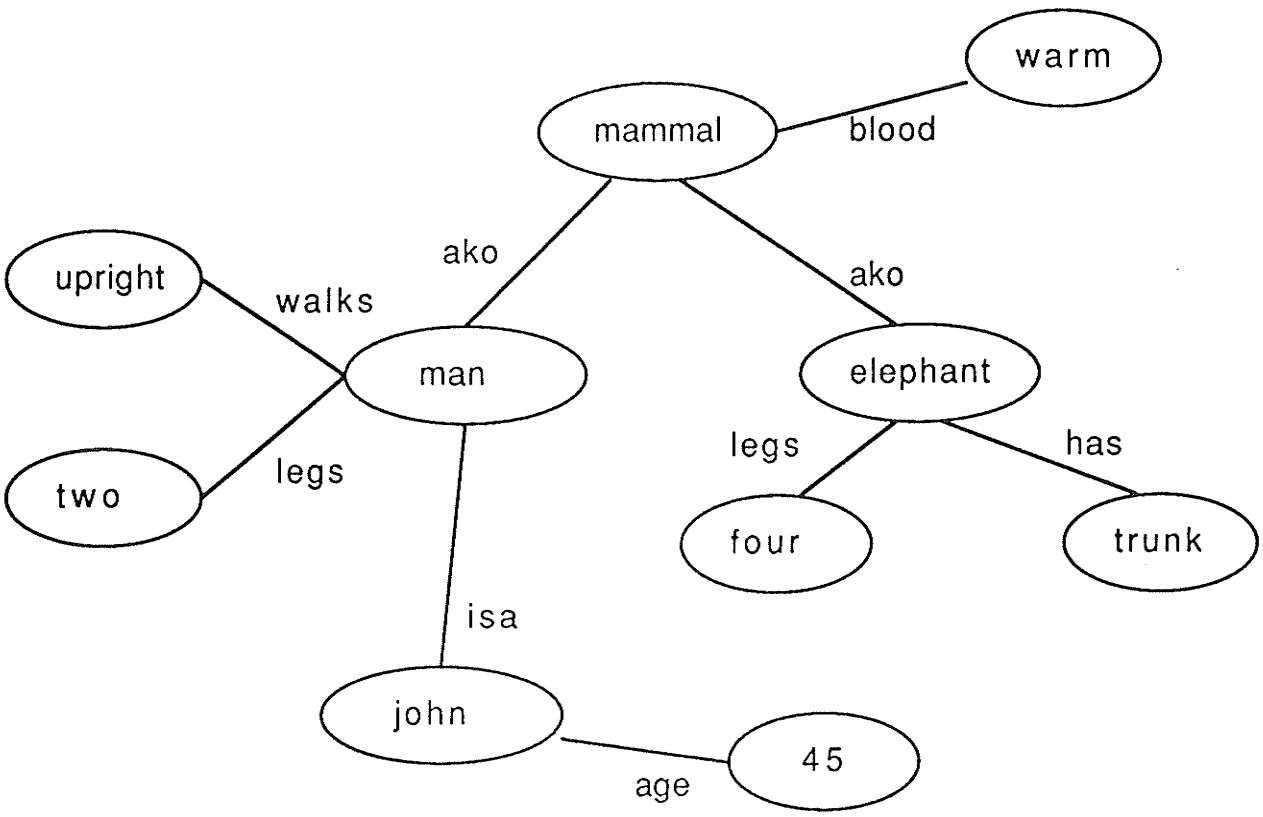


Figure 2.10: A Semantic Network

In contrast to conventional networks, the links in a semantic network embody meaning such as the functional or declarative attributes of a node. In addition, two special links AKO (a-kind-of) and ISA (instance-of) are used to support the most important principle of most structured representations - the power of inheritance. These inheritance links enable the construction of structured taxonomies in which entities are organized based on class and subclass relationships. As a result, knowledge can be stored in the most appropriate place in the network and inherited when needed by subordinate entities. For example, the semantic network in Figure 2.10 represents the entities MAMMAL, MAN, ELEPHANT, and JOHN. MAMMAL is the parent class of both the MAN and ELEPHANT subclasses (AKO links), and JOHN is an instance of the MAN subclass (ISA link). Information common to all mammals is represented as links from the MAMMAL node, while information specific to MAN, ELEPHANT, and JOHN is represented as links from the corresponding node. As a result, if we needed to know whether or not JOHN has warm blood, then the node for JOHN can inherit the required information from the MAMMAL node by first following the ISA link to the MAN node, and then following the AKO link from the MAN node to the MAMMAL node where the required information is stored.

A frame-based paradigm is another structured representation scheme. Whereas semantic networks are a collection of nodes representing individual entities and links representing the relationships among the entities, frames provide a means of grouping related information into data structures consisting of slots and fillers. Each data structure can be thought of as a complex node in a network with a special slot to be

filled with the name of the entity the node represents and other slots being filled with the values of various attributes associated with the entity [Fikes 85]. A frame-based scheme is actually a different conceptual approach to organizing knowledge than a semantic network. However, the two approaches are both implemented in a similar fashion - the links and their meanings in semantic networks are akin to the slots and their values in frames. Conceptually, the links and nodes in the semantic network are combined into slots and fillers in larger, frame-like nodes (see Figure 2.11). The frame-based scheme also employs an inheritance mechanism through the use of special AKO and ISA slots.

A major advantage of structured representations is the space saving achieved because information shared by nodes is not repeated at each node. Instead, shared information can be stored in parent nodes and inherited as required by subordinate nodes. Such space savings can be huge in problem domains that have large amounts of inheritance and many instances of each class and subclass [Cercone 83]. In addition, most structured representations provide mechanisms for representing default knowledge in class nodes which can be inherited by subordinate nodes. Furthermore, most structured representations provide mechanisms enabling subordinates to override all or some of this inheritance in order to represent exceptions [Bobrow 86].

By structuring the concepts of a problem domain into a taxonomy, complete with inheritance capabilities, it is possible to reason at various levels of abstraction. This is a result of the explicitness of the representation; all possible paths to more general or more specific instances of a given concept are represented by inheritance links/slots.

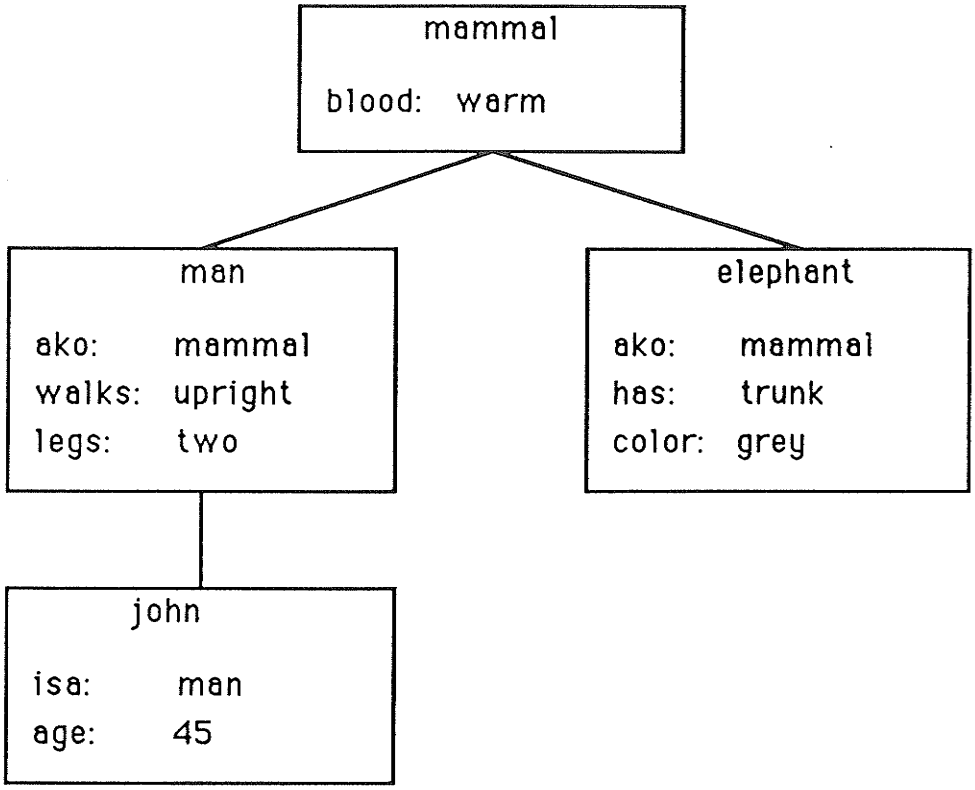


Figure 2.11: A Frame-Based Network

Therefore, if the problem requires more specific reasoning, the taxonomy supports the direct retrieval of the subclasses related to the class currently being used. The opposite is also true: more information about a concept can be retrieved by traversing up the inheritance links to more general class information.

Many structured representation schemes support the attachment of procedural knowledge to the nodes and links/slots. The procedures

enable the nodes to be utilized as event-driven processors rather than as passive data structures [Brachman 85]. The domain knowledge can be structured into modular, self-activating nodes with each node having procedures attached to it that indicate how and when the node should react when the information it contains is modified, extended, or deleted [Waterman 86].

Although structured representations provide many advantages, there are drawbacks in their use. The first disadvantage is inherent in the network structure of the schemes; because entities are linked together the number of interdependencies among the entities will be large, resulting in reduced modularity. The conciseness of the scheme often makes both the acquisition of new knowledge and the modification of existing knowledge difficult. This difficulty increases dramatically when working with a large taxonomy with many levels of inheritance - a change to one node in the taxonomy can affect any number of related nodes (facilities can be added to help alleviate some of this difficulty [Davis 82]).

Another drawback of structured representations is the loss of simplicity in the representation. Because the formalism enables procedures to be stored with the data, much of the reasoning process may be represented in procedures distributed throughout the nodes. As mentioned, this provides dynamic, context-sensitive data structures which is advantageous. However, encoding knowledge in program code will significantly hinder its accessibility which in turn will limit its ease of modification. Moreover, the decision of where to place the procedures and what the procedures should do is left up to the designer, a decision that can have a great bearing on the success of the system.

2.4.3 Multiple Representation Paradigms

Each of the representations we have examined concentrates on encoding a particular type of knowledge. For example, the rule-based paradigm attempts to represent knowledge that is modular and dynamically applicable, whereas structured representations attempt to represent knowledge that is highly structured. In doing so, each scheme cannot effectively represent knowledge other than the type for which it was designed. For instance, the rule-based paradigm cannot represent taxonomic knowledge as effectively as structured representations, and structured representations cannot provide situation-action processing and modularity as effectively as rule-based approaches. Consequently, knowledge that does not 'fit' into the chosen scheme must be forced into it, which often results in a drastic reduction in the effectiveness of that knowledge.

Most real-world applications require the representation and interaction of several types of knowledge in order to solve problems. However, there is no single representation available that is powerful enough to represent all facets of the various types of knowledge [Buchanan 84]. Therefore, researchers have turned to another approach - multiple representation paradigms. In this approach, a representation scheme is built from various standard schemes in order to enable each different type of knowledge to be represented using the representation that best suits it. The flexibility and expressiveness of the combined formalism increases because the knowledge is not overly constrained by a single representation. As well, different types of knowledge can be separated explicitly which helps in applying the

knowledge appropriately [Aikins 83]. The underlying goal of multiple paradigms is to take advantage of the powers of various representations in order to reduce the limitations of the combined representation.

An example of a multiple paradigm system is CENTAUR [Aikins 83], an expert system for diagnosing pulmonary disease which combines a frame-based paradigm with a rule-based paradigm. The system consists of frame-like structures that provide an explicit representation of the context in which the rules are to be used. This allows the separation of strategic knowledge, indicating how to control the reasoning, from situational knowledge describing the inferences that can be made from given facts. As a result, the inferential knowledge embodied in the rules can be put to different uses in different contexts, providing an increase in the modularity, economy, and coherence of the overall representation.

Although the multiple paradigm approach provides numerous advantages, inevitably the approach suffers from several drawbacks. The first drawback is the additional complexity that incorporating multiple paradigms produces. By allowing several representations to be employed, we leave the designer with the difficult task of deciding how to separate the knowledge into the various schemes and organize these schemes into integrated units. Because there are no well-defined guidelines for employing representation schemes and because of the dynamic nature of knowledge itself, it is very difficult to decide how to organize the domain knowledge into an effective combination of paradigms [Hayes-Roth 83]. On top of this added complexity, there is also the problem of determining how the chosen schemes are to be

controlled during problem solving [Nii 86]. In most cases, the underlying representations all share one major control strategy responsible for co-ordinating the entire reasoning process. Employing a single control strategy drastically reduces the effectiveness of the representations, especially in applications dealing with very complex domains [Evans 87a]. As a result, although no single paradigm is powerful enough, current multiple paradigm schemes suffer from several deficiencies which reduce their effectiveness.

2.5 THE ROLE OF METAKNOWLEDGE IN KNOWLEDGE SYSTEMS

The techniques examined in the previous sections enable the construction of special-purpose agents that can exhibit limited forms of intelligent behaviour. The intelligence attributed to these systems is based on the results that they are able to produce, not the methods used to attain the results. The systems are deemed intelligent because they are able to generate solutions to very difficult problems. However, the ability to generate solutions relies heavily on the knowledge that has been "programmed" into the systems. The lack of any substantial intelligence, in particular the lack of self-knowledge, increases the onus on the human developers in maintaining, extending, and employing the systems. Hence, as the complexity of potential applications increases, these techniques become increasingly less applicable; humans supply the intelligence in the programming techniques employed in the systems and, as the systems become larger and more complex, it becomes increasingly difficult for humans to maintain an understanding of the internal workings of the system.

Metaknowledge, which is knowledge about knowledge, offers a partial solution to this difficulty. Metaknowledge increases the intelligence embodied in a knowledge system by providing the system with knowledge about itself - its components, their capabilities, functions, and roles, and ways in which the components can be employed to solve problems effectively. Thus, the system is able to assume more of the burden of understanding its own behaviour, documenting and justifying itself, and even modifying itself [Hayes-Roth 83]. The incorporation of large amounts of metaknowledge into knowledge systems can greatly enhance the robustness and efficiency of a system. Figure 2.12 illustrates a knowledge system which contains a metaknowledge component consisting of a metaknowledge base and a metacontrol strategy for controlling the application of the metaknowledge.

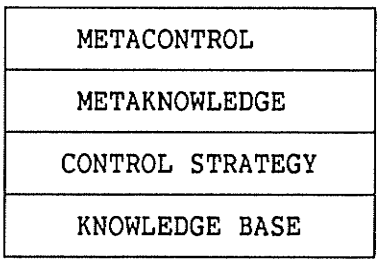


Figure 2.12: Knowledge System Embodying a Metaknowledge Component

The metaknowledge discussed above falls into two categories: (1) task metaknowledge that describes and justifies the domain knowledge that the system possesses, and (2) control metaknowledge that describes the best approaches to solving various tasks for which the system is

responsible. Task metaknowledge enables a system to determine whether or not it can solve a given problem. As well, it provides the system with justifications for its knowledge and for its reasoning decisions. The task metaknowledge and control metaknowledge can be represented by facts, procedures, frames, and rules. An example of the latter case is illustrated in Figure 2.13 which contains sample meta-rules from a spill advisor system [Hayes-Roth 83]. The rules in this example include justifications for the application of the operations that can be used to neutralize a chemical spill.

```
R6 If spill is sulfuric acid
   Then  use lime.

 Justification: Lime neutralizes acid
        and the compound that forms is
        is insoluble and hence will
        precipitate out.

R7  If spill is acetic acid
    Then use lime.

  Justification: Lime neutralizes acid.

R8  If spill is hydrochloric acid
    Then use lime.

 Justification: Lime neutralizes the acid.
```

Figure 2.13: Sample Meta-Rules For Justifying Actions

Control metaknowledge represents knowledge about the system's problem solving strategies. It enables a system to determine the best method (control decisions and appropriate knowledge) to use in solving a given problem, and it enables a system to reason about its own reasoning process and choose the best (effective/efficient) solution paths

R3 Use rules that employ cheap materials before those that employ more expensive materials.

R4 Use less hazardous methods before more hazardous methods.

R5 Use rules entered by an expert before rules entered by a novice.

Figure 2.14: Sample Meta-Rules for Controlling Reasoning

available. Figure 2.14 illustrates a set of meta-rules from the spill advisor system that represent metaknowledge describing the cost, danger, and so on, of the available ways to neutralize a spill (i.e. reason about the application of other rules).

The above examples of metaknowledge are not the only examples of self-knowledge that may be incorporated into knowledge systems. A more in-depth analysis of metaknowledge and the various forms it may assume is found in [Davis 82].

2.6 PARALLEL PROCESSING

The computational resources required to run large AI systems often results in poor performance, making the systems impractical for everyday use. Much of the problem surrounding execution speed is that the conventional serial machines currently being used for AI systems are poorly suited to AI applications and techniques. AI problem solving requires extensive search, and search on a conventional computer must be carried out sequentially; the nodes of a problem space are examined one after another until a solution is found. Large-scale AI systems often have very large problem spaces, resulting in time-consuming searches which frequently make the problem in question intractable.

One way around this problem is to use heuristics to guide the search and thus prune the problem space. For many problems, however, effective heuristics do not exist and much search is still required. An alternative approach is to use computers designed with multiple processors. Unlike conventional computers in which the processing powers are harnessed in one large processor, these machines distribute the processing resources across several, possibly thousands, smaller processors (e.g. Connection Machine [Hillis 85]).

Parallel processing machines are designed to facilitate concurrent computation and are well-suited to large problems which can be divided into collections of smaller problems which are distributed to, and solved by, individual processors. For example, the search of a very large problem space can be carried out by distributing the problem space across several processors which search their respective portions in parallel. As a result, the entire problem space can be examined in a fraction of the time required on a conventional machine [Gelernter 87].

Most parallel architectures simply attempt to provide increased execution speed for algorithms (problem-solving techniques) that are well-suited to parallel processing. However, another class of parallel processing paradigms known as connectionist or neural architectures attempt to provide alternative forms of problem-solving, much like that employed by the human brain [Hecht-Nielsen 88]. In these architectures, several very simple processing elements are organized into massively interconnected networks and communicate by exchanging messages across these connections. Each processing element operates on several inputs and produces output signals that fan out along many pathways to provide

input signals for other processing elements [Tank 87]. The connections among the processing elements are weighted so as to increase, decrease, or inhibit input signals received along the respective connections.

These neural systems are not necessarily programmed, rather they may be trained [Eliot 87]. One such training approach employs a supervised technique in which the network is presented with a given input and the desired output. The input propagates through the network of processing elements and an output is produced which is then compared to the desired output. Corrections are then made using a learning algorithm which modifies the strengths of various connections among the processing elements [Wasserman 88]. The training process is repeated until the network produces an acceptable output, at which point the weights of the connections throughout the network are frozen. The network will then respond with the desired output when presented with the appropriate input. This form a problem-solving is known as memory-based reasoning. A more extensive description of memory-based reasoning can be found in [Stanfill 86].

Systems of this sort show promise in solving problems in the areas of pattern recognition, associative memory, and image processing, largely due to their adaptive nature and their ability to produce correct outputs when presented with incomplete or error-filled inputs [Shapiro 87, p. 201].

2.7 SUMMARY

AI is the study of principles, techniques, and methodologies for dealing with **HARD** problems for which there is a large space of possibilities (solution space) that must be searched to find a solution. Such problems are beyond the scope of conventional DP techniques because they are too complex and resist precise algorithmic analysis or because potential algorithmic solutions are too costly, time-consuming, or inconvenient. Instead, we require an intelligent problem-solving method which employs a search strategy guided by domain-specific knowledge in order to reduce the amount of search that must be performed to solve a problem. The intelligence exhibited by such methods is not a measure of the amount of search being done, rather it is a measure of the amount of search that would be required if an intelligent method was not used [Newell 76].

I examined the factors that influence the development of these knowledge-based techniques and briefly described some of the paradigms that have been created to represent and manipulate knowledge in computer programs. In addition, several advanced techniques that attempt to augment the amount of intelligence that is embodied in the systems were discussed.

Chapter III

DISTRIBUTED PROBLEM SOLVING

3.1 OVERVIEW

This chapter presents a general overview of the principles employed in distributed problem solving. I begin with a brief discussion of the shortcomings of the AI techniques examined in Chapter 2, followed by an examination of distributed problem-solving techniques and the potential role these techniques play in the development of enhanced AI systems. This examination includes a discussion of the fundamental concepts that underlie distributed problem-solving systems and a description of the attributes that an ideal distributed environment should embody. This is followed by an analysis of the role that interaction plays within distributed problem-solving systems and a discussion of several factors that influence the ways in which multiple agents can interact to solve problems co-operatively. The remainder of the chapter provides a brief survey of various paradigms that have been developed to facilitate the construction of distributed problem-solving systems.

3.2 INADEQUACY OF THE CLOSED-WORLD ASSUMPTION

Chapter 2 examined the methodologies employed in the majority of current AI systems. These methodologies rely on what is known as the closed-world assumption [Hewitt 85]: all of the information that a system requires to solve problems can be derived from local knowledge.

In other words, the systems operate under the assumption that the locally available knowledge is complete and therefore any inference that does not follow from the local knowledge is assumed to be false. The closed-world assumption is theoretically valid, but in practical applications it is not possible to provide a single agent, or for that matter a group of agents, with a complete and consistent knowledge base. Massively parallel architectures help to increase the processing powers of single-agent systems, but do not significantly reduce the complexity inherent in the amount and variety of knowledge that is necessary to perform numerous difficult tasks. Even the human brain is limited in the amount of information it can retain and manage effectively, and, as a result, many problems are too complex to be solved by a single human problem solver [Axelrod 84].

3.3 DISTRIBUTED PROBLEM SOLVING OFFERS A SOLUTION

Agents operating under the closed-world assumption do not even consider the possibility that the scope of their knowledge may be incomplete. Hence, these agents operate under the false assumption that they need not look beyond their own knowledge in order to solve problems. Distributed problem-solving research involves supplementing closed-world techniques with a methodology that concentrates on creating intelligent systems from collections of special-purpose agents. In effect, this research examines techniques that enable several special-purpose agents to be integrated to form a co-operative problem-solving environment [Sridharan 87]. Each special-purpose agent is capable of performing specific tasks within the environment. These agents are then organized into groups or networks and are allowed to interact with one

another. The agents work individually (concurrently) for the most part, and interact when they require information which is not available locally, but which other agents in the environment can provide. The agents are able to look beyond the scope of their own knowledge when the situation warrants it in order to cope with the complexities inherent in very difficult problems.

The ability to organize several agents into groups and the ability to enable the agents to interact represent the fundamental principles in distributed problem solving [Georgeff 83]. Without the ability to interact, individual agents are unable to take advantage of the resources and capabilities of other agents and are restricted to solving problems using only their own abilities. When allowed to interact with other agents, however, an agent is able to augment its own abilities with those of the other agents in the group. Consequently, each agent has access to more resources which in turn increases its ability to contribute to the group as a whole. This represents the theory that a team of agents working co-operatively will be more productive than several individual agents working independently [Lesser 79].

Consider the corporate example presented in Chapter 1. A company is a structured organization of employees who for the most part work independently. The employees may also interact with one another to function as team. If, however, the employees were restricted to functioning as individuals who could not communicate, then each employee would only be able to perform tasks using his own abilities. In such a scenario each employee must perform his assigned tasks without aid from any of the other employees. Employees, however, rarely are able to

perform all of their required functions without assistance. For example, an employee in the accounting department often relies on employees in the computing department to collate and produce reports of financial data; otherwise each accounting employee must perform both the accounting duties and the computing duties which would require each employee to have a vast amount of knowledge and skill in a variety of fields. Rarely do we find individuals who possess such diverse capabilities. Furthermore, a large amount of duplicated processing would be performed by each employee because the results produced by an employee could not be shared with any of the other employees.

It is more cost-effective and productive to have employees working on separate subsets of the company's workload and enable the employees to interact with one another to function in a co-operative manner. When an employee cannot satisfy some portion of his assigned tasks locally, then he may interact with other employees to complete the task co-operatively. Creating a co-operative environment of this sort is invariably more productive and represents the fundamental goal of distributed problem solving [Cammarata 83]. Note: several agents that compete against one another may be less effective than a single agent, thus co-operation among the agents is a fundamental requirement of the distributed problem-solving systems examined in this thesis.

3.4 OPEN SYSTEMS ARE IDEAL DISTRIBUTED ENVIRONMENTS

3.4.1 Synergy in Open Systems

In what shall undoubtedly become a landmark paper, Hewitt predicts that the next generation of computer applications will be based on communication among subsystems that will have been developed separately and independently [Hewitt 85]. The economics of the times, the vast geographical distribution of systems and users, and the diverse goals and responsibilities that computer systems in the future will embody are the underlying reasons necessitating what they call "open systems" [Hewitt 85]. An open system is an ideal distributed environment that is comprised of a slowly increasing collection of semi-autonomous, communicating parts called sites which operate in parallel.

Each site has its own conceptual model of the environment, both locally and globally. The model of the local environment consists of knowledge describing the functions the site can perform locally, while the model of the global environment consists of partial knowledge of the other sites in the environment and the methods that can be used to interact with those sites. The sites must interact when they face tasks that they cannot carry out on their own, but which may be carried out in co-operation with some other sites in the global environment. Consequently, sites must co-operate to perform the tasks that are the responsibility of the open system as a whole. Goals, tasks, and responsibilities are distributed across multiple sites which must co-ordinate their interactions in order to contribute towards global solutions and actions.

3.4.2 Requirements of An Open System

An open system is formed from the reservoir of processing resources embodied in its member agents and the processing resources that are inherent in the potential interactions among these agents. Nevertheless, simply grouping a set of agents into a distributed environment is not sufficient to attain an open system. Hewitt defines four major attributes that an open system must possess: continuous change and evolution, decentralized decision making, individual perspectives of the environment, and negotiation among system components [Hewitt 85]. In the remainder of this section I examine these properties and the contributions that each makes towards realizing an effective distributed problem-solving system.

One of the most important characteristics an open system must possess is the ability to evolve continuously through changes and extensions to its member agents and their ability to interact. Moreover, an open system must be able to incorporate additional agents and integrate their interactions with existing agents in a smooth and orderly manner. Thus, an open system is an open-ended environment in which additional agents may be added, existing agents may be extended or modified, and the ability of the agents to interact may also be extended and modified. The most important aspect of this openness is that extensions and modifications must not impose undue burden on any single component in the system. The responsibility for modifying and extending the environment must be distributed across all of the agents in the system. Thus, the ease with which the agents and their interaction capabilities can evolve plays a central role in open system technology. When a new

agent is added or an existing agent is modified, the other agents in the system must adjust their behaviors appropriately to take advantage of the additional resources for their own benefit and, more importantly, for the benefit of the system as a whole.

In addition to being open-ended, an open system must function using decentralized decision making. No single agent or group of agents may control all of the facets of the system's processing functions. Rather than denoting a single controller, control in an open system is a product of the actions of its member agents and the interactions between the agents [Hewitt 77]. Agents are autonomous entities which operate concurrently and which control their own local actions. Global control is produced as a result of co-ordinated interactions among the agents. The interactions are in the form of communications between agents rather than one agent directly accessing the internals of another. Consequently, an agent can obtain assistance from another agent without imposing undue control over the operation of that agent (an agent must request that another agent perform some task for it). Furthermore, the agent does not need to know how a communication will be processed by the destination agent, it need only know the format and content of the communication that will invoke the desired response. In this scenario, it is extremely important that agents be capable of accepting communications from sources that are not anticipated because it is impossible to program all anticipated interactions into a system. Furthermore, attempting to program all anticipated interactions would restrict the ease with which the system could evolve because all of the agents affected by a modification would have to be "reprogrammed" appropriately.

Another important characteristic that must be embodied in an open system is the ability to retain the individual conceptual models of each member agent. Each agent must reflect its own perspective of its local functions and its roles and functions in the global environment. Maintaining individual perspectives may result in conflicts or inconsistencies between portions of the conceptual models of the agents in the system. Nevertheless, inconsistent or conflicting perspectives are not meaningless, rather they play a major role in the reasoning processes employed in solving real-world problems [Hewitt 84]. Rarely does a problem solver or a set of problem solvers have all of the information that is required to produce an infallible solution. Instead, reasoning must take into consideration evidence for and against potential solutions from different perspectives. This type of reasoning is known as due-process reasoning and represents an important form of co-operative problem solving that is ideally suited to open system technology [Hewitt 85].

Negotiation is the major principle underlying interactions between agents in an open system. In order to avoid master-slave control situations it is imperative that agents interact based on mutual negotiation. Interactions among agents with varying or diverse perspectives cannot rely on predefined agreements because that would impose undue constraints on the contributions an agent can make to global problem solving [Hewitt 84]. Instead, the agents must be able to negotiate dynamically mutually beneficial interactions based on their local needs and the needs of the global environment. These needs are represented by each agent's conceptual model of its local and global

goals and responsibilities. Furthermore, the agents must have knowledge about each other's needs either represented locally or exchanged between the agents during the negotiation process. The ability to negotiate plays an important role in achieving a powerful synergy of the agents' abilities because agents are not restricted to all-or-nothing interactions. Instead, interactions which may normally be incompatible can be negotiated to find a common ground.

3.5 INTERACTION IS THE CENTRAL THEME IN DISTRIBUTED SYSTEMS

3.5.1 Overview

I have presented the attributes of an ideal distributed problem-solving system in which a synergy of the abilities of a group of individual agents is formed by enabling the agents to interact and function as a team. The agents' ability to interact through communication is fundamental in constructing effective distributed problem-solving systems. Interaction provides the basis for producing synergy among the member agents which opens access to processing powers in the system which would otherwise remain untapped. As a result, the environment not only contains the sum of the resources of the individual member agents but it also contains the resources formed through interactions among its agents.

3.5.2 Communication

A communication involves three layers of information: the frame message, the outer message, and the inner message [Hofstadter 79]. The syntax and semantics of a communication mechanism must embody these three layers in order to facilitate inter-agent communication. The

frame message is implicit in the gross structural aspects of a communication. In other words, it indicates that a communication is present and needs to be decoded. Without a frame message it would be impossible for agents to recognize the presence of communications. The outer message is represented implicitly by the symbols and symbol structures that make up a communication. The agent sending a communication must ensure that the outer message can be understood or decoded by its recipients. Consequently, agents must agree on a standard set of symbols and symbol structures that may be used to represent communications.

The final layer, the inner message, represents the message the sender intends the recipient to receive. The recipient must interpret the outer message in order to determine the essence of the communication. Once the inner message is interpreted, the recipient then initiates processing, which may entail additional communications with other agents, to fulfill the processing requests contained in the communication. There is no guarantee, however, even if the inner message is successfully decoded, that the recipient can satisfy the processing requested in the communication. When a message cannot be processed, some form of negative acknowledgement must normally be sent to the source agent, possibly containing a reason for the communication being rejected (this may lead to negotiations among the agents in order to reformulate the communication appropriately). A more extensive discussion of communication protocols can be found in [Burkowski 88], [Kuo 81].

3.5.3 Interaction Through Communication

Interactions among agents are based on the conceptual models of each agent and the current state of the problem solving environment, both locally and globally. Consequently, the meaning embodied in a communication is determined by how it affects its recipients [Hewitt 77]. A recipient interprets the contents of a communication based on its conceptual model, and determines the local processing that must be performed and any additional communications that may be required. Consider an example in which an agent X sends a communication to an agent Y. As a result of the initial communication, the recipient (agent Y) may perform some local processing and determine that it must send communications to additional agents in order to have other required processing performed. These agents would perform the requested processing (which may also involve subsequent communications) and communicate the results back to agent Y which would combine the results and communicate a final result back to agent X.

This scenario illustrates that the meaning of a communication is determined by the conceptual models of each agent involved in the communication or any subsequent communications spawned from the initial communication. Meaning unfolds dynamically as communications are interpreted and processed. Moreover, the meaning of a communication is not necessarily determined by a single agent, often it is the composite interactions of several agents that determines the ultimate meaning of a communication [Hewitt 77].

An agent initiating an interaction does not have to know how a communication is processed by its recipients, it need only know the

content and format of the communication that will initiate the desired processing. An agent need only have abstract knowledge of the other agents in the system, enough to enable it to communicate effectively with those agents to request that they assist in performing given tasks. Consequently, communication often occurs at arms length and in an abstracted form. Nevertheless, in some circumstances the sender of a communication may dictate some control over how a communication is processed. For example an agent which is denoted as the manager of one or more subordinate agents may exercise greater control over a subordinate, but not to the point where a master-slave relationship is created. Such control of processing through interaction must, however, only be used for the benefit of the global environment, not solely for the benefit of a single agent. Therefore, interactions must still be based on negotiation and decentralized decision making, although the form and level of negotiation and decision making may vary.

3.6 FACTORS INFLUENCING INTERACTION AMONG AGENTS

3.6.1 Overview

The model used to co-ordinate interactions among several co-operating agents is a central issue in distributed problem solving. This section examines some of the other the factors influencing distributed problem solving (besides negotiation, task distribution, openness, etc.). It is a discussion of several issues that influence interactions among agents in a co-operative problem-solving environment. The factors, however, are not independent; instead, one must balance the effects of the factors in order to construct a model of distributed problem solving that facilitates effective interactions among agents in a distributed environment.

3.6.2 Grain Size

The grain size of the agents in a distributed system influences the amount and type of communication that is required to enable the agents to interact effectively. A fine grain system is comprised of numerous agents which are capable of very limited processing. As a result, these agents must communicate frequently to achieve the desired global effects. The agents operate in a tightly coupled environment in which they communicate small amounts of low-level information frequently. The information is considered low-level because for the most part it is raw data. No single agent is capable of significant processing, but when numerous agents work together closely on several small parts of a problem they are able to produce the desired synergistic results [Hewitt 73].

A coarse grain system, on the other hand, is comprised of only a few agents which embody large amounts of processing resources. These agents are capable of more autonomous behavior, and, as a result, are able to contribute more towards the global results than their fine grain counterparts. The agents operate in a loosely coupled environment in which the agents communicate less frequently than in a fine grain system. Communication among agents is still a vital part of producing a synergistic effect, but communications generally deal with higher-level information, summary or packaged information rather than raw data. Furthermore, the agents are able to perform more significant processing between interactions and are capable of understanding and interpreting communications containing high-level information [Durfee 85].

An alternate approach uses both fine grain and coarse grain agents to represent various parts of a system. In this scenario, the grain sizes of the agents are mixed, with each agent playing a particular role within the environment, however large or small that role may be. The agents are able to interact with any other agent in the system regardless of their grain size (unless restrictions are imposed on which agents can interact with which other agents). Consequently, the amount of interaction and the level at which interactions occur varies throughout the system depending on the agents involved.

3.6.3 Explicitness

The method used to facilitate communications among agents can have a significant impact on the effectiveness of a distributed problem-solving system. There are a variety of communication mechanisms available, but they may be grouped into two basic categories: explicit and implicit communication. An explicit communication mechanism requires agents to communicate by exchanging explicit communication packages, often called messages [Hofstadter 79]. A message is an object containing a communication that is to be sent from one agent (source) to one or more agents (destination). The source must indicate explicitly the address or addresses of the destination. The arrival of a message at a destination is an explicit indication that a communication has been received, and the sending of a message by a source is an explicit indication that an external communication has been initiated.

In contrast, an implicit communication mechanism employs a much less lucid approach. For example, agents which communicate using a shared

data structure utilize an implicit communication mechanism. Communication is carried out by agents posting or modifying information in the data structure and other agents recognizing when posted or modified information requires their attention [Nii 86]. Consequently, communication in this scenario relies on the agents' recognizing that a communication has taken place. In other words, the source initiates a communication and thus explicitly knows that an interaction has begun, but the destination must implicitly recognize the presence of a communication by periodically examining the contents of the shared data structure. Furthermore, the posted communications can normally be examined by any of the agents that have access to the data structure. This is often unacceptable because communications may contain sensitive information.

Communication carried out using procedure calls among agents may also be viewed as an implicit form of communication, although somewhat more explicit than the previous example. In this case, the agents must be part of a shared environment in which they may directly invoke one another using procedural calls [Moulin 87]. This can be viewed as a form of explicit communication if we consider that the destination explicitly receives indication that a communication is present (procedural call). Nevertheless, it may also be viewed as a form of implicit communication because the source may not be aware that an external communication occurred; it is difficult to differentiate between an external call which represents an inter-agent communication and a procedural call to a local subroutine within the source agent which represents local computation.

Employing an explicit communication mechanism greatly enhances the understandability and ease of modification of interaction capabilities of the agents, and it also increases the amount of global knowledge embodied in each agent. Maintaining an understandable and modifiable means of facilitating inter-agent communication significantly reduces the difficulty involved in controlling the complexity surrounding the creation and tracking of interactions among agents. Communications that may occur are represented explicitly as SENDs to external agents rather than determined implicitly by the actions of the various agents. Furthermore, the collection of external SENDs forms an explicit body of global knowledge making the agent aware of the various interactions it may perform. This is particularly important in a distributed environment because it enables the agents to reason about the interactions they can perform and thus plan interactions effectively [Corkill 83]. Furthermore, employing an implicit mechanism such as the shared data structure method may subject the agents to inconsistent or incomplete information while the information is in preparation. Explicit communication solves such problems by ensuring that messages are generated only when preparation of a communication is complete.

3.6.4 Focus

The focus of communication deals with the type of information that may be communicated among agents. The three basic categories of information are: objects, operations, and control information. Objects generally represent data that are transferred between agents, possibly as the result of some operation. Operations represent requests from an agent

to one or more other agents for the execution of a given operation (i.e. perform a task or solve a problem). Operations may be generic or agent-specific depending on the type of operation requested and the agents involved. Agents may also exchange control information such as priorities, interrupts, aborts, routing information, and general status information (an agent is busy, disabled, etc.).

An agent should be capable of sending and receiving all three types of communications, although the focus of communication may vary across agents; one agent may focus primarily on data transfers while another focuses on operations. More importantly, the focus of communication that an agent uses does not have to be prespecified, rather it may be determined dynamically based on incoming communications, the current status of the agent, and the agent's conceptual model. Therefore, when a communication is received by an agent, that agent can examine the contents of the communication and apply its conceptual model to determine the required actions.

A communication may embody or foster composite interactions in which several homogeneous or heterogeneous types of information are communicated. Consider an agent X sending an agent Y the message: 'Perform operation A and send the results to agent Z'. This message requests an operation be performed, but it also sends control information indicating that the results of the operation should be routed to agent Z. Initially, agent Y must focus on receiving and interpreting the designated operation, but once the operation has been received, interpreted, and executed, agent Y must focus on communicating objects that represent the results of the operation. Hence, the focus of communication often is a product of the interactions between agents.

3.6.5 Structuring of Interactions

In order to create a distributed environment one must organize several individual agents into a topology and enable the agents to interact. So far, however, I have assumed that interactions among agents in the environment are unstructured; there are no stipulations or protocols imposed on how interactions may be carried out. Thus, any agent can communicate directly or indirectly with any other agent in the system. Often, however, it is advantageous to use protocols which govern the ways in which agents can interact [Smith 79]. Enforcing protocols provides increased explicitness and greater control over interactions which serves to increase the understandability and maintainability of the system. One must be careful that the protocols are not too stringent as to overly constrain the decentralized decision-making process. Some bureaucracy is required to manage a large collection of interacting agents, but too much bureaucracy can defeat the purpose of distributed problem solving.

Consider the example in Figure 3.1 which illustrates a hierarchical environment. Agents are organized within the hierarchy based on their roles and functions and their relationships with other agents. Agents at the top of the hierarchy embody high-level functions and act as co-ordinators for their subordinate agents (but not controllers). As we move down the hierarchy the agents have more specific functions and less responsibility for co-ordinating the actions of other agents in the system. Hence, the hierarchy is actually a composite agent which consists of several nested environments often called agencies. Agencies consist of a co-ordinator and one or more subordinate agents or

agencies. In this example, agent A acts as the co-ordinator of the overall environment. It has two agencies as its subordinates: one agency co-ordinated by agent B which has two individual subordinates, agents D and E; and another agency co-ordinated by agent C which has as its subordinates an individual agent, agent F, and an agency co-ordinated by agent G consisting of two individual agents, H and I.

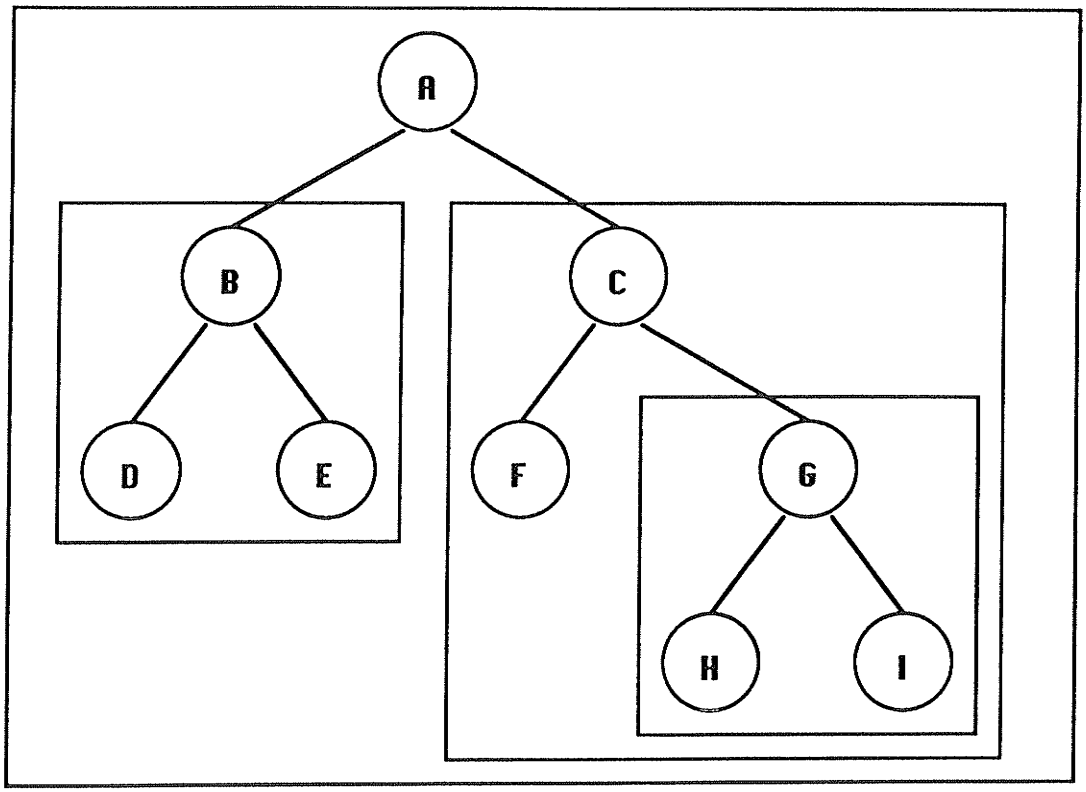


Figure 3.1: Hierarchy of Agents and Agencies

A possible protocol that can be used with this topology is to restrict interaction to the explicit links among the agents created by the hierarchical topology. Hence, communications must traverse the

appropriate paths in the hierarchy, possibly being forwarded by intermediate agents responsible for co-ordinating various portions of the environment. For example, agent C must pass all communications destined for agent B through the co-ordinating agent, agent A. Furthermore, any subordinates of agent C that wish to communicate with agent B, or any of the agents in the agencies that agent B co-ordinates, must route communications through the paths created by the topology. This type of protocol enables the system developer to create an explicit set of communication paths and distribute the task of co-ordinating the paths to specific agents within the environment. This does not, however, impose any restrictions on the contents or frequency of interactions that may be performed.

Employing a structured protocol increases the explicitness surrounding the ways in which interactions may be performed. This serves to ease the task of maintaining and extending the distributed environment because agents cannot communicate in an ad hoc manner which would increase the complexity involved in understanding the interactions that occur among the agents. Furthermore, agents which share common tasks and capabilities can be grouped into agencies, and greater control can be exercised over how other agents can interact with the members of the agency.

3.6.6 Flexibility

The flexibility inherent in a model of distributed problem solving is another important factor to consider. Inflexible models constrain the type of information that may be communicated and the ways in which

interactions may be performed. As a result, the scope of interactions that may be performed is limited which often drastically reduces the synergy of the agents' abilities that can be achieved. Therefore, it is imperative that a model provide a flexible means of facilitating the various forms of interaction that are required within each subset of a distributed environment.

Providing the availability of both implicit and explicit communication accounts for one aspect of flexibility which may be embodied in a model of distributed problem solving. Thus, although explicit communication is more desirable in the long run, there may be instances in which implicit communication is beneficial. This form of flexibility enables the agents to interact using the communication paradigm that is most appropriate for the interactions that must be carried out.

Structured protocols like the one examined in the previous section are designed to provide increased explicitness, but they also provide added flexibility. Once again consider the example in Figure 3.1. Using a structured hierarchical protocol, a communication from agent D to agent F must be routed through the topology appropriately (D to B to A to C to F) because a direct link between the agents does not exist. Although this imposes restrictions on the ways in which communications may be routed, it does increase the robustness of the system. For instance, the agents acting as co-ordinators of each subset of the environment that the communication must traverse can examine the communication and determine an alternate route or destination when agent F is disabled. Agent D on the other hand probably would not contain

such information. This information would most probably be contained in agent C which co-ordinates the agency in which agent F resides. Consequently, if agent D were to directly communicate with agent F and it found agent F to be disabled, then agent D would be required to wait for agent F to be repaired or abandon the processing that requires assistance from agent F. Both alternatives are undesirable. Therefore, the structured protocol provides the basis for increasing the robustness of the system by providing methods for creating alternate routes for interactions which incur difficulties that would otherwise prevent completing them successfully.

Some models of distributed problem solving also provide flexibility by enabling interactions among agents to occur at more abstract levels in which an agent initiates an abstract communication that is subsequently refined as it moves through the environment. For example, agent D could initiate a communication to agent B requesting some task be performed without specifying which agent should perform the task. Agent B would apply its conceptual model to determine how it can satisfy the request; it may perform the task itself, or it may distribute the request to another agent whom it knows is capable of performing the specified task (or distributing it appropriately). This process would continue until the request was distributed to an agent which performs the task and communicates the results back along the communication path until the source of the request receives it. This planning of interactions provides a very flexible method of incorporating additional agents or extensions to existing agents because interactions occur indirectly. Hence, agents responsible for co-ordinating portions of the environment assume responsibility for knowing the capabilities of their

subordinates and can use this knowledge to route communications through the topology appropriately, possibly transforming an initial communication into several appropriate communications and managing the integration of the results which are produced.

It is also often advantageous to allow agents from different agencies to communicate directly. Such interactions are useful when agents from different agencies interact frequently because the time and effort required to route the communications through the topology would be reduced significantly. A truly flexible model enables direct interaction among agents when it is beneficial, but also provides the ability to route the interactions indirectly to alternate destinations when the desired destination agents are disabled. Hence, cost-effectiveness is achieved and robustness is maintained.

3.7 SURVEY OF DISTRIBUTED PROBLEM-SOLVING PARADIGMS

3.7.1 Overview

Distributed problem-solving research has developed several paradigms designed to facilitate the creation of co-operative problem-solving environments. Four such paradigms are presented in this section. Although the paradigms share a common goal, the methods employed in each vary significantly. The effectiveness and applicability of each paradigm is dictated by the amount and nature of distributed problem-solving techniques that each embodies. I examine the general principles of each paradigm and discuss the types of co-operative problem solving that each permits. This is by no means a complete survey of all available distributed problem-solving paradigms, nor is it an exhaustive examination of the paradigms presented.

3.7.2 Transactional Blackboards

The blackboard paradigm is a very powerful problem-solving model that is widely use in AI systems [Nii 86]. In the blackboard model, knowledge is organized into one or more knowledge hierarchies with each level in the hierarchy representing increasingly detailed knowledge about a specific subset of the underlying domain. The result is a set of knowledge sources that are capable of contributing information to the overall problem solution. Each knowledge source represents a particular subset of the domain knowledge. The knowledge sources are organized around a global blackboard that stores the current state of the problem-solving process. The knowledge sources are self-activating components that monitor the blackboard contents and react appropriately when changes to the blackboard occur that require their attention. Communication and interaction among the knowledge sources take place through the blackboard, imposing an organized protocol on the overall model.

The performance and flexibility of a blackboard system may be enhanced significantly by enabling the knowledge sources to operate concurrently. The transactional blackboard model is designed to provide transaction-based facilities supporting the parallel execution of knowledge sources in a blackboard system [Ensor 85]. The model extends the basic blackboard system by providing the blackboard component with a transaction manager. All accesses to the blackboard must be part of a transaction, and the transaction manager is responsible for scheduling and synchronizing transactions requesting access to the blackboard contents. The transaction manager controls access to the blackboard by

using a data locking mechanism consisting of two basic locks: a read lock which enables concurrent reading of data but which prevents any writes, and a write lock which provides a knowledge source with exclusive access to a blackboard data structure enabling the locked data structure to be modified. The transaction manager schedules the execution of transactions and determines data locks based on pending transactions and transactions currently in progress. Consequently, the transaction manager preserves data consistency by preserving serializeability [Eswaran 76].

The transaction blackboard model provides a distributed environment in which agents (knowledge sources) communicate by modifying or reading the contents of a shared data structure (blackboard) in separate transactions. The transaction-based mechanism provides an explicit means of initiating communication, but the actual communications are implicit because they rely on agents knowing when and how to modify the blackboard to "send" a communication and knowing when and how to read the blackboard contents to recognize that a communication is present. Therefore, the explicitness is misleading, it is required to ensure data consistency, not to achieve greater explicitness in modeling interactions among the agents.

Another aspect to be considered is that interactions are co-ordinated by a single agent, the blackboard. Consequently, the flexibility of the model is limited and the co-ordination of co-operative problem solving becomes inefficient as the number of agents and amount of interaction increases. The decentralized decision-making capabilities provided by the model are limited by the bureaucracy imposed by the central

controller. The effects of this bureaucracy increase dramatically as the number of agents (knowledge sources) increases. The model is designed for creating environments consisting of coarse-grained agents which are capable of highly autonomous actions and which need to interact infrequently [Corkill 79]. Moreover, interactions among agents focus primarily on data representing intermediate results or hypotheses. Although the model's synergistic abilities are limited, it does provide a structured mechanism for creating distributed problem-solving systems in which semi-autonomous agents may operate concurrently, yet are able to interact and work co-operatively to solve problems.

3.7.3 Contract Nets

The contract net framework is a model of distributed problem solving which emphasizes task-sharing as a means of facilitating co-operative problem solving [Smith 78]. The model distributes the computational load across several agents operating concurrently and provides a task distribution mechanism which enables the agents to contract the services of other agents. As a result, the agents are able to interact and distribute tasks to other idle agents when necessary or advantageous. Task distribution is viewed as a mutual selection process, a discussion carried on between an agent with a task to be executed and a group of agents that are capable of executing the task [Smith 79]. The collection of agents is referred to as a contract net and task execution is viewed as a contract between two or more agents.

The model facilitates interaction among agents using contract negotiation in which an agent requiring the execution of a task

```
To: #              <managers make announcements of this form>
From: 25
Type: TASK ANNOUNCEMENT
Contract: 22-3-1
Message:
  Task Abstraction:
    TASK TYPE 'signal
    NODE NAME '25 POSITION 'p
  Eligibility Specification:
    MUST-HAVE DEVICE TYPE 'sensor
    MUST-HAVE NODE NAME 'SELF POSITION area 'A
  Bid Specification:
    NODE NAME 'SELF POSITION
    EVERY DEVICE TYPE 'sensor TYPE NUMBER


To: 25             <sensors nodes respond to manager>
From: 42
Type: BID
Contract: 22-3-1
Message:
  Node abstraction:
    NODE NAME '42 POSITION 'q
    sensor TYPE 'S NUMBER '3
    sensor TYPE 'T NUMBER '1


To: 42                      <award message is transmitted>
From: 25
Type: AWARD
Contract: 22-3-1
Message:
  Task Specification:
    sensor NAME 'S1
    sensor NAME 'S2
```

Figure 3.2: Contract Net Announcement, Bid, and Award Messages [Smith 78]

generates a task-announcement message that is broadcast on the contract net (Figure 3.2). An agent initiating a task-announcement is called a task manager. A task manager is responsible for co-ordinating the negotiation of a contract to execute a specified task and processing the results of its execution. Agents in the network listen for task announcements and may submit bids to the corresponding task manager when

an announcement that interests them is detected. An agent submits a bid to try to become a contractor for the announced task. A task manager may receive several bids and must select a contractor or a set of contractors based on the information in the various bids [Shapiro 87, p. 248]. The selection process represents the awarding of a contract.

The contract net model provides a structured task distribution mechanism which enables decentralized interactions among agents via contract announcements, bids, and awards. Agents take on the role of task manager or contractor dynamically according to their local needs and the task announcements propagating throughout the network. This provides greater flexibility in the organization of agents and in the ways they may interact. Furthermore, the contracting process provides a powerful vehicle for enabling agents to negotiate; task managers and bidders can exchange information in order to determine mutually acceptable contracts. The model, however, is still somewhat restrictive in that more advanced structured protocols such as multi-level protocols have not yet been addressed and interactions are restricted to broadcasting task announcements - mechanisms for direct communication are not part of the model (particularly direct communication based on knowledge of the other agents). The model is geared towards loosely coupled, coarse-grained applications which limit the amount of communication that must occur and the number of agents that are involved in communications. This ensures that the potential bottlenecks that may occur from broadcasting all communications are kept under control. Agents are assumed to be highly autonomous and the model

provides the agents with the ability to interact using an explicit mechanism in order to facilitate co-operative problem solving.

3.7.4 Actors

The actor model of concurrent programming was developed by Hewitt to facilitate the development and analysis of open systems [Hewitt 84]. An actor system is comprised of a collection of actors representing active agents which play roles on cue according to finite length scripts [Agha 86]. Each actor's script contains a behaviour definition that defines the actions that the actor may perform. These actions are based on the communications received from other actors in the system. Thus, actors may be viewed as computational agents which map each incoming communication to a 3-tuple consisting of: a finite set of communications sent to other actors; a new behavior (which will govern the next computational behaviour); and a finite set of new actors that are created [Agha 86]. Computation in an actor system is driven by communications among the various actors using a message passing protocol. Sending messages among actors results in computation which causes the actor system to evolve to include new computations and new actors. A computation may involve communication among two actors or communication among a set of actors. The behaviours of the participating actors interact to create a synergistic effect which achieves the necessary global computation.

The actor formalism provides a decentralized model of distributed processing in which the programmer is able to concentrate on the concurrency required in an application rather than on how and when to

force concurrency [Agha 86]. Each actor contains its own local model of processing which specifies the actions it must perform for each communication it may receive. An actor also contains a partial model of the global system stipulating the computation it may initiate and the new actors that must be created when specific communications are received. Consequently, control does not have to be defined a priori; instead control mechanisms and strategies are created dynamically through communications among the actors.

Negotiation plays a vital role in determining the actions and control characteristics that the actor system will exhibit [Hewitt 84]. Furthermore, the topology of the community of the actors within the actor system is not restricted to any predefined scheme. In fact, the topology of the actors may be dynamically determined based on the communications propagating among the actors [Agha 86].

The actor model represents a powerful formalism for creating distributed environments of numerous fine-grained agents (actors). Each actor is capable of limited processing and of initiating limited global effects (new actors and additional communications). Nevertheless, using an explicit message passing mechanism, the computational resources embodied in the actors may be combined to produce effective high-level processing. This fine-grained, tightly coupled environment results in extensive amounts of communication during program execution. As a result, architectural considerations such as load balancing, locality of reference, process migration, et cetera must be examined in order to ensure an efficient implementation [Agha 86]. Furthermore, resource allocation must be carefully controlled as actors must have dynamic

amounts of computational resources in order to create new actors and initiate further communications.

3.7.5 Functionally Accurate, Co-operative Distributed Systems

The distributed vehicle monitoring system developed by Corkill and Lesser embodies a generic model of distributed problem solving based on functionally accurate, co-operative computational agents [Lesser 81]. The model facilitates the construction of distributed problem-solving networks for applications in which there is a natural spatial distribution of information and processing requirements, but insufficient information for each agent to make completely accurate control and processing decisions without extensive inter-agent communications [Lesser 83]. Agents are organized as members of a network in which each agent is an independent problem solver with specific responsibilities within the overall environment. The agents work concurrently to produce intermediate partial solutions and co-operate through communication to revise and extend their partial solutions in order to converge towards a complete solution.

The agents are considered functionally accurate because they are capable of generating accurate solutions without requiring all intermediate solutions to be consistent and complete. Hence, agents are able to work on specific subsets of the overall problem from different local perspectives and co-operate by exchanging high-level tentative results. Each agent contains knowledge describing its general functions and interests and those of other agents that it may interact with [Corkill 83]. This knowledge is used to co-ordinate effective

interactions among the agents. The agents are able to plan activities and exchange such plans with other agents to determine how best to co-operate. This enables the agents to interact while maintaining global coherence [Durfee 85].

The model provides a formalism for creating distributed environments of semi-autonomous, coarse-grained agents and for co-ordinating communications among the agents to enable result-sharing without flooding the communication bandwidth with partial solutions or withholding partial results that may hinder the evolution of global solutions [Lesser 83]. Embodying meta-level knowledge in each agent ensures that communications are well planned and that the computational resources of each agent and the system as a whole are used effectively. Each agent contributes partial solutions but also contributes to the integration of partial solutions into a final result.

Although the model provides a powerful testbed for examining distributed problem-solving semantics, it is geared primarily towards coarsed-grained applications. In such applications it is assumed that the agents embody significant processing resources and are able to plan inter-agent communications effectively. Consequently, the majority of inter-agent communications represent summary information rather than raw data. The agents must be capable of interpreting high-level communications and, more importantly, they must be able to determine at what level interactions should take place. Therefore, the onus of co-ordinating interactions is placed squarely on each agent which increases the understandability, maintainability, and robustness of the system [Shapiro 87, p. 247]. However, this approach is often infeasible

in applications which require fine-grained agents that for the most part must communicate with significantly less awareness of the communications being performed, let alone the level of information contained in such communications.

3.8 SUMMARY

This chapter began with a discussion of the drawbacks inherent in the closed-world methodology employed by many current AI methodologies which invariably renders these techniques inadequate for constructing truly intelligent systems. As an alternative, distributed problem-solving systems were examined in which several closed-world systems are integrated and provided with the means to co-operate with one another to significantly increase their computational abilities and those of the environment in which they reside. After presenting the requirements of an ideal distributed system (open systems), I examined several characteristics and factors that influence interaction among agents in distributed problem-solving environments. This discussion underscored the importance of a powerful and flexible mechanism to co-ordinate interactions effectively. Several distributed problem-solving paradigms were presented which facilitate the construction of distributed problem-solving environments, and, more importantly, provide a powerful testbed for examining the semantics of problem solving in such environments. Note: there are several formal approaches to representing distributed problem-solving systems using model logics, but these approaches cannot be implemented because of the lack of appropriate tools. Formal approaches do, however, provide invaluable research in the modeling of distributed problem solving

systems. Examples of formal approaches to DAI can be found in [Rosenschein 85], [Ginsberg 87].

Underlying this discussion is the fundamental principle of distributing problem-solving: many complex problems can only be solved through the co-operative efforts of several, diversely qualified agents. Co-operation is the key to the success of such systems, it enables multiple agents with differing goals and abilities to work together to solve complex problems effectively.

Chapter IV

A MODEL OF DISTRIBUTED PROBLEM SOLVING

4.1 OVERVIEW

In this chapter I present a conceptual model of distributed problem solving. The chapter begins with a summary of the fundamental requirements such a model must embody in order to facilitate the construction of co-operative problem-solving environments. I emphasize the importance of an explicit distributed problem-solving mechanism, and I describe a model that facilitates the construction of intelligent agents capable of planning and co-ordinating co-operative problem-solving activities. The goal in this chapter is to define a conceptual model that provides an agent with explicit mechanisms for co-ordinating both autonomous and co-operative problem solving in a distributed environment. The reader can refer to Appendix A for a detailed examination of the engineering issues surrounding the implementation of the model.

4.2 REQUIREMENTS OF DISTRIBUTED PROBLEM SOLVING

At the core of distributed AI research is the belief that intelligence is formed from creating effective connections among individual agents and allowing the agents to communicate using these connections. This connectionist theme is based largely on the neural mechanisms embodied in the human brain [Hillis 85]. Minsky summarizes the role of connectionism in the brain as follows:

> "Thinking in the human sense is facilitated by the well-connected meaning structures (neurons and nets of neurons) that let you turn ideas around in your mind, to consider alternatives, and envision things from many perspectives until you find one that works" [Minsky 86].

What Minsky refers to as meaning-structures may be viewed as individual agents or collections of agents (agencies). His emphasis on well-connected agents represents the importance of connections among the neural agents and the interactions that occur across these connections. He also emphasizes the importance of perspectives and the manipulation of various perspectives. As a result, the neural agents and agencies must co-operate to achieve the levels of intelligence achieved by the brain as a whole.

In the remainder of this section, I examine the three basic requirements that a model of distributed problem solving must embody in order to facilitate the creation of co-operative problem-solving systems.

4.2.1 Requirement 1: Ability to Construct Individual Agents

A distributed problem-solving system consists of individual agents which embody the problem-solving capabilities of the environment. The tasks assigned to each agent and the level of intelligence embodied in each may vary substantially, but each agent is capable of making some contribution to the overall environment. For example, consider the variety of employees that constitute a company. The intellectual responsibilities of the president of the company and of a janitor may vary tremendously, but the company cannot function effectively without each performing his assigned tasks. If the janitor stopped performing

his task of cleaning up the environment, then at some point the environment would become overrun with garbage which would hinder the effectiveness of the other employees. The role of the janitor could be distributed to the other employees, but this would lead to a reduction in their effectiveness because they would have to carry out their own tasks and those of the janitor. Therefore, although the roles of the employees may vary significantly, each makes a contribution to the overall environment without which the company could not function effectively and efficiently.

The knowledge-based programming techniques discussed in Chapter 2 provide the means for creating individual computerized agents. There is, however, nothing preventing the construction of all or some of the agents using conventional programming techniques. Thus, it is possible to create a distributed problem-solving environment using a combination of knowledge-based agents and procedure-based agents which may embody diverse abilities. This provides a tremendous amount of flexibility which enables the processing embodied in each agent to be dictated by the abilities that it must possess. As a result, a developer can focus on programming the required problem-solving abilities into a set of individual agents and then concentrate on integrating the agents into a distributed problem-solving environment.

4.2.2 Requirement 2: Arbitrary Topology

The next step is to connect individual agents to create a distributed environment; the connections among the agents will ultimately be used to enable the agents to interact. The topology of the environment must be

based on the attributes of each agent and the relationships inherent among the agents. Often the topology should incorporate infrastructures that represent collections of computationally related agents which function as agencies. These agencies represent composite agents that embody structured relationships among the processing resources and conceptual models of a set of individual agents (and possibly other agencies).

It is imperative that the topology of the environment of agents and agencies be adaptable. This enables the developer to tailor the organization of the agents to the specific characteristics of an application in order to facilitate effective interactions among the agents. In fact, various subsets of a distributed environment may be constructed using distinctly different topologies of agents and agencies. This flexibility enables the environment to be organized arbitrarily based on the agents it embodies and the connections required among the agents. Flexibility of this sort facilitates a natural integration of the agents and is essential in achieving the desired synergistic effect.

In many cases the most appropriate organization of the agents must be determined through empirical studies of the environment. As a result, the organization of the agents and agencies must be easily modified when adjustments are required. Moreover, because a distributed problem-solving system must remain open-ended, it is important that the environment be easy to extend. It must be possible to incorporate additional agents or agencies into the environment in a smooth and effective manner without unnecessary burden to any single portion of the

environment. The burden of modifying and extending the environment must be distributed to the various agents and agencies. Distributing responsibilities in this manner enables the agents to co-ordinate the modification and extension of the environment co-operatively.

4.2.3 Requirement 3: An Explicit, Knowledge-Based Planning Mechanism

The most important component of a distributed problem-solving system is a planning mechanism that can plan and co-ordinate co-operative problem-solving activities among the agents in the system. Individual agents represent the processing resources of the system and the topology of the environment defines the connections among the agents, but a planning mechanism completes the integration of the agents by enabling them to utilize inter-agent connections to interact with one another and perform co-operative problem solving. Interactions are carried out using communications among the agents, but it is the ability to plan and co-ordinate the interactions that facilitates effective distributed problem solving.

Although several communication paradigms have been developed to facilitate interactions among agents in distributed environments, none of these paradigms provides sufficient facilities for planning and managing distributed problem-solving activities effectively. What we require is a model that enables each agent to embody an explicit planning component which co-ordinates the agent's distributed problem-solving activities by manipulating meta-level knowledge. This meta-level knowledge can be divided into two categories: plan-centred knowledge describing the tasks that should be performed to solve

problems; and resource-centred knowledge describing the problem-solving abilities embodied in the agent and in some of the other agents in the environment that can be used to perform the specified tasks. Thus, the planning component of an agent determines the problem-solving tasks that must be executed to solve given problems and the agents that can be used to perform these tasks (an agent can solve a particular problem by itself or in co-operation with one or more other agents).

4.3 A MODEL OF DISTRIBUTED PROBLEM SOLVING

4.3.1 Overview of the Model

In this section I present a model of distributed problem solving. I assume that facilities exist to construct individual problem-solving agents and to organize the agents into a distributed environment of arbitrary topology. The model contains an explicit planning mechanism that enables the agents to perform co-operative problem solving effectively. As a result, the model represents the core of the distributed problem-solving methodology that I have developed to facilitate the construction of co-operative problem-solving environments from collections of individual agents.

Figure 4.1 illustrates the basic organization of the model in which each agent consists of a problem-solving component and a planning component. The problem-solving component is a knowledge system constructed using either knowledge-based or procedure-based techniques (as described in Chapter 2). It embodies problem-solving abilities that enable it to perform a set of tasks. The planning component is a knowledge-based system that plans and co-ordinates the problem-solving

functions that its problem-solving component should perform and the functions that should be distributed to other agents in the environment. Some agents may not participate in the planning process and so will contain only a problem-solving component. These agents will have their co-operative problem-solving functions co-ordinated for them by a shared planning component.

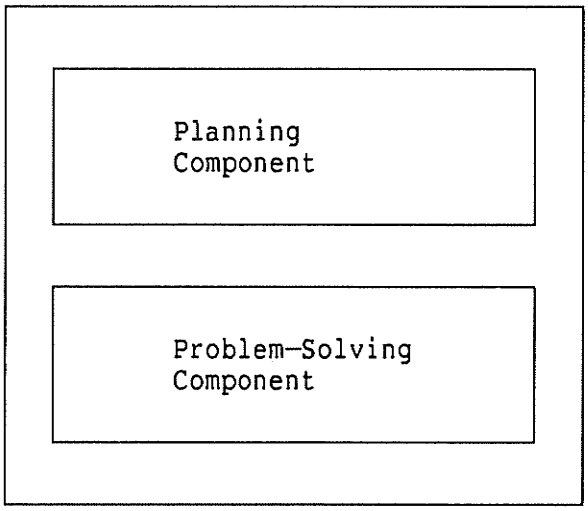


Figure 4.1: A Basic Agent in a Distributed Problem-Solving System

Planning components are programmed with knowledge that describes meta-level problem-solving plans and the agents (resources) in the environment that can perform the tasks specified by these plans. This knowledge describes the problems an agent knows how to solve, either using its problem-solving component, other agents, or some combination of the two. Essentially, a planning component performs meta-problem solving to plan and co-ordinate the execution of basic problem-solving tasks by distributing the tasks to the appropriate agents (local and/or

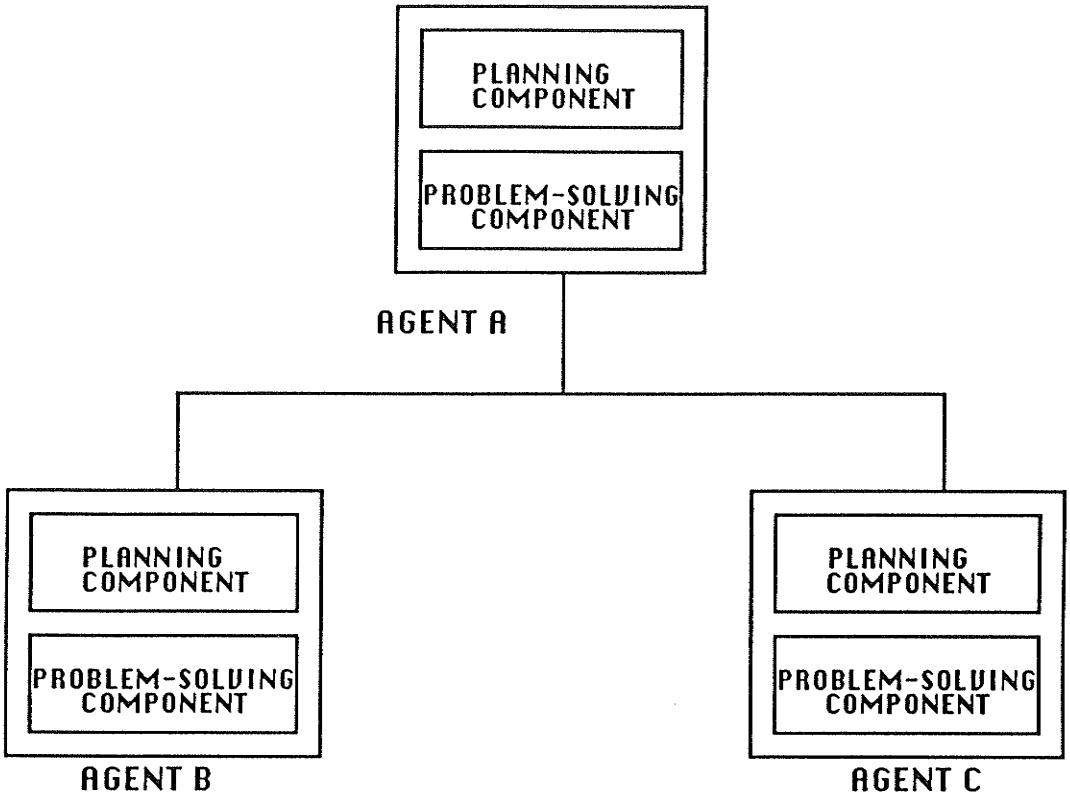


Figure 4.2: An Agency of Agents

external). The distributed environment is a collection of planning and problem-solving component pairs that function in parallel (Figure 4.2).

The agents within the environment communicate through their planning components using a message passing protocol; all interactions among agents are performed through communications between their respective planning components. A planning component may act on messages received either from its problem-solving component, or from external agents (through their planning components). When a planning component receives

a message containing a problem-solving request, it uses its knowledge base to plan and co-ordinate interactions with appropriate agents to initiate the necessary problem-solving activities. All responsibility for managing interactions and co-ordinating distributed problem solving is removed from the problem-solving components and distributed to the appropriate planning components in the environment. This distribution of responsibilities represents the fundamental principle of the model: in addition to a basic problem-solving component, each agent must have an explicit component responsible for all distributed problem-solving functions. An agent is therefore able to reason about several aspects of its operation: how it can perform basic problem-solving functions (problem-solving component); and when it should perform problem solving for other agents as well as when it should have other agents perform problem solving for it (planning component). The planning components enable the agents to reason about their own problem-solving abilities and those of some of the other agents in order to perform co-operative problem solving effectively.

4.3.2 Physical-Level Assumptions

Several assumptions are made regarding the implementation characteristics of the distributed environments in which the model may be employed. This enables the model to focus on more abstract features required to perform distributed problem solving, rather than being bogged down dealing with implementation-dependent communication details. Individual agents may all reside within the same physical system where they operate as concurrent processes in a time-sharing environment, or the agents may be distributed among two or more distinct physical

systems where one or more agents are assigned to a processor in a multi-processor environment (the processors may be located at the same location or at different locations). In the case of a multi-tasking environment the agents are connected by logical links within the environment. In a multi-processor setting, the agents are interconnected using physical links (wires) when the connections span processors and using logical links when the connections are between agents in a shared processor. Either scenario is feasible and it is assumed that the details required to create suitable connections can be implemented accordingly. The specific characteristics of a particular application will dictate the degree of coupling that should be implemented; some applications will be better suited to loosely coupled environments (to avoid potential thrashing), while others will require a tightly coupled environment.

Finally, the model assumes that a basic message passing mechanism exists and that a suitable communications link is available to enable each agent to send and receive messages (Figure 4.3). The actual construction of the messages to be sent and the deciphering of messages received are the responsibilities of the planning components and, to a lesser degree, the problem-solving components (a problem-solving component must be able to send and receive basic messages between itself and its planning component). The responsibilities of the communications link include: sending and receiving messages containing data, operation, or control information via a message passing mechanism; managing a queue where messages are placed while waiting to be processed by an agent; handling priorities assigned to messages and interrupts

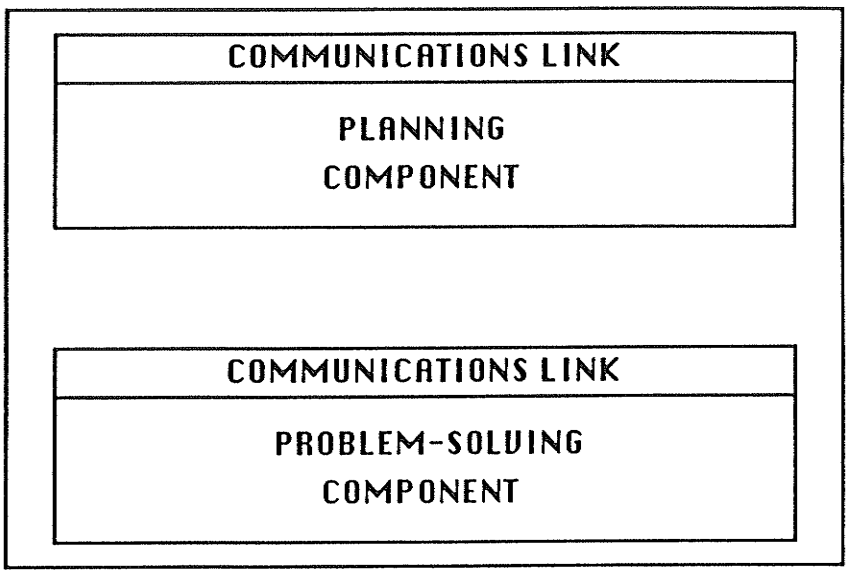


Figure 4.3: Communications Link in the Components

that may be sent in the form of messages which require special attention; and co-ordinating the transfer of messages among agents to ensure delivery (handshaking tasks).

4.4 PLANNING COMPONENTS

The central element in the model of distributed problem solving outlined in this chapter is the planning component which enables an agent to manage its problem-solving activities within a distributed environment. Figure 4.4 illustrates the structure of a planning component. A planning component consists of several parts including: an implementation-dependent communications link that handles the sending and receiving of basic messages; an activity blackboard that represents a working memory structure containing the processing activities that are

in progress or are scheduled to be processed; a meta-level knowledge base describing problem-solving plans and the agents that can carry out the tasks specified by these plans; and a control strategy that monitors the communications link for problem-solving requests and then plans and co-ordinates appropriate problem-solving activities for each request received.

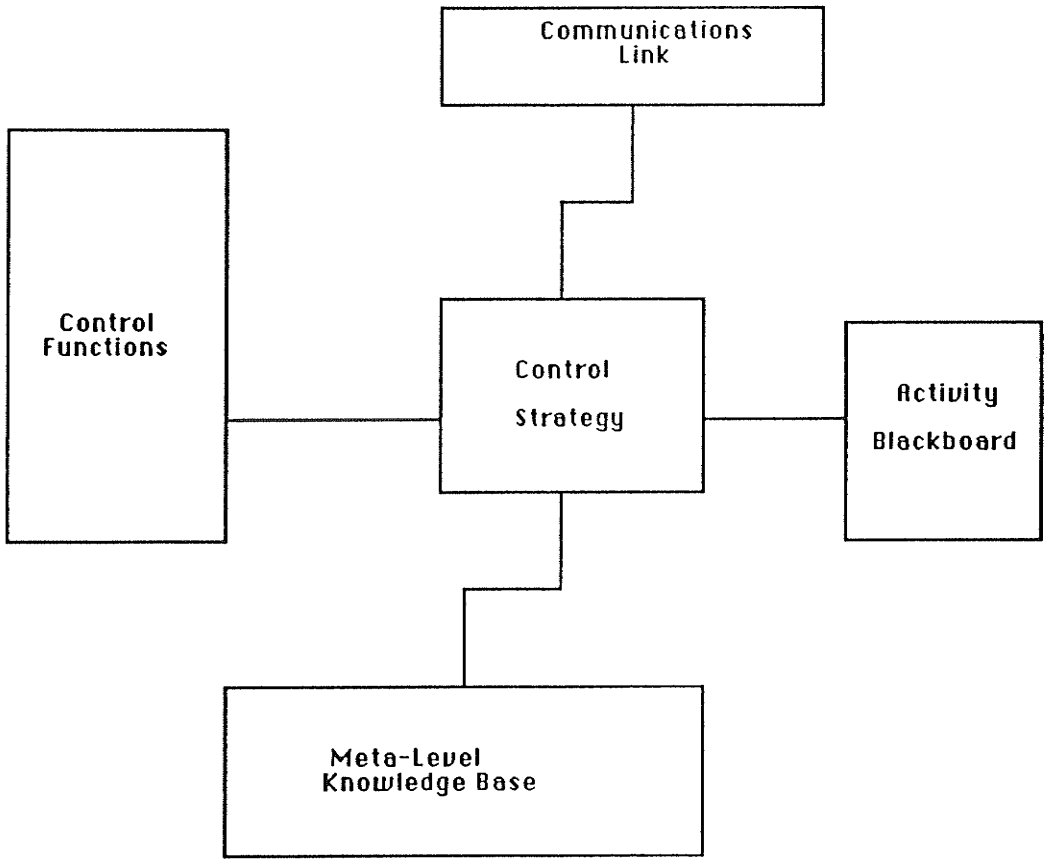


Figure 4.4: Structure of a Planning Component

An important characteristic of the model is its adaptable nature. Each planning component's knowledge base will initially describe the agent's own problem-solving abilities (embodied in its corresponding problem-solving component(s)). In addition, a developer can add knowledge describing the external problem-solving abilities of which an agent should be aware. As problems are solved co-operatively, the meta-level knowledge base will expand to include more extensive knowledge of the capabilities of other agents. Thus, although each planning component shares the same organizational structure and processing mechanisms, it is the knowledge contained in a planning component's knowledge base that ultimately dictates the meta-problem solving that it can perform.

The knowledge contained in each planning component will vary, reflecting each agent's model of the problem-solving environment. These models will be incomplete and often will be inconsistent or in conflict with one another. This is due to discrepancies in the beliefs and capabilities that the agents possess and the fact that an agent can never have a complete model of the problem-solving environment because of the size, complexity, and open-endedness of the environment [Smith 79]. Therefore, each agent utilizes its own partial model of the environment to reason about the problems it can solve using its problem-solving component, other agents, or a combination of the two. When conflicts arise, the agents must negotiate with one another in order to formulate problem-solving activities that are acceptable to each agent involved.

In the remainder of this chapter, I discuss the organizational principles of the model, including the representation and control facilities provided to plan and co-ordinate co-operative problem-solving activities.

4.5 META-LEVEL KNOWLEDGE BASE

4.5.1 Overview

A planning component's knowledge base represents an agent's model of its own problem-solving abilities and those of some of the other agents in the environment. This meta-level knowledge can be divided into three categories: plan knowledge, task knowledge, and agent knowledge, each of which is represented as a hierarchy of frame structures called scripts (Figure 4.5). Plan knowledge represents meta-plans that specify abstract problem-solving steps (sets of tasks), while task and agent knowledge, collectively referred to as resource knowledge, describe the problem-solving abilities of the agent's problem-solving component and some of the other agents in the environment. As a result, each agent can apply an explicit knowledge-based description of the problem-solving abilities available in the environment in order to co-ordinate co-operative problem solving effectively.

An agent can initiate distributed problem solving by sending another agent a communication containing a description of the problem solving that is to be performed. The planning component of the recipient of the problem-solving request must search its meta-level knowledge base for plan and resource knowledge which specifies the tasks to be performed to fulfill the request and the agents that can carry out these tasks. It

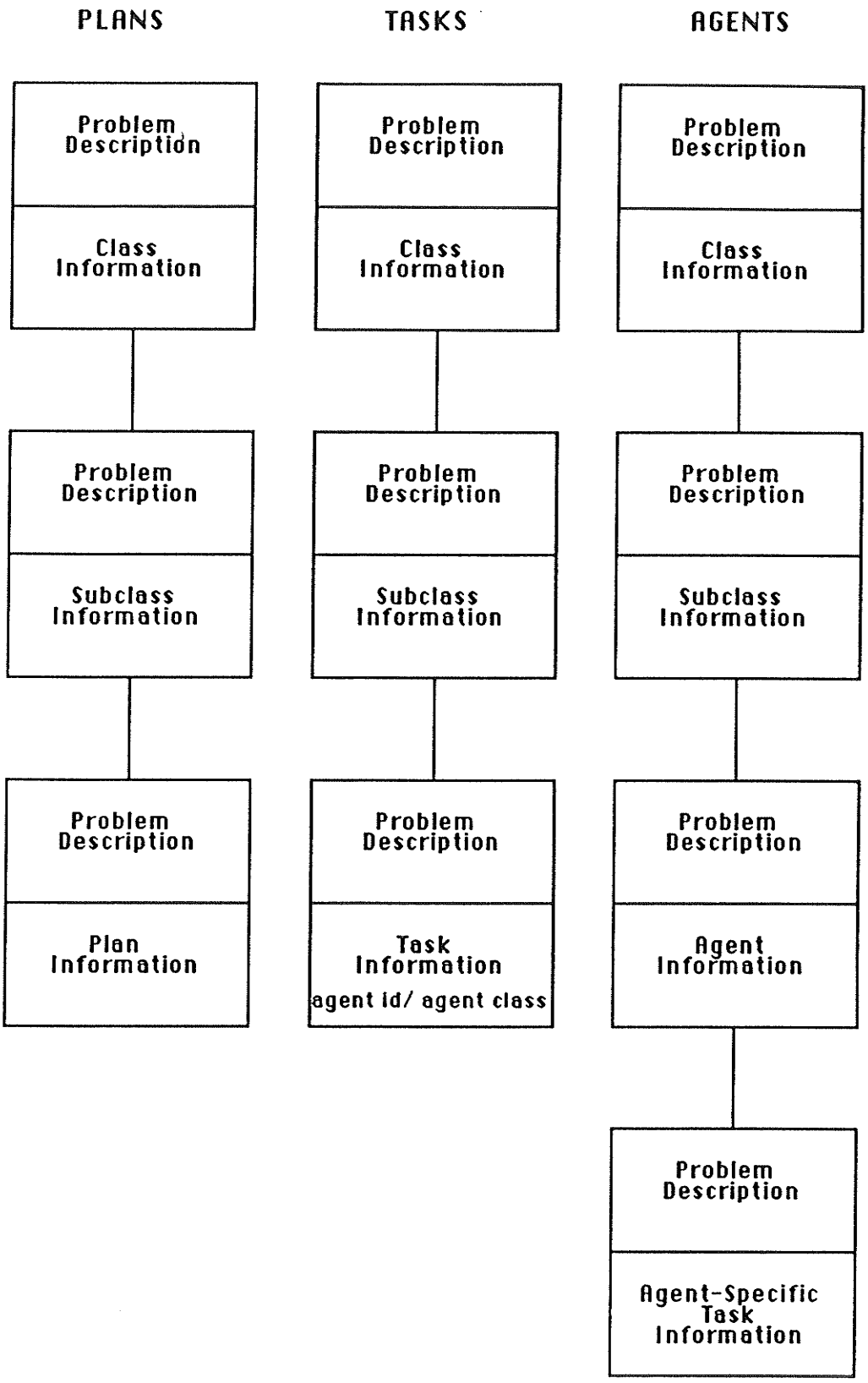


Figure 4.5: Types of Knowledge in a Planning Component

then uses this knowledge to plan and co-ordinate co-operative problem-solving activities that will carry out the problem-solving request.

4.5.2 Problem Descriptions

Problem Description =

[DESCRIPTION CONSISTS OF SEVERAL SLOTS]

SLOT:

FACETS=

Values= ***value specifications***

Default= ***value specifications***

Constraints= ***criteria*** **[Unless** ***criteria*** **]**

If-needed= ***demon specifications***

If-added= ***demon specifications***

Inheritance= **APPEND / OVERRIDE**

[SOLVED-WHEN: ***production rules*** **control info]**

[APPLICABLE-WHEN: ***production rules*** **control info]**

Figure 4.6: Frame Structure for Representing Problem Descriptions

A problem description is a frame structure which describes problem-solving requests or problems and tasks to which plan and resource scripts are applicable (see Figure 4.6). A problem description consists of a set of slots which describe operations to be performed, objects to be manipulated, and any other problem attributes to be considered. Each slot consists of a set of facets which describe various characteristics of the problem attribute represented by the

slot. These facets include: required or specified values, default values, constraints, and IF-NEEDED and IF-ADDED slot demons. Values specified in a slot represent facts about problems or tasks (a single value, a list of values, or arbitrarily complex expressions). Default values provide value specifications that are to be used when the value facet of the slot is empty. This enables a problem description to represent default facts about the problems or tasks which it describes. Value and default value specifications are used to describe characteristics of problems, both of problems to be solved and of problems and tasks to which plan and resource knowledge scripts are applicable.

Constraints specify criteria which the values assigned to a slot must satisfy (situations in which constraints can be relaxed are represented using the UNLESS option). A set of legal values and logical or relational expressions (e.g. predicates) are examples of possible constraints that can be imposed on a slot value. Constraints are specified in problem-solving requests to restrict the values that problem attributes can be assigned during problem solving. In order to fulfill a problem-solving request, acceptable values must be determined for each problem attribute. In addition, constraints can be specified in the problem description components of plan and resource scripts to constrain the types of problems and tasks to which the scripts are applicable (e.g. operations that can be performed, objects that can be manipulated, etc.). Plans and resources are only applicable if the constraints associated with each problem attribute are satisfied. Values for each problem attribute are obtained from corresponding attributes in

problem-solving requests, by interrogating other agents, or by invoking associated IF-NEEDED or IF-ADDED demons.

IF-NEEDED demons are procedures (expressions, rules, functions, etc.) that can be invoked to determine a value when a slot's value and default facets are empty. IF-ADDED demons, on the other hand, are invoked when values are added to a slot. These demons can calculate and assign values to other slots based on the specified values (or perform other appropriate actions). This enables a problem description to function as an event-driven data structure [Bobrow 75]. As a result, when a problem description is only partially instantiated, IF-ADDED demons in the slots for which values were specified can be invoked to infer or generate values for other problem attributes (generated values usually must be verified by the agents involved).

A problem description structure also contains a slot which specifies control knowledge (rules and procedures) that can be used to determine how to instantiate the structure; this includes information for determining when the structure is considered instantiated and methods that can be used to reformulate initially incompatible instantiations through co-operative negotiations with other agents. The SOLVED-WHEN slot is used for this purpose in problem-solving requests and the APPLICABLE-WHEN slot specifies control information in the problem description component of knowledge scripts. These slots may specify heuristic rules which represent criteria that suggest with some certainty whether a problem-solving request is solved or a knowledge script is applicable. These heuristic rules specify partial instantiations of problem descriptions which suggest that a problem is

potentially solved or a knowledge script is potentially applicable. Rules may also specify criteria that can be used to determine when a knowledge script is not applicable, as well as ways to negotiate reformulations of the problem attributes that will result in a knowledge script becoming applicable or a problem-solving request becoming fulfilled. Additional processing is normally required to verify that the problem has in fact been solved or that the knowledge script is in fact applicable. The rules specified in these slots can be chained together so that the inferences made by some rules can be used to enable other rules to be applicable.

The control facet of these slots specifies the control strategy to use in applying the control knowledge appropriately. This enables arbitrarily complex criteria and control information to be specified in problem-solving requests and knowledge scripts. As a result, each knowledge script specifies information describing co-operative problem-solving functions and control information for co-ordinating the application of those functions. Simlarily, each problem-solving request can specify information describing the problem solving to be performed and control information for co-ordinating the execution of that problem solving. Control information is therefore distributed throughout the knowledge scripts (and problem-solving requests) rather than grouped into a single control algorithm. A general-purpose control strategy is still required to co-ordinate the processing involved in instantiating knowledge scripts and fulfilling problem-solving requests. This control strategy is discussed in subsequent sections of this chapter.

4.5.3 Plan Knowledge

```
Type:              PLAN
Applicable-to:     problem description
Plan-Class:        plan-list
Sub-Plans:         plan-list
More-Specific:     criteria

Plan:
      { DO (step-list)
        IF condition THEN DO (step-list)
        [ ELSE DO (step-list) ]
        DO-CONCURRENTLY (step-list)
        DO-ANY-OF (step-list)
        DO-ANY-N-OF N (step-list)
        DO-BEST-N-OF N (step-list)
        DO-WHILE condition (step-list)
        DO-UNTIL condition (step-list)
        DO-TIMES N (step-list) }

   each step can specify an INHERITANCE
   parameter which indicates whether subordinate
   plans should inherit or override the step information
```

Figure 4.7: Frame Structure for Representing Plan Scripts

The plan definition component defines the meta-plan designed to solve the problems to which the script applies. A plan script specifies a set of tasks that are associated with a problem-solving request (or class of requests). The number and complexity of the tasks specified by a plan script will vary according to the complexity of the problem-solving

request(s) to which it corresponds (and the characteristics of the problem domain). Figure 4.7 illustrates the basic structure of a plan script. The APPLICABLE-TO slot specifies a problem description structure which defines the problems (classes or instances) to which the plan information contained in the script is applicable. The PLAN-CLASS slot links a script to its ancestors and enables the script to inherit information (problem descriptions and general plan information). The SUB-PLANS slot links a script representing a class of plans to one or more subclasses or instances of more specific plan scripts (i.e. subordinate plans that are applicable to specific subsets of the general problems to which the script is applicable). The MORE-SPECIFIC slot specifies criteria (e.g. production rules) that can be used to determine which of the subclasses or instances of the script are most applicable based on a given problem-solving request (possibly in conjunction with subordinate and ancestor plan definitions). A plan definition consists of a set of steps which can be grouped using control specifications into sequential blocks (DO), concurrent blocks (DO-CONCURRENTLY), special-purpose blocks (DO-ANY-OF, DO-ANY-N-OF, BEST-N-OF), conditionals (IF-THEN-ELSE), and iteration structures (DO-WHILE, DO-UNTIL, DO-TIMES) that specify the order in which the steps are to be executed. A step specifies a task or set of tasks that fulfills part of a problem solution. These tasks are solved by distributing them in the form of problem-solving requests to appropriate agents using resource knowledge. In some cases, a task may have to be refined using other plan knowledge until a set of more specific tasks is generated which can then be distributed appropriately. Tasks that can be distributed directly using resource knowledge to other agents are referred to as primitive tasks,

while tasks that can be further refined using other plan knowledge are referred to as composite tasks. When a task is distributed to an external agent, that agent's planning component uses its own plan and resource knowledge to plan and distribute the tasks appropriately. Tasks are ultimately executed by appropriate problem-solving components (possibly after being refined and distributed by several agents).

Each step in a plan script can specify an inheritance element which defines the type of inheritance that subordinate plan scripts are to employ: APPEND indicates that the specified task information should be appended to corresponding information in subordinates plans, while OVERRIDE indicates that corresponding information in a subordinate plan script should supersede ancestor information (a subordinate script can override these inheritance specifications if necessary). This inheritance mechanism enables general plan information in ancestor scripts to be combined with specific information in subordinate scripts in order to create appropriate problem-solving plans (or specific plan information to override general plan information).

Plan scripts specify meta-plans consisting of several steps which in turn can consist of several substeps. The ability to manipulate meta-plans facilitates co-operative problem solving because no single agent has to know how to solve an entire problem. Instead, agents can represent problem-solving plans at various levels of detail and co-operate to refine the plan steps until basic problem-solving tasks are determined and subsequently executed by appropriate agents. The plans are not intended to represent a conventional programming language. The steps specified by plans are actually problem-solving requests. A

plan therefore represents a divide-and-conquer method that can be used to solve a particular problem-solving request by dividing it into a set of simpler requests, distributing these requests appropriately, and integrating the results.

Plan scripts which describe classes of problems need not contain a plan definition. These scripts can be used to classify problem-solving requests so that appropriate subordinate plan scripts can be selected.

4.5.4 Task Knowledge

Task knowledge represents a planning component's model of the problem-solving abilities that are contained in its problem-solving component(s) and some of the other agents in the environment. Task scripts are used to determine how the tasks specified by plan knowledge can be performed using the problem-solving abilities of various agents in the environment (i.e. which agents (or classes of agents) can perform specified tasks). Figure 4.8 illustrates the basic structure of a task script. The APPLICABLE-TO slot specifies a problem description structure which describes the tasks to which the script is applicable. The task information specified by the steps in a plan script is used to search the task knowledge hierarchy in order to select task knowledge that describes the agents or class of agents that can perform the specified tasks. In addition, problem-solving requests which correspond to primitive tasks can be processed by selecting the appropriate task knowledge (plan knowledge in these cases is not applicable or necessary).

Type: TASK
Applicable-to: *problem description*
Task-Class: *task-list*
Sub-Tasks: *task-list*
More-specific: *criteria*
Task-Description: *problem description*
Agent-Class: *agent-class-list*
[Agent-ID: *agent-list* **]**

Figure 4.8: Frame Structure for Representing Task Scripts

The TASK-CLASS slot links a task script to its ancestors, enabling it to inherit all or some of the task information. The SUB-TASKS slot links a script to one or more subordinate scripts representing task subclasses or specific instances of a task class. The MORE-SPECIFIC slot specifies criteria (e.g. production rules) that can be used to determine which subordinates of the script are most applicable to the given task.

The AGENT-CLASS slot specifies the class of agents (possibly several) that are capable of performing the tasks described by the script. This information enables a task script to be linked with specific agent classes in the agent hierarchy which can perform the tasks described by the script (an actual agent in the specified agent classes may, however, only be able to perform a subset of the tasks). The TASK-DESCRIPTION slot specifies a problem description structure which stipulates problem attributes that describe the tasks to be performed. These description structures act as templates which can be used to specify or generate specific task information. The task information specified in these task

descriptions is combined with that of the script's ancestors and of the associated plan knowledge (all or some of the descriptions can be inherited, refined, or overridden) in order to describe the tasks that are to be performed (i.e. a detailed problem-solving request that can be sent to an appropriate agent). This enables plan definitions to specify general task information which is then extended appropriately using the information contained in the corresponding task scripts.

The AGENT-ID slot is an optional slot which can specify one or more specific agents (or agent subclasses) that are capable of performing the specified tasks or class of tasks. This slot is used to specify particular agents that can perform given tasks (or these agents should be approached first). When none of these agents can perform the task (busy, disabled, problem constraints cannot be met), then the AGENT-CLASS slot can be used (the general method of selecting an agent) to traverse the agent hierarchy and determine alternate agents that may be able to perform the specified tasks.

4.5.5 Agent Knowledge

Agent scripts represent a planning component's knowledge of the agents in the environment, both of specific agents and of classes of agents. These scripts are used in conjunction with task scripts to describe the problem-solving resources that can be used to perform the tasks specified by plan knowledge. Figure 4.9 illustrates the basic structure of an agent script. The AGENT slot specifies the class of agents or a specific agent that the information in the script describes. The APPLICABLE-TO slot specifies a problem description structure which

provides a general description of the tasks that the agent or class of agents can perform. The task description information specified in plan and task scripts is used to search the agent hierarchy to determine one or more agents or classes of agents that can perform specified tasks. A task script may stipulate specific agents or agent classes that can perform a task, in which case this information is used to categorize the agents to be considered. The problem description of a chosen agent must match the description of the tasks to be performed before an agent can (will) perform the tasks. In some cases, a partial match is sufficient. The planning component would then interact with the agent to formulate the problem description appropriately.

Type:	**AGENT**
Agent:	***agent or agent class***
Applicable-to:	***problem description***
Agent-Class:	***agent-list***
Sub-Agents:	***agent-list*** **[If-needed demons]**
More-Specific:	***criteria***
Communication:	***protocol specifications***
Can-perform:	***agent-specific-task-scripts***

Figure 4.9: Frame Structure for Representing Agent Scripts

The AGENT-CLASS slot links an agent script to its ancestors thus enabling it to inherit all or some of the problem description and general agent information. The SUB-AGENTS slot links a script to one or more subclasses or specific agent instances. IF-NEEDED slot demons can be attached to this slot in order to specify ways of determining actual

agents that belong to a specified agent class. This mechanism is used when specific agents belonging to a class are not known, or known agents cannot perform specified tasks (busy, disabled, problem constraints cannot be met, etc.). The MORE-SPECIFIC slot specifies criteria (e.g. production rules) that can be used to determine the agents (subclasses or specific agents) that are best suited to performing a given task based on specified and inherited task characteristics. The COMMUNICATION slot specifies communication protocols such as routing and packaging information that can be used to interact with an agent or class of agents effectively. Once again, a subordinate agent script can inherit all or some of this information from its ancestors, overriding and supplementing general communication protocols with agent-specific information when necessary.

Each agent script is linked, using the CAN-PERFORM slot, to one or more agent-specific task scripts which define the task description and communication protocols that are associated with each specific task that the agent (or class of agents) can perform (see Figure 4.5). The information in these scripts is combined with that of the agent's general script, its ancestor agent scripts, and any plan and task scripts that have been used to specify the task and agent. As a result, problem and task descriptions are propagated through the plan knowledge, the task knowledge, and the agent knowledge hierarchies in order to determine the tasks required to solve a given problem and the agents in the environment that can perform these tasks (as well as agent-specific information that these agents require to perform the tasks successfully).

4.5.6 Reasoning With The Knowledge

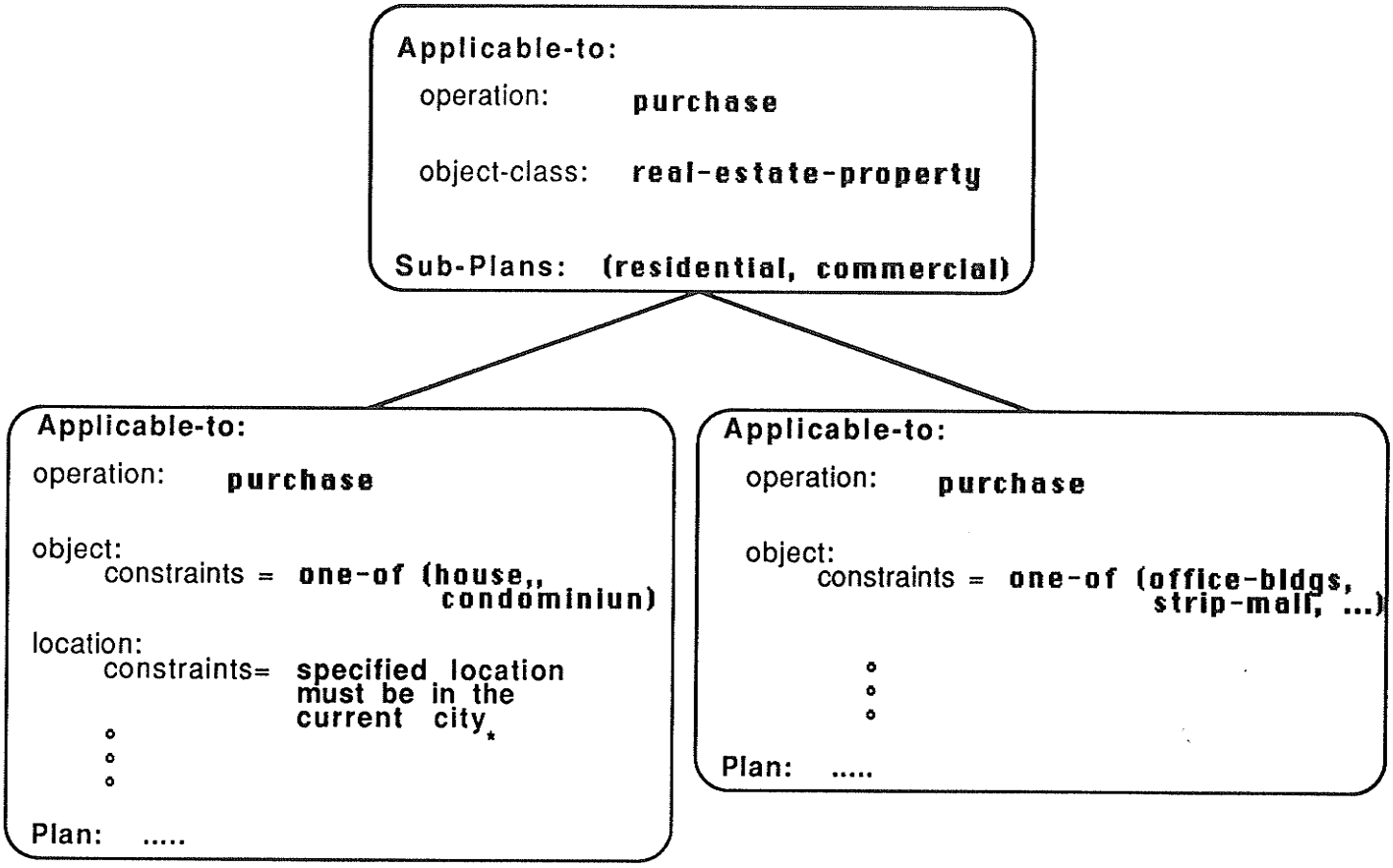
When an agent's planning component receives a problem-solving request, it must select plan and resource knowledge that specifies how to go about solving the given problem. The selection process consists of matching the problem-solving request to the problem description components of the agent's plan and resource scripts in order to instantiate applicable scripts. The plan and resource knowledge scripts are organized into hierarchies so that problem descriptions of scripts representing classes of plans or resources specify general information and subordinate scripts specify more detailed information. As a result, scripts near the top of the hierarchies will match several different, yet related problem-solving requests or task specifications (related by one or more problem attributes), while subordinate scripts will match specific problems or subclasses of problems (all or some of the problem descriptions in general scripts can be inherited or overriden by the problem descriptions of subordinate scripts as specified by the INHERITANCE facet of each slot). Consequently, the planning process involves matching a given problem-solving request (specified by another agent) to the problem descriptions of the plan and resource scripts, starting at the top of the hierarchies (to classify problems and resources) and gradually converging on specific scripts (plans, tasks, and agents). This selection process is examined in further detail in subsequent sections of this chapter.

Consider the problem description in Figure 4.10. This is an example of a problem-solving request that a real-estate agent might receive from a house buyer, describing particular attributes of the type of house

```
OPERATION:    purchase
OBJECT-CLASS: real-estate-property
OBJECT:       (?a house)

LOCATION:     Constraints= ( near a school,
                             not near railroad tracks)
                  unless ( IF   style is acceptable  and
                                price is reasonable
                           THEN location is flexible )

BEDROOMS:     3

PRICE:        Constraints= less-than $100,000

STYLE:        preference (bungalow)
              Constraints= Not (duplex)
```

Figure 4.10: Problem-Solving Request for a House Purchase

that the buyer wishes to purchase (the real-estate agent and house buyer in this case are artificial agents, each consisting of a planning and problem-solving component). This request includes constraints on the location and price that are acceptable to the buyer as well as the style of house that is desired (e.g. three bedroom bungalow). The real-estate agent's planning component would search its knowledge base for plan scripts that describe how to find an appropriate house (based on the description given by the house buyer). Figure 4.11 illustrates that, in this example, the real-estate agent's planning component has a general plan script corresponding to purchasing real estate property and two subordinate plan scripts specifying the tasks to perform in order to co-ordinate the purchase of each type of real estate (residential and commercial). In this case, the planning component would select the script for purchasing residential property.



* the constraint imposed on the location indicates that this plan is not applicable to out-of-town purchases

Figure 4.11: Partial Description of the Real-Estate Agent's Plan Knowledge

The planning component co-ordinates the execution of the plan by searching its knowledge base for resource knowledge scripts that describe how each specified task can be carried out (determine specific task information and the agents who can perform each task). Tasks could include searching home listings, contacting sellers and other

real-estate agents, negotiating a deal, and arranging for financing and transfer of ownership. Resource knowledge might specify that the real-estate agent's problem-solving component should perform some of the tasks (e.g. search home listings, negotiate a deal), and other tasks should be performed by other agents (e.g. the company's Financial Manager is responsible for arranging financing, while a lawyer should perform the legal work involved in transfering ownership). The planning component would then distribute these tasks to the appropriate agents who would carry out the required problem-solving, while the planning component supervises and co-ordinates the integration of the results returned by these agents.

A more extensive example of the co-operative problem solving that the planning components are capable of co-ordinating is presented in the next chapter.

4.6 ACTIVITY BLACKBOARD

The activity blackboard is a composite data structure that acts as the working memory area for a planning component. The blackboard is divided into several partitions each representing information about a particular aspect of the problem-solving activities that a planning component is currently managing. Figure 4.12 illustrates the basic organization of an activity blackboard. The data structures in each partition contain information describing planned activities (tasks) and the problem-solving requests that have been sent to the agents which can perform each activity. In addition, related entries in or among the partitions are linked explicitly. As a result, an activity blackboard

functions as a composite agenda which a planning component uses to represent its problem-solving activities, those currently underway, those pending activation, and those which have been completed together with their results.

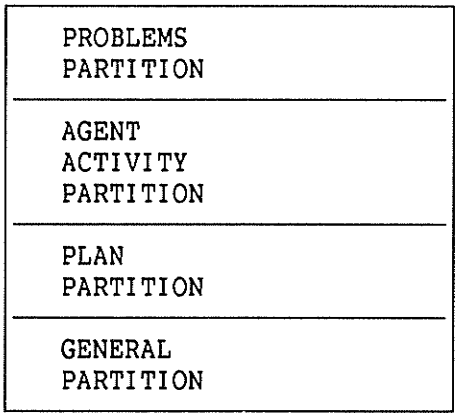


Figure 4.12: Organization of an Activity Blackboard

The next sections briefly describe the processing information that is stored in each partition.

4.6.1 Problems Partition

The problems partition is used to store descriptions of problems that a planning component has been requested to solve. Each entry in this partition is a problem description structure (see Figure 4.6). The entries are used to describe attributes of the problems for which the planning component must plan and co-ordinate appropriate problem-solving activities. The entries are created from the information contained in problem-solving requests that are received on the communications link. The entries can also include justifications for each request.

Justifications can be obtained from a problem-solving request or they can be obtained by interacting with the agent who sent the request. A planning component can use justifications to determine which requests should be processed next. Entries are modified accordingly as new information is obtained, either from the results of problem-solving activities or from interactions with other agents. Results of completed tasks are stored together with their corresponding entries in the problems partition, including relations among the results (e.g. dependencies).

4.6.2 Plan Partition

The plan partition serves as a working area where information generated during the planning process can be stored. This plan information is represented as data structures that describe the problem-solving plans associated with each problem in the problems partition. These data structures consist of plan information and association links. Plan information represents the problem-solving plans which have been created to fulfill given problem-solving requests. This information includes: a list of tasks to be performed, dependencies among tasks, alternative tasks that may be scheduled when any or all of the primary tasks fail, and any other details regarding the execution of a plan. Such information is obtained by manipulating the appropriate plan and resource scripts in the knowledge base. Links to these scripts are represented in each plan partition entry to enable direct access to the plan and resource knowledge associated with a plan. This is useful when all or part of a plan must be regenerated or alternate plans, tasks, or agents must be used.

Association links also are used to relate the encoded plan information with the corresponding entry or entries in the problems partition. A plan partition entry can have multiple links when several entries in the problems partition share common plan information. In addition, other links can associate the entry with the corresponding agent activity partition entries which represent the problem-solving requests that have been sent to the agents who are to perform the tasks specified by the plan. If two or more entries in the plan partition share common problem-solving tasks, then they may share the corresponding entries in the agent activity partition. This avoids scheduling duplicate activities when it is unnecessary. The control strategy is responsible for determining when shared activities occur and for setting up the links among the plan partition entries appropriately (results of shared tasks are distributed to the appropriate problems partition entries as necessary).

4.6.3 Agent Activity Partition

The agent activity partition represents information about the problem-solving activities that the planning component has scheduled. The entries are problem description structures which represent the problem-solving requests that the planning component has sent to its problem-solving component, other agents, or both, requesting that tasks be performed in order to fulfill the computation specified by the plan partition entries.

4.6.4 General Partition

Often it is useful to keep track of general processing information that an agent has accumulated. The general partition of the activity blackboard is used for this purpose. Entries may be posted on this partition to describe information about the general characteristics of the agent's operation such as information describing the processing work load of other agents in the environment and information regarding previous processing that the agent has performed (successfully or unsuccessfully). This information represents dynamic knowledge that has been collected during the agent's operation which may be used to guide the application of plans and selection of agents that can perform planned tasks. Information about disabled or busy agents in the environment enables the planning component to consider alternatives when primary processing involves these agents. Furthermore, information about previously successful and unsuccessful operations can be used to generate the most effective scheduling of processing activities for subsequent problem-solving requests that match or are similar to previously processed requests. As well, the entries in the partition may serve as a dynamic audit trail that is useful when modifications and extensions to the agent's processing abilities are required.

The information that is actually collected and stored in the general partition depends largely on the processing functions that the agent performs and the information that would be most useful in co-ordinating the agent's actions effectively.

4.7 MANAGING DISTRIBUTED PROBLEM SOLVING

4.7.1 Overview

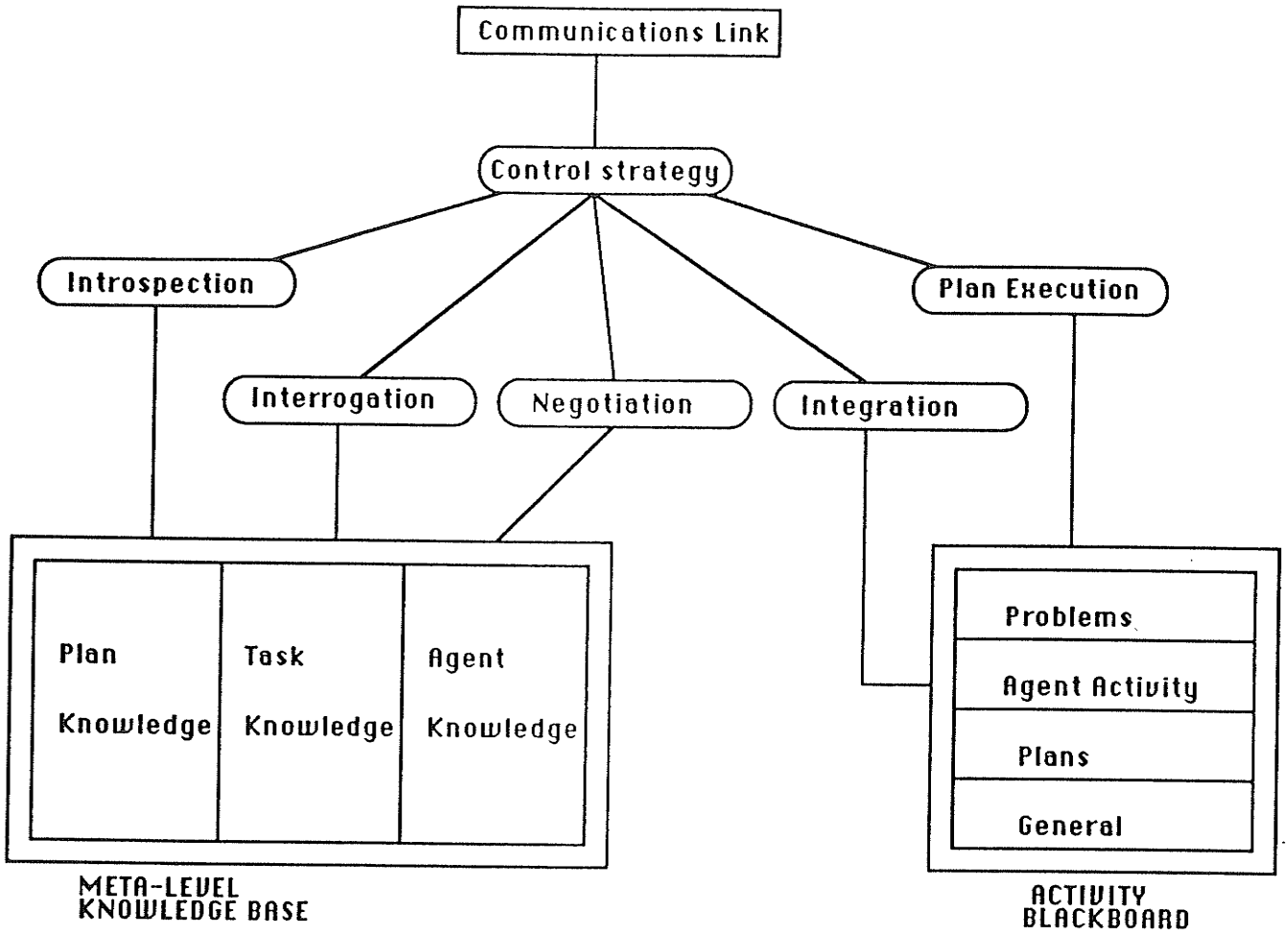


Figure 4.13: Components of the Control Strategy

So far I have examined the knowledge-based component (meta-level knowledge base), the working memory component (activity blackboard), and the input/output component (communications link). A planning component

also contains a control strategy which controls the meta-problem solving required to plan and co-ordinate co-operative problem-solving activities. The functions performed by the control strategy are illustrated in Figure 4.13.

The control strategy monitors the communications link for any problem-solving requests that the agent may receive. When problem-solving requests are received, the control strategy represents them as entries in the problems partition of the activity blackboard. An entry is then selected from the problems partition to become the active problem. The control strategy then performs introspection to select plan knowledge corresponding to the active problem. The selected plan knowledge is used to construct a plan specifying a set of tasks to be performed in order to solve the problem. The control strategy interprets the plan information and performs introspection to select resource knowledge describing how each task specified in the plan can be executed (the agents that can perform each task and the problem-solving requests that must be sent to these agents). The selection of plan knowledge may not be necessary if the active problem represents a primitive task; in this case, the control strategy will create a special plan by selecting the appropriate resource knowledge. The control strategy may have to interrogate agents to determine whether or not they can perform specified tasks, or it may have to co-ordinate negotiations with agents in order to reformulate parts of problem-solving requests or planned tasks that initially are incompatible. The objective of the planning process is to determine the necessary problem-solving activities (autonomous and/or co-operative) and to initiate interactions

with agents who can perform these activities (by sending these agents problem-solving requests corresponding to the activity they are to perform).

After the specified tasks have been distributed to appropriate agents, the control strategy returns to monitoring the communications link in order to supervise the execution of the planned activities. This involves waiting for responses to scheduled activities, starting negotiations when difficulties occur, and processing pending plan steps when scheduled activities are completed successfully.

Co-ordinating the execution of a plan may also require performing integration functions that take knowledge generated by the agents and integrates it with existing knowledge. Integration may be required during the execution of a plan to generate information that is required in subsequent plan steps. Integration may also be required to generate a final solution to a problem from the results produced by the agents that performed the planned tasks.

In the remainder of this section I examine each of the control functions discussed above. These are the basic functions that the control strategy can apply to plan and co-ordinate problem-solving activities.

4.7.2 Introspection

The control strategy performs introspection to search the meta-level knowledge base in order to select plan knowledge that is applicable to given problem-solving requests. This processing causes applicable plan

knowledge to be selected by matching the information in a given problem-solving request to the problem description component of the plan scripts in the plan knowledge hierarchy. Plan scripts which are instantiated successfully represent plan knowledge that is applicable to the problem-solving request. This instantiation process begins at the top (root) of the plan knowledge hierarchy where one or more general plan scripts are selected, resulting in the classification of the problem-solving request. The process continues using the criteria in the MORE-SPECIFIC slot of these general plan scripts to determine which of their subordinates are applicable to the given request. In some instances, introspection may identify plan scripts which are only potentially applicable; their problem description structures are only partially instantiated and the information in the APPLICABLE-WHEN slot of their problem description structure suggests, with some certainty, that they are applicable. In order to confirm the applicability of these plan scripts, the control strategy must perform additional processing such as interrogating the agent who sent the problem-solving request for additional problem information or negotiating with the agent to resolve conflicts between its problem-solving request and the potentially applicable plan scripts (control information in the SOLVED-WHEN slot of the problem-solving request and in the APPLICABLE-WHEN slot of the plan scripts is used to co-ordinate the instantiation process).

The control strategy also uses introspection to select resource knowledge that is applicable to the tasks specified by a problem-solving plan. In this case, introspection uses the information specified in a

problem-solving request in conjuction with the task descriptions specified by plan steps in order to search the task and agent knowledge hierarchies for applicable resources. The search may begin at the root of the resource hierarchies or at a specific branch (class of tasks or agents) within the hierarchy depending on the information specified in the plan steps. The search continues using the criteria in the MORE-SPECIFIC slot of these scripts to determine which of their subordinates are applicable. Once again, introspection may identify several potentially applicable resources, in which case the control strategy must perform additional processing to confirm their applicability and to determine which resources are most applicable. Preferences may be represented in the knowledge scripts (i.e. in the criteria specified in the MORE-SPECIFIC slots) or the control strategy may select a resource at random. In either case, if the primary resource cannot perform the task, then the control strategy can arrange to have one of the other resources attempt to perform it.

Therefore, introspection enables the control strategy to search the meta-level knowledge base for knowledge describing the tasks required to solve given problems, the agents who can perform these tasks, and the protocols necessary to interact with these agents effectively. It is essential that control knowledge necessary to determine when a plan, task, or agent is applicable be distributed to the knowledge scripts rather than hardcoded in a large and cumbersome control algorithm; control becomes more understandable and thus easier to modify and extend because the control knowledge is represented explicitly as is the context in which the knowledge is applicable.

4.7.3 Interrogation

Interrogation is when an agent asks another specific agent, group of agents, or all agents for more information about a problem-solving request or a planned task. An agent interrogates other agents in order to determine whether or not the agents can (will) perform specified tasks. A successful interrogation will result in responses from the interrogated agents indicating whether they can or cannot perform the tasks. These responses may include a description of problem attributes that are required in order for the agents to perform the tasks. This information is used to reformulate problem-solving requests or task descriptions so that they can be fulfilled successfully. An agent that sent a problem-solving request can also be interrogated when additional problem attributes omitted in the original request are required to instantiate plan and resource knowledge.

When an agent interrogates other agents, it may make several types of inquiries. A general inquiry would entail asking other agents whether or not they can perform specified tasks. This type of inquiry is used when the planning component does not have resource knowledge describing the agents that can perform a specified task (it may know the class of agents that can perform the task but not a specific agent). A more specific inquiry might involve questioning specific agents for more detailed information about given tasks. This type of inquiry would be used when the planning component has partial knowledge of another agent's ability to perform tasks and requires additional information to formulate an appropriate interaction with the agent. The information received from the interrogated agents can be added to the appropriate

knowledge script to alleviate the need to interrogate the agents during subsequent processing that involves the same or similar tasks (this requires an application-specific learning mechanism that can determine when and how the knowledge should be updated).

Therefore, the interrogation operator enables the control strategy to obtain additional information about problem-solving requests that it receives and provides it with the ability to ask other agents for more information about their problem-solving abilities and the protocols that must be used to have them perform various tasks. This function enables a planning component to expand its plan and resource knowledge dynamically during co-operative problem-solving activities. Information that is acquired through experience normally must be verified before it is used in subsequent processing (by interrogating the agents involved or by applying an application-specific learning mechanism).

4.7.4 Negotiation

Negotiation is the process by which agents attempt to reformulate co-operatively problem-solving activities that initially are incompatible. Incompatibilities occur when constraints imposed by the agents involved cannot all be satisfied (i.e. one or more problem attributes are incompatible with the plans and resources that the control strategy can employ). Agents negotiate by communicating with one another in order to exchange information that can be used to relax constraints or reformulate problem-solving requests or task descriptions to satisfy the constraints.

Negotiation can occur in two ways: a planning component can ask the agent which sent a problem-solving request whether part of the request can be modified so that incompatibilities with partially applicable plan and resource knowledge are resolved; and a planning component can ask another agent that is capable of solving a portion of a problem-solving request whether the task description that it requires can be modified so that it matches the attributes of the given problem-solving request. Knowledge describing applicable reformulations is typically represented in requests and in the preconditions of the knowledge scripts (typically the UNLESS option of a constraint facet will specify one or more relaxation methods that can be used to satisfy the constraint). In either case, the control strategy can send the agents a reformulated problem-solving request or task description and ask that they verify that the reformulated description is acceptable or reply with an indication of the information that is or is not acceptable so that further negotiations may be performed. Negotiation is carried out until all parties involved are satisfied with the description of the problem-solving that they are to perform. The amount of negotiation required will vary according to the capabilities of the agents involved. In addition, the agents involved in negotiation must be able to detect when further negotiations will be unfruitful in order to avoid unnecessary expenditures of resources and to avoid deadlocks during negotiations. Knowledge describing the situations in which negotiations should be terminated should be maintained in the knowledge scripts used to drive the negotiations.

The agents' ability to negotiate prevents problem solving from being restricted to all-or-nothing scenarios. The agents can negotiate with one another to determine acceptable or mutually beneficial problem-solving activities so that problem-solving can be performed successfully. After successful negotiations, the control strategy will modify the plan and resource knowledge scripts appropriately so that subsequent problem solving can take into account any potentially negotiable attributes (e.g. create new slots in the problem description structure of a script describing additional problem attributes or add new information to the APPLICABLE-WHEN slot that can be used to determine when a script is applicable). This enables an agent to increase its knowledge of the abilities of the other agents in the environment by representing explicitly any and all plausible means of negotiation that may be used when incompatibilities occur. As a result, the robustness of the agents is increased significantly.

4.7.5 Plan Execution

Plan execution involves applying plan information in order to co-ordinate the co-operative problem-solving activities associated with given problem-solving requests. Before applying a plan, the control strategy may have to perform introspection to retrieve additional plan knowledge that can be used to refine composite steps of the plan into a set of primitive tasks. This refinement process can be applied repeatedly to any or all of the steps in the plan. Composite plans are continually refined, not necessarily all at once, until executable plans are produced (plans that consist of primitive plan steps) or the planning component determines that the request cannot be satisfied.

Executable plans are stored as entries in the plan partition of the activity blackboard. If several executable plans are applicable, then the control strategy must select the most appropriate plan and the other plans are used as alternatives which can be applied if the primary plan fails. The most applicable plan may be the plan which was selected with the greatest certainty during introspection or it may be identified explicitly by knowledge in the plan hierarchy (e.g. criteria in the MORE-SPECIFIC slot of the plan scripts).

The next phase of plan execution requires the control strategy to perform introspection to select resource knowledge describing agents that can perform each step of an executable plan. A step can be executed either by distributing it to the agent's own problem-solving component or to another agent. If the resource knowledge does not specify agents who can perform a task (it may specify a class of agents), then the control strategy will have to perform interrogations of agents to determine which agents are capable of performing the task. If more than one agent can perform the task, then the control strategy must determine which agent is the most appropriate. Choosing an agent may require dealing with constraints and possibly performing negotiations with the agents.

Tasks are distributed by sending messages containing problem-solving requests to the appropriate agents. Entries representing problem-solving requests that have been sent to agents are stored on the agent activity partition of the activity blackboard. Tasks which are distributed to the problem-solving component will be executed using appropriate problem-solving knowledge, while tasks which are distributed

to other agents are processed by the planning components of these agents which can apply any of the control functions to refine, distribute, or execute (using their problem-solving components) all or some of the tasks. This process is repeated for each step that can initially be executed. Further planning will be performed as subsequent steps in the executable plans are processed (some steps may be contingent on preceding steps). Steps which are not applicable may, using the FAIL element, trigger a plan to be aborted, in which case alternate plans can be used or negotiation can be initiated.

Therefore, plan execution consists of applying plans and co-ordinating the execution of the tasks specified by the plans. This may involve reasoning about alternative plans and resources when conflicts or other difficulties arise. The ability to reason about alternative plans that can be used to carry out problem-solving requests and about alternative resources that can be used to execute the tasks specified by plans is of the utmost importance because it increases the robustness of an agent's problem-solving capabilities significantly.

4.7.6 Integration

Integration involves taking knowledge generated by agents and integrating it with existing knowledge. This process enables the results of planned problem-solving activities to be combined and used by subsequent plan steps or to produce final solutions to problems. Integration typically will involve ensuring consistency among the results of plans steps to determine whether these partial solutions can be combined into a final solution or whether the plan should be

terminated. The types of integration functions that the control strategy can perform will vary and can include: merging, summarizing, pruning, and augmenting intermediate or final results. Furthermore, the methods used to perform the integration functions can also vary. A planning component may be able to perform some basic integration functions itself and can distribute more complex integration functions to its problem-solving component or to other agents.

4.7.7 Co-ordinating Control

The control strategy must process several types of communications that may be received on the communication link. When a problem-solving request is received, the control strategy must decipher the message, create an entry in the problems partition of the activity blackboard, and co-ordinate several operations including: introspection to determine whether the problem can be solved, and, if so, which plan and resource knowledge should be used to solve it; creation and execution of executable plans that specify the tasks to be performed; negotiation with or interrogation of agents to determine how to distribute the planned tasks; and integration of the results produced by these agents.

In addition, subsequent communications may arrive that represent responses to tasks which have been distributed to other agents. Responses may indicate failed tasks, in which case the control strategy may have to initiate negotiations to attempt to reformulate the tasks, or it may have to invoke the plan execution function to determine if alternate resources can be used to perform the task. On the other hand, a response may represent results produced by another agent. In this

case, the control strategy may have to integrate the results with existing information, or it may initiate additional plan execution to plan and schedule the next steps of a plan which were waiting for the given results (i.e. results which satisfy preconditions of pending plan steps).

The control strategy therefore handles all aspects involved in supervising the construction of meta-problem-solving plans and the execution of the tasks specified by these plans. This may involve co-ordinating activities for several problem-solving requests that the agent has received. In this case, the control strategy will have to supervise several active plans and must co-ordinate the processing associated with each (e.g. processing of responses and scheduling steps associated with each plan). In addition, the control strategy may have to supersede the execution of one plan in favour of another when problem-solving requests with higher priority are received (e.g. emergency requests).

4.8 SUMMARY

In this chapter I presented a conceptual model of distributed problem solving. The fundamental principle in the model is the separation of basic problem solving from distributed problem solving. The model accomplishes this by dividing basic agents into two components: a problem-solving component and a planning component. Problem-solving components perform problem-solving functions that carry out basic computation, while planning components perform problem-solving functions that plan and co-ordinate distributed problem-solving activities among the agents.

Each planning component embodies a knowledge-based model of problems that it knows how to solve and the problem-solving resources (local/external) that it can use to solve each problem. Employing a knowledge-based approach provides a more understandable, extendable, and robust representation of an agent's distributed problem-solving functions than a procedure-based approach would because the knowledge is explicit and the application of the knowledge is not predefined (the control strategy determines dynamically the knowledge that can be used to co-ordinate co-operative problem solving). Each agent in the environment is able to perform a set of assigned tasks using its problem-solving component, and can reason using its planning component about how it can contribute to the problem-solving activities of other agents and how other agents can contribute to its own problem-solving activities.

Appendix A provides a detailed examination of the engineering issues involved in creating an implementation of the conceptual model of distributed problem solving outlined in this Chapter.

Chapter V

APPLYING THE MODEL

5.1 OVERVIEW

This chapter illustrates the application of the model of distributed problem solving presented in the previous chapter to problem solving in the medical domain of cancer diagnosis and treatment. The complexity inherent in the domain prevents any single agent (health-care professional) from performing or controlling all of the required problem-solving activities. Instead, the agents must co-operate to perform distributed problem solving which enables them to distribute tasks and subsets of tasks to other agents in order to overcome local problem-solving limitations (i.e lack of skill, knowledge, and time). The examples presented in this chapter illustrate that in order to perform distributed problem solving, the agents must have an explicit, meta-level model of the environment and the ability to reason about how they can co-operate with one another based on this model. The goal of the chapter is to demonstrate that it is the embodiment of an explicit distributed problem-solving mechanism in each agent that enables the agents to perform co-operative problem solving effectively.

5.2 THE DOMAIN

Oncology is generally referred to as the 'cancer problem' [Rubin 83]: the study of the diagnosis and treatment of malignant tumors that can arise in various areas of the human body. In our society, deaths from cancer are exceeded only by those resulting from cardiovascular disease. Almost one in every four Americans now living will be diagnosed as having cancer within his/her lifetime [Rubin 83]. Furthermore, at current rates almost two thirds of cancer patients will probably die of their disease [Rubin 83]. Although there is no known cure for all cancers, there have been, and continues to be, considerable advances in the areas of early diagnosis and effective treatment [Old 88]. This has led to an increase in the effectiveness with which the disease can be battled, and pronounced decreases in the lethality rate of many cancers (uterus cancer, stomach cancer, and liver cancer) [Rubin 83]. Nevertheless, the death rates associated with many cancers continue to rise (lung cancer, prostate cancer, pancreatic cancer) [Rubin 83]. Much research has been directed toward determining the mechanisms by which cancer cells proliferate and spread and finding improved diagnostic and treatment methods [Beck 85].

The complexity inherent in cancer diagnosis and treatment dictates the need for co-operative problem solving. The numerous types of cancers, multiple occurrences of cancers (possibly different types) in several areas of the body, and the extensive set of testing and therapy procedures required to diagnose and treat patients necessitates co-operation among several specially-trained health-care professionals (agents). No single agent, not even the most knowledgeable oncologist

(a physician who specializes in the diagnosis and treatment of cancer), can effectively perform all of the tasks associated with the diagnosis and treatment of cancer patients. This would require an enormous amount of knowledge, far more than is humanly possible to retain and apply effectively. As a result, problem-solving responsibilities must be divided among several agents, each of whom possesses skills and knowledge that are applicable to some subset of the domain tasks. The skills embodied in the agents will vary and the tasks to which each can contribute will vary accordingly. For example, an oncologist is capable of diagnosing cancer, selecting an appropriate treatment plan, and managing the patient's progress during treatment. A lab technician does not have the knowledge and training to diagnose a patient, plan treatment, or manage treatment, but is capable of performing a specific set of laboratory tests (blood, urine, tissue, etc.) that contribute vital information so that the oncologist can make diagnosis and treatment decisions.

Another advantage in having the agents perform distributed problem solving is that it enables the agents to co-operate in order to utilize the resources in the environment effectively. This is often advantageous even when an agent is capable of performing a task because it enables that agent to concentrate on more important tasks which require his advanced skills. For example, an oncologist can distribute the task of administering a patient's treatment to a oncology nurse or clinician so that he can concentrate on diagnosing and planning the treatment for other patients. This enables high-level skills which are normally very scarce and expensive to be utilized to their fullest,

while less demanding tasks are carried out by less skilled agents. The agents may be less skilled but are still vital to the overall environment - they reduce the burden on skilled agents. Similarly, the agents can co-operate to ensure that available equipment (e.g. CTscan machines) is used effectively. Agents can schedule tests and treatments that require scarce and expensive equipment co-operatively in order to resolve conflicts and, more importantly, avoid periods of under-utilization.

The following sections describe a model of the basic activities involved in diagnosing and treating cancer patients and the agents who co-operate to perform these activities. A general overview of the domain, the agents involved, and the tasks that each performs, is presented. The model illustrates the complexity involved in diagnosing and treating cancer patients and the co-operative problem solving that the agents utilize in order to cope with this complexity. A more extensive description of oncology can be found in [Devita 85].

5.3 AGENTS

Figure 5.1 illustrates the collection of agents who participate directly in cancer diagnosis and treatment (administrators, secretaries, etc. have not been included). The agents are organized into structured groups (agencies) according to their skills and relationships with other agents. The agents have well-defined roles that are based on their problem-solving abilities. In addition, each agent is capable of interacting with some of the other agents in its own agency or in other agencies either directly or indirectly. Interactions are also

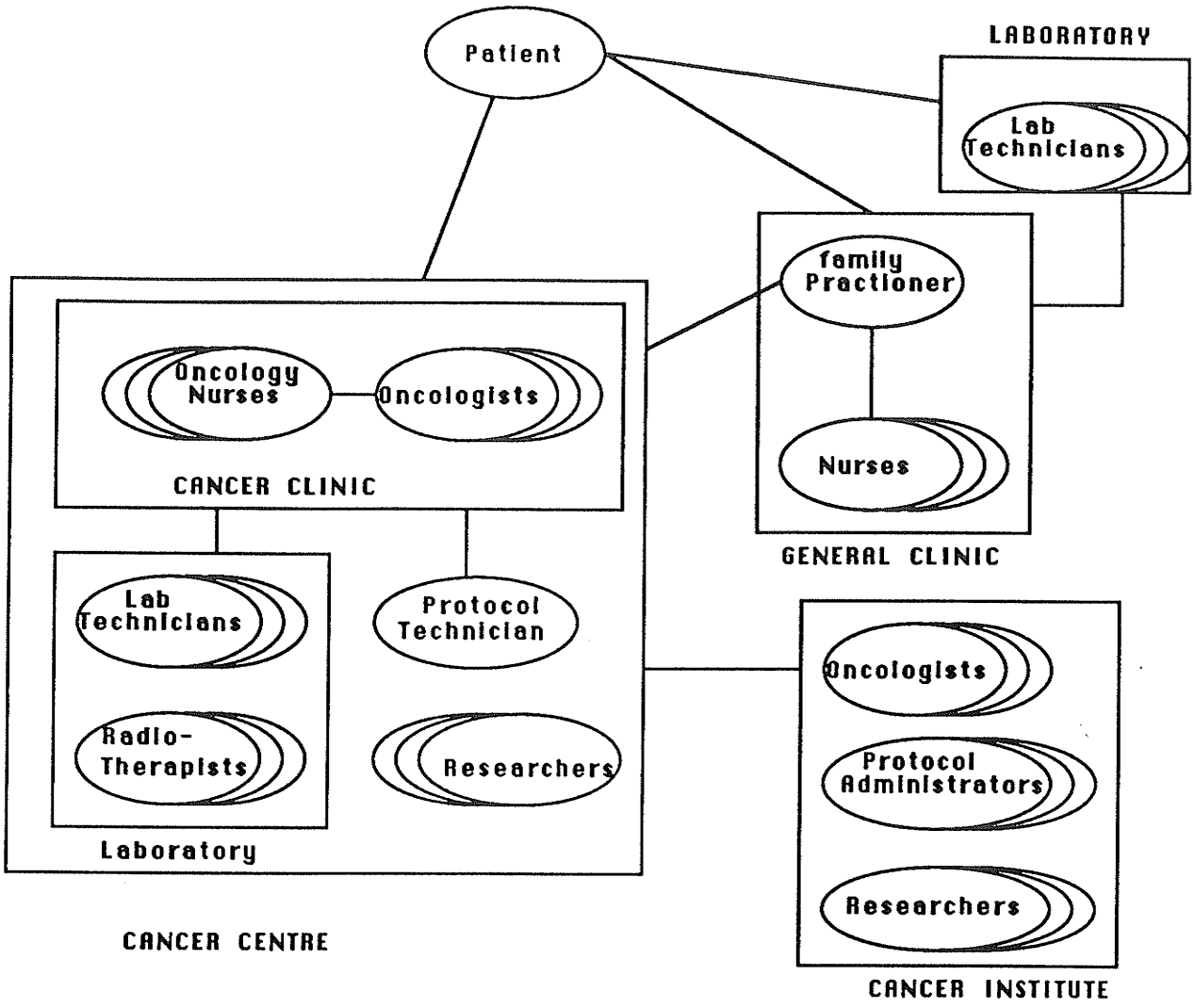


Figure 5.1: Organization of Agents Involved in Cancer Diagnosis and Treatment

well-defined based on the tasks that must be carried out and the contributions that each agent can make to performing those tasks. Initially, only a small subset of the agents will participate, but as more specialized skills are required, additional specially trained

agents will be brought into the process. As a result, the agents must work co-operatively to perform the numerous complex tasks necessary to diagnose and treat cancer patients.

The three basic types of agencies involved in diagnosing and treating cancer patients are: general medical clinics, cancer centres, and cancer institutes. General clinics are designed to provide primary health care and usually consist of one or more physicians and several nurses. The physicians are classified as family practice specialists because they are trained to provide comprehensive family health care. Nurses perform basic health care functions that aid the family practice specialists (e.g. manage patient histories, perform basic tests, administer injections). These clinics use laboratories to analyze test samples and perform special tests (e.g. X-rays, blood tests). The laboratories are often autonomous entities that are shared by several clinics; patients are sent to the labs to have various tests administered and analyzed, and test samples taken by clinic personnel are sent to the lab for analysis.

A cancer centre is a special-purpose medical agency which specializes in cancer diagnosis, treatment, and research. Within each cancer centre is an agency referred to as a cancer clinic which is responsible for diagnosis and treatment of cancer patients. Cancer clinics may also reside outside of a cancer centre (e.g. rural regions), but they normally are affiliated with a cancer centre. The majority of the diagnostic and treatment tasks can be performed in these clinics, but some tests (CTscans) and therapies (radiotherapy) must be performed at a cancer centre. A cancer centre consists of several specially trained

agents, many of whom reside in the cancer clinic component of the centre, including: oncologists, oncology nurses, lab technicians, and a protocol technician [Singer 86]. Oncologists are physicians who specialize in cancer diagnosis and treatment, while oncology nurses are specially trained in the administration and management of cancer treatment. Oncologists frequently specialize in a particular type of cancer such as breast cancer, lung cancer, or leukemia. Each centre has a protocol technician who co-ordinates the treatment plans used to treat the different types of cancer. In addition, each cancer centre usually has its own laboratory where tests can be performed and analyzed (basic health and cancer-specific tests) and special therapies can be administered (e.g. radiotherapy).

A cancer institute is a national agency responsible for creating, distributing, and monitoring treatment plans. Clinical trials compare the effectiveness of treatment plans (standard plans and newly developed plans) in order to determine the treatment plans that are most effective in treating a particular type of cancer (a specification of doses and frequency of a therapy that is accepted as being applicable to a particular type of cancer is referred to as a standard treatment plan). The formal definition of a clinical trial, the eligibility criteria for participating in the trial, the treatment plans used in the trial, and the data collection requirements are referred to as a protocol [Scuse 88]. Cancer institutes monitor the treatment of cancer patients participating in the clinical trials in order to analyze the successfulness of the treatment plans and to create improved plans. The staff of a cancer institute include highly qualified oncologists who

supervise the development and administration of standard treatment plans, protocol administrators who register patients in clinical trials and collect clinical data, and researchers who analyze the data collected from clinical trials. These institutes can be consulted to determine whether or not a patient is eligible to participate in a clinical trial and to verify any treatment modifications that may be necessary when a patient experiences adverse reactions to the therapies prescribed by a treatment plan of a clinical trial.

5.4 ACTIVITIES

5.4.1 Overview

Figure 5.2 illustrates the basic activities that are performed during cancer diagnosis and treatment [Rosenthal 87]. The process normally begins with a family physician examining the patient. When a potential cancer problem is detected, the physician will normally order special tests (such as a biopsy). If a diagnosis of cancer is determined or suspected, the physician will refer the patient to a cancer centre. The patient's first visit to the cancer centre consists of several tests and consultation with an oncologist. If the oncologist confirms the diagnosis, then appropriate therapies are determined. The therapies for cancer include surgical removal of some or all of the tumor, radiotherapy, and/or chemotherapy. The therapies may be curative (to remove the tumor), prophylactic (as a preventative measure), or palliative (to reduce the suffering but without curing) [Rosenthal 87]. Establishment of a treatment plan (appropriate therapies) is carried out by the oncologist, after which initial therapies may be administered. The patient is then scheduled for future visits at which time the

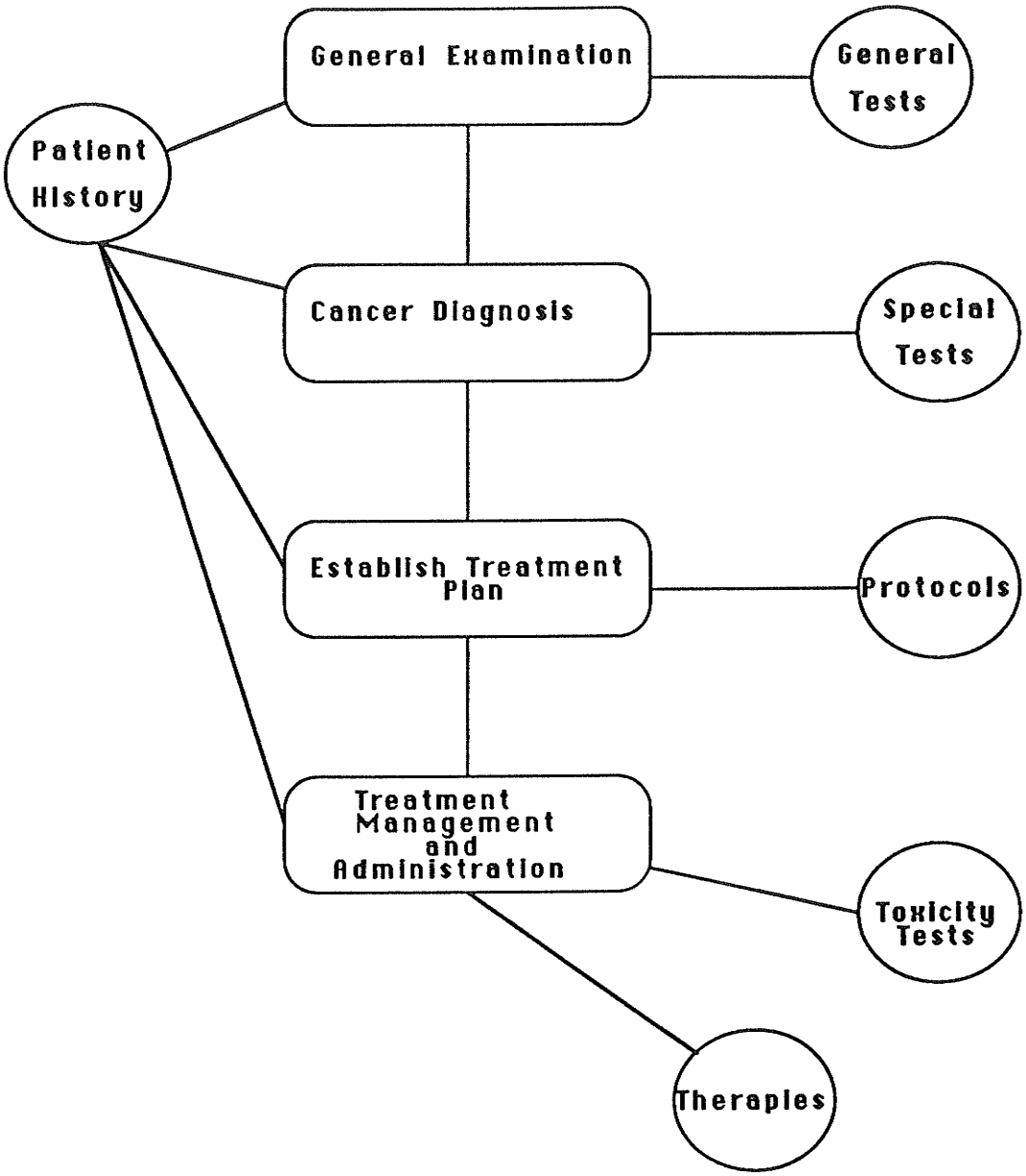


Figure 5.2: Basic Cancer Diagnosis and Treatment Activities

administration of the prescribed therapies will occur. During these visits, the patient's status is monitored and the treatment plan is modified when problems arise. Treatment administration and management

continue for the duration of the treatment plan or until the oncologist determines that the treatment is no longer necessary or useful.

Each of the activities described above is examined in more detail in the remainder of this section, the goal being to define a rudimentary model of cancer diagnosis and treatment. Many of the tasks involved in each of the basic activities have been simplified for the reader who is unacquainted with oncology.

5.4.2 General Examination

The potential presence of cancer is typically detected during a clinical examination performed by a family physician. Early warning signs such as dizziness, headaches, abnormal pain, shortness of breath, or the appearance of lumps may prompt a patient to visit a physician (potential problems are also often detected during annual checkups). The doctor performs a general examination of the patient using the symptoms the patient is experiencing to guide the initial diagnosis process. A family physician, however, may not be able to make a firm diagnosis; instead, the physician interprets the results of the examination to determine whether or not cancer is potentially present. If necessary, the physician arranges for special tests to be to be administered (e.g. biopsy). Signs such as abnormal lumps, shadows on X-rays, and symptoms that cannot be explained can prompt the physician to refer the patient to a cancer centre (assuming that the signs indicate a potential cancer problem). In order to refer a patient, the physician must contact a cancer centre (possibly a specific oncologist at a centre) and forward the patient's history (test results, symptoms, etc.). The patient is

then given an appointment at the cancer centre which takes over the diagnosis and treatment tasks. The patient's physician receives regular reports detailing the treatment because the family physician remains responsible for the patient.

5.4.3 Diagnosis at a Cancer Centre

A patient's first visit to the cancer centre consists of a clinical examination followed by a consultation with an oncologist. The oncologist uses the patient's health history and consultations with the patient's family physician to acquire an initial summary of the patient's status. The oncologist uses this information and interpretations of the clinical examination results to make a diagnosis. Selecting a diagnosis may require consultations with other oncologists (in the centre, other cancer centres, or cancer institutes), or indicate the need for further tests that can provide additional information. For example, a female patient with a lump in her breast may initially have a mammogram performed (breast x-ray) and then a breast aspiration (extraction of liquid from the tumor) or possibly a biopsy (surgical removal of a portion of the tumor) in order to make a complete and accurate diagnosis. A diagnosis will normally categorize a patient according to the type of cancer that is present and possibly the stage to which the cancer has progressed (based on the size of the tumor, spread of the disease, etc.) [Scuse 88].

5.4.4 Establishing a Treatment Plan

A suitable treatment plan must be selected once a patient has been diagnosed. Normally, an oncologist will first try to establish a patient in a clinical trial based on the patient's diagnosis. The oncologist selects an appropriate protocol and instructs the protocol technician to register the patient with the cancer institute sponsoring the protocol. The cancer institute will verify the patient's eligibility to participate in the clinical trial according to the eligibility criteria specified by the protocol (e.g. age restrictions, tumor size and location, stage of cancer, and the general health status of the patient). If the patient is eligible, the cancer institute will randomize the patient into one of the treatment plans specified by the protocol. The cancer institute will inform the protocol technician of the results of this process and the protocol technician will then in turn inform the oncologist.

A patient who is not eligible to participate in a particular clinical trial may still be treated using a treatment plan used within the trial, although the patient will not be part of the trial. An oncologist may also decide to treat a patient using a treatment plan from a previously completed clinical trial (a standard treatment plan), or the oncologist may decide to create a special-purpose treatment plan or to apply a modified version of a standard treatment plan (plans of this sort are referred to as non-standard treatment plans).

Once a treatment plan is established, the prescribed regimen of therapies is generated. The treatment plan dictates the schedule of tests and therapies that are to be administered, the frequency of

therapy, and any constraints that must be satisfied before each therapy can be administered. The oncologist explains the treatment plan to the patient, including the treatment methods and goals and the schedule of future visits to the centre when therapies will be administered.

5.4.5 Treatment Administration and Management

The administration of a patient's treatment plan is performed during regular visits to the cancer centre. The patient undergoes a series of prescribed tests (blood, urine, etc.) before therapy is administered. These tests are used to assess the patient's status and determine whether the patient is having any adverse reactions to therapy. Many of the therapies (especially chemotherapy) in cancer treatment must be administered in doses that are almost intolerable to the patient and the therapies may lead to varying degrees of toxicity in the patient. As a result, the status of the patient must be monitored closely in order to ensure that the patient is not seriously compromised by the therapy, but that the therapy the patient is receiving is effective. The patient's treatment plan may have to be modified if the results of the toxicity tests indicate that toxicity problems have occurred. Treatment modifications can include reducing dosages of therapies, delaying a therapy, or omitting a scheduled application of a therapy until toxicity levels return to acceptable levels (in severe cases treatment may be terminated).

The management of therapy is supervised by an oncologist and the administration of therapies is normally performed by an oncology nurse based on the plan management and modification criteria prescribed by the

patient's treatment plan. If toxicity problems arise, the oncology nurse consults with the oncologist to determine whether treatment modifications are necessary, and, if so, the modifications that are appropriate. In extreme cases, the oncologist may have to consult with the cancer institute to determine the modifications that can be made if a patient develops severe reactions to therapy. This ensures that patients participating in a clinical trial are treated using acceptable modifications so that the data collected will not bias the clinical trial.

After the patient's status has been assessed and the treatment for that day has been determined, the nurse administers the prescribed therapies using any modified dosages that are required (possibly omitting or delaying one or more therapies). Chemotherapy is normally administered by an oncology nurse, while radiotherapy is administered by a radiotherapist. After all therapies have been administered, the patient is informed of the next visit and the patient's chart is updated accordingly. If severe toxicity problems were detected, then special visits are scheduled at which time toxicity levels are tested and delayed therapies are administered (if appropriate).

5.5 SNAPSHOTS OF DISTRIBUTED PROBLEM SOLVING IN THE DOMAIN

5.5.1 Overview

This section examines the application of the model of distributed problem solving to co-operative problem solving in oncology. Each agent in the domain is viewed as embodying a planning component and a problem-solving component. An agent's planning component co-ordinates

the problem-solving tasks that its problem-solving component should perform and those tasks that should be performed by other agents. This discussion is not intended to imply that the human agents discussed in the previous section are comprised of a planning component and a problem-solving component. Instead, it attempts to illustrate how the co-operative problem solving that human agents perform can be achieved using the model of distributed problem solving presented in Chapter 4. Therefore, references made to agents within this section such as patients, family physicians, and oncologists refer to the artificial agents created using the model; these agents carry out the tasks assigned to their human counterparts. The goal is to demonstrate that these agents must possess plan and resource knowledge and the ability to manipulate this knowledge in order to determine when and how they can co-operate to perform the various tasks involved in diagnosing and treating cancer patients.

Examples are taken from each of the basic domain activities: general examination, cancer diagnosis, treatment plan establishment, and treatment administration and management. Each example is intended to illustrate some of the plan knowledge used to determine the tasks that must be performed, and some of the resource knowledge (tasks and agents) that describes how these tasks can be carried out (either by an agent itself or in co-operation with one or more other agents). The process of refining an agent's composite plan steps to one or more primitive tasks is examined, as is the process of applying resource knowledge to determine how (by whom) each primitive task can be performed. The examples also illustrate that a primitive task may actually be viewed as a composite task by the agent to which it is distributed.

Snapshots of plan, task, and agent knowledge are presented in several diagrams throughout this section. The examples are not intended to be exhaustive, nor is the knowledge and reasoning illustrated in the examples intended to be complete; however, it serves to illustrate the basic structure, content, and application of the knowledge that enables the agents to perform co-operative problem solving (in practice, each knowledge script would contain significantly more extensive information). For instance, the problem description elements of the knowledge scripts provide a summary description of the types of problems, tasks, and agents to which a script is applicable (this information would normally be represented in an extensive frame structure as outlined in Chapter 4). The control information for determining when the knowledge scripts are applicable and how to negotiate reformulations of the problem descriptions so that incompatibilities that prevent a knowledge script from being applied can be resolved, have been omitted from the scripts. The steps of each plan script are represented in a task and attribute-list form, the plan steps would in fact be represented using problem description structures. The task description slot in the task scripts has also been omitted, but again would be represented using a problem description structure which would be instantiated using the problem attributes from the appropriate plan steps (this enables the task knowledge to specify additional information about the tasks to be performed).

The problem-solving components of the various agents are not examined, it is assumed that each problem-solving component possesses appropriate knowledge and reasoning mechanisms to perform the tasks

that are distributed to it by its planning component. In addition, the activity blackboards that each agent's planning component uses to maintain an agenda of the problem-solving activities currently underway are not discussed: it is assumed that the planning components will maintain and manipulate their activity blackboards appropriately. The goal of this section is to demonstrate, using snapshots of distributed problem solving, that it is the agents' ability to embody and manipulate explicit meta-level knowledge (plans, tasks, and agents) that enables them to perform co-operative problem solving effectively.

5.5.2 General Examination

Figure 5.3 illustrates a snapshot of the plan knowledge that a patient's planning component typically uses to plan the diagnosis and treatment of health problems (autonomous treatment methods and co-operative treatment methods). The hierarchy of plan scripts represents the patient's knowledge of ways to treat health problems. When health problems develop, the patient's planning component selects the appropriate plan knowledge based on the description (symptoms) of the health problems that are present. The patient's problem-solving component will normally initiate this process by reporting the presence of health problems to the planning component. The planning component may have to interrogate its problem-solving component to obtain a sufficient description of problems in order to determine the appropriate plan knowledge. If the health problems are minor, then one of the special-purpose plans designed to treat minor health problems would be selected (e.g. colds, minor cuts, etc.). The plan information in these scripts (not shown) would describe self-treatment tasks such as bedrest

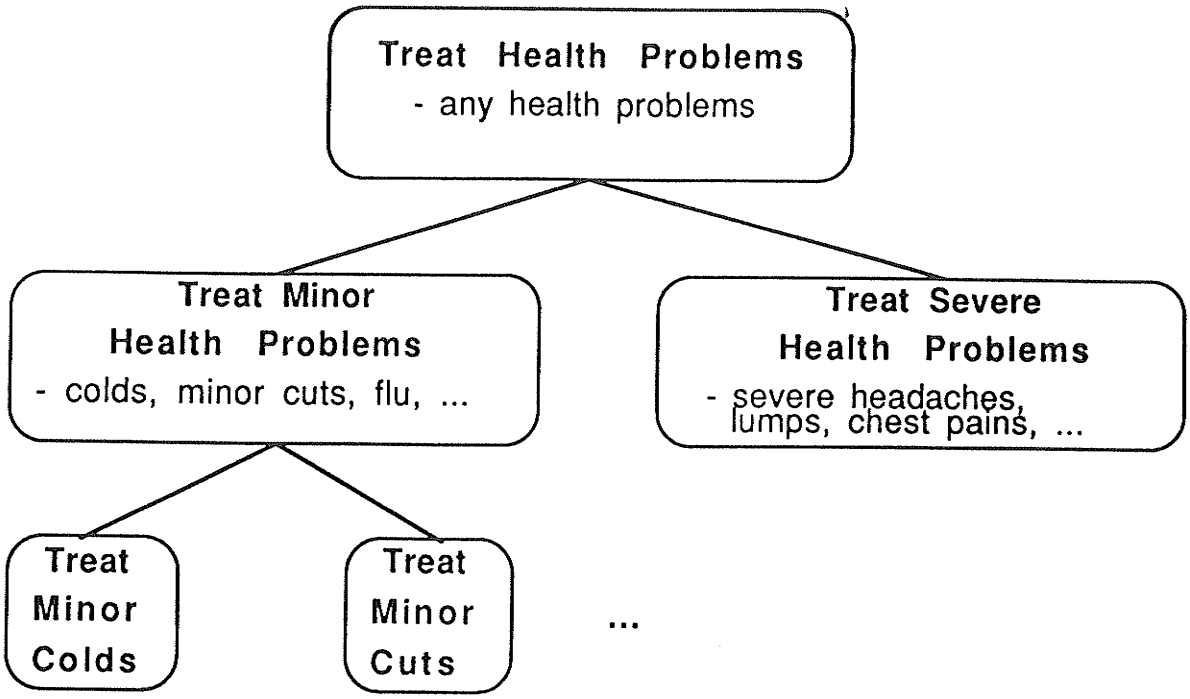


Figure 5.3: Snapshot of Patient's Plan Knowledge Hierarchy

Treat Severe Health Problems

Applicable-to:
- Operation: health care
- Object: (?a patient)
- Symptoms: constraints = ANY-OF (lumps, chest pains, severe headaches, ...)

Plan class: *treat health problems*

Plan: DO (obtain-professional-health-care (**symptoms**)
(apply-prescribed-care (**therapy**))

Figure 5.4: Snapshot of a Patient's Plan Script for Treating Severe Health Problems

and non-prescription drugs. On the other hand, if the patient is experiencing severe health problems then the plan script designed to treat severe problems would be selected.

Figure 5.4 illustrates a more detailed description of the plan information for treating severe health problems. The script is applicable if the symptoms that the patient is experiencing (obtained from the patient's problem-solving component) match those specified in the APPLICABLE-TO slot (e.g. lumps, severe headaches, chest pains, etc.). The plan knowledge in this example specifies one task - obtain treatment from a health-care professional. The symptoms stored in the problem description component of the script are used as arguments for the task (i.e. information to be passed on to the health-care professional). This plan knowledge illustrates that the patient's planning component is aware that the patient is not capable of treating severe health problems itself, but that treatment can be received from health-care professionals (other agents). Therefore, the planning component must arrange to obtain the necessary care from other agents.

Figures 5.5 and 5.6 illustrate the task and agent knowledge describing the classes of professional health-care that the patient is aware of and the agents who belong to each class. The task knowledge is used to determine the type of health care required and the class of agents that can provide the chosen care based on the health problems that the patient is experiencing. This task knowledge indicates that general care can be provided by a family physician or walk-in clinic, while emergency clinics are capable of providing emergency health care (e.g. treating heart attacks, broken bones, etc.). In addition, the

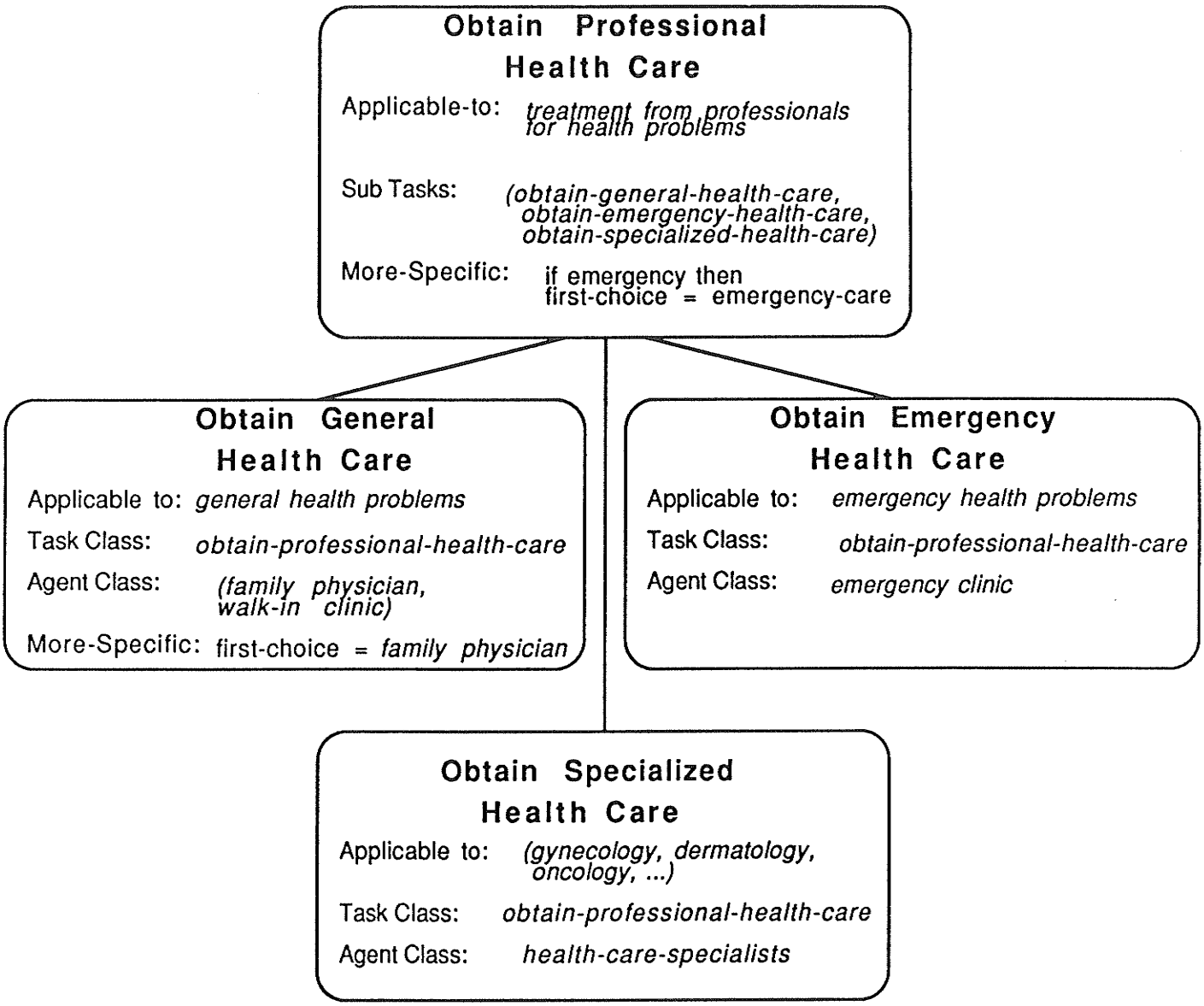


Figure 5.5: Snapshot of a Patient's Task Knowledge For Obtaining Professional Health Care

patient's planning component may have scripts describing how to obtain specialized health care (gynecology, oncology, dermatology, etc.). The

planning component normally acquires (builds) these scripts after earlier referals to specialists from general health-care professionals (i.e. acquire new knowledge through experience).

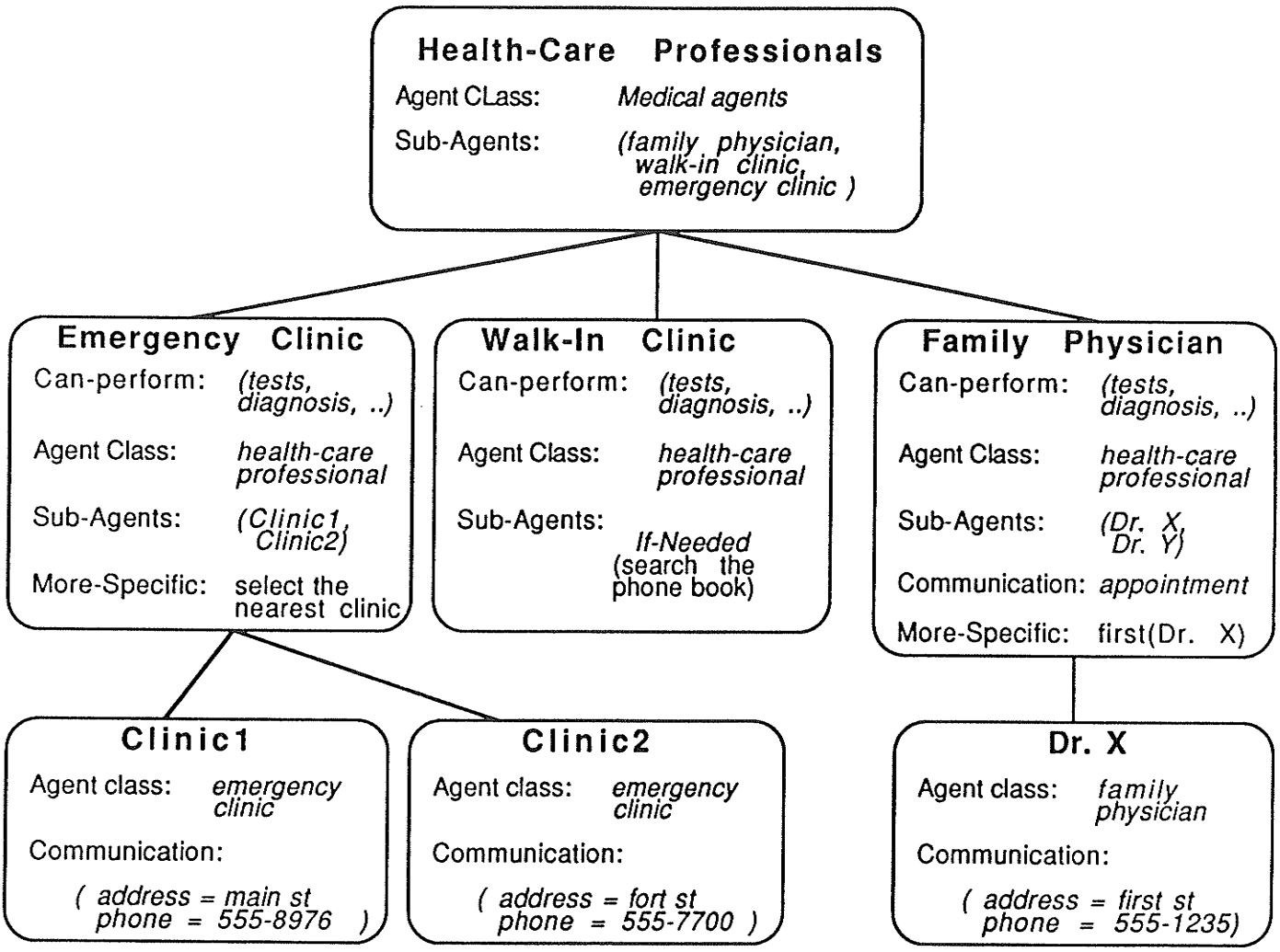


Figure 5.6: Snapshot of a Patient's Agent Knowledge Describing Health-Care Professionals

The agent knowledge hierarchy in Figure 5.6 is used to select an agent, or agency in the case of health-care clinics, belonging to the agent class specified by the chosen task script (the agent knowledge describing health-care specialist such as gynecologists, oncologists, and dermatologists is not shown). The subordinates of the specified agent class represent actual agents that can perform health-care tasks (e.g. Clinic1 and Clinic2 are emergencies clinics, and Dr. X is a family physician). The MORE-SPECIFIC slot specifies criteria that can be used to determine which of several applicable alternatives should be used. For example, the criteria defined in the general health care script specifies that the patient normally prefers to obtain general care from a family physician. The communication information specified in the agent scripts (e.g. address and telephone number) is used to plan and co-ordinate interactions with the selected agent to request health care. This may require the patient's planning component to negotiate with the health-care professional to set up an appointment for a clinical examination. Constraints such as the severity of the patient's problems and the physician's work load can be taken into consideration in order to schedule an acceptable appointment.

When the patient arrives for his first appointment, he will be interrogated by a receptionist (possibly a nurse) who must gather some basic socio-demographic information such as the patient's name, age, health care number, and previous physician where applicable. The receptionist's planning component employs plan and resource knowledge to co-ordinate this interrogation process (not shown). The patient is then examined by a physician. Figure 5.7 illustrates the plan knowledge that

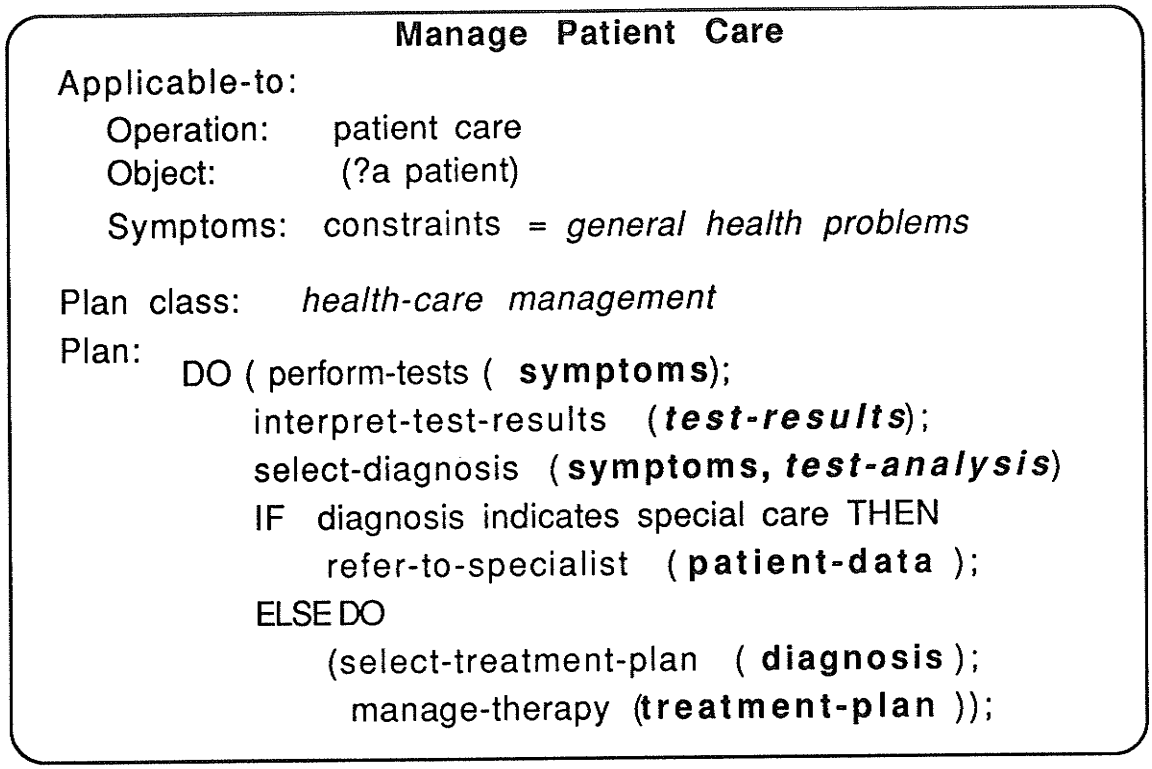


Figure 5.7: Snapshot of a Physician's Plan Script for Co-ordinating Health Care

a family physician's planning component uses to manage patient care (this is a plan script from the physician's plan knowledge hierarchy). This plan information specifies that a set of general tests should be performed and interpreted, followed by the selection of a diagnosis. At this point the plan stipulates a choice based on the selected or suspected diagnosis (represented in the plan by an IF-THEN-ELSE construct). If the diagnosis indicates the need for specialized health care (e.g. presence of cancer signs), then the plan specifies that the patient should be referred to an appropriate health-care specialist. On

the other hand, if the diagnosis is complete and represents a health problem that the physician is capable of treating, then a suitable treatment plan can be selected, followed by the task of managing the administration of the prescribed therapies.

The physician's planning component interprets this plan and uses task knowledge (Figure 5.8) to determine how each plan step can be carried out, either by the physician's problem-solving component (SELF) or by one or more other agents (nurses, surgeons, etc.). The planning component co-ordinates interactions with the appropriate agents in order to have the specified tasks carried out successfully. This may involve interrogating agents to obtain additional information about test results and interpretations, and negotiating with agents when conflicts or difficulties arise during task execution (scheduling tests, resolving misinterpretations of test results, etc.).

In addition, the agents who are asked to perform the tasks may interact with the physician's planning component and request additional information. Consider the task of selecting a diagnosis. The task knowledge in this case indicates that the physician's problem-solving component (SELF) can perform the diagnosis. While attempting to determine an appropriate diagnosis, the problem-solving component may request that additional tests be performed. The physician's planning component must co-ordinate such requests by planning further co-operative activities such as distributing tasks of performing the requested tests to appropriate agents and communicating the results to its problem-solving component. For example, while trying to select a diagnosis, the physician's problem-solving component may request a

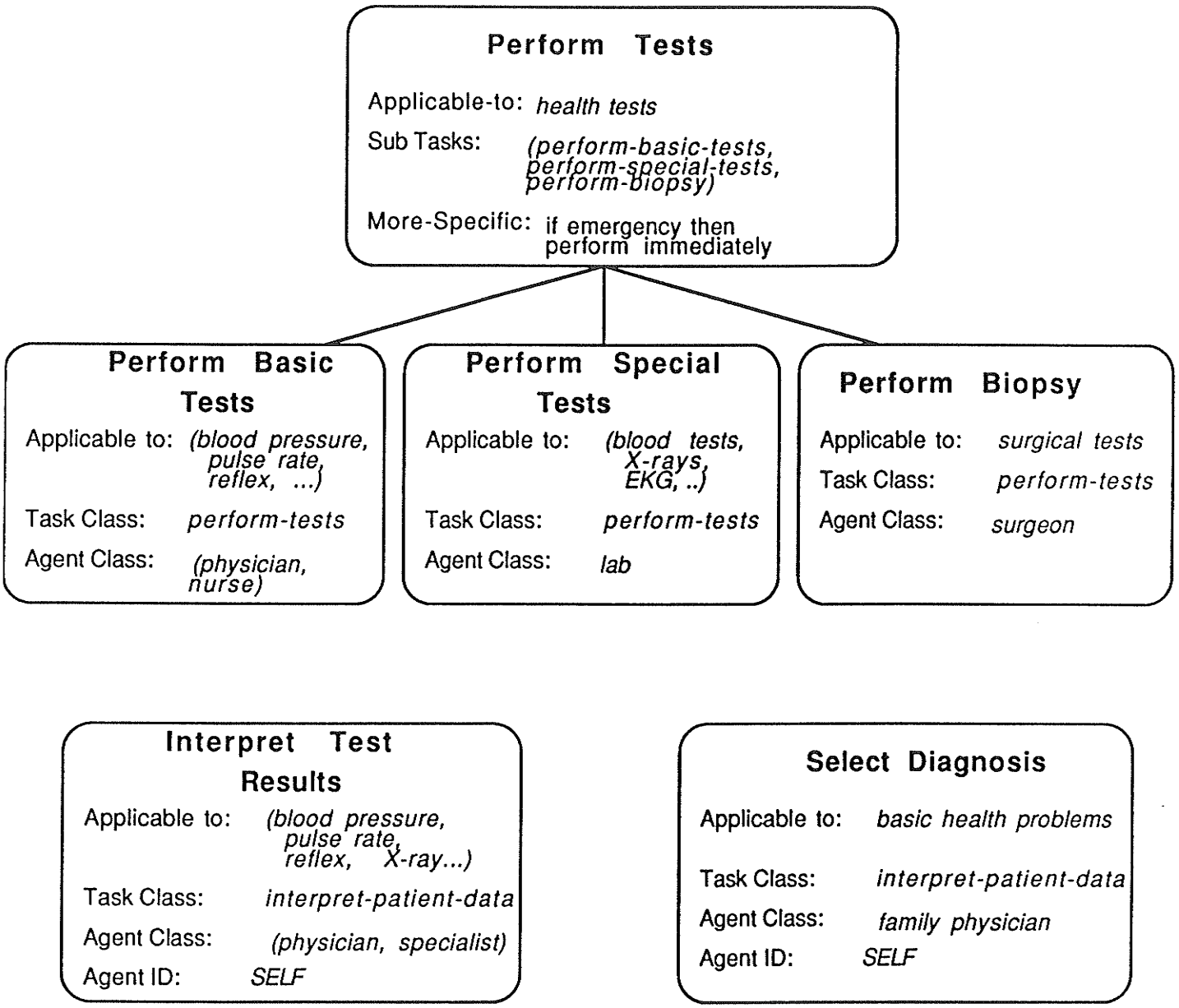


Figure 5.8: Snapshot of a Physician's Task Knowledge

biopsy be performed. Knowledge describing how to arrange for a biopsy to be carried out is maintained by the planning component which would

distribute this task to a surgeon; the planning component would co-ordinate scheduling the surgery with the surgeon and then route the results of the biopsy back to the problem-solving component once surgery was completed.

Once the task of selecting a diagnosis has been completed, the planning component must determine its next actions. If a complete diagnosis was generated, then the planning component must co-ordinate the selection of an appropriate treatment plan and the management of the prescribed therapies. Again, the physician's planning component would use task knowledge to determine how to perform these tasks. The task knowledge describing these tasks is not shown; however, these tasks normally are performed by the physician's problem-solving component.

If, however, the diagnosis indicates that specialized health-care is required, then the planning component must co-ordinate the referal of patient care to an appropriate health-care specialist. This step is necessary if the problem-solving component is unable to make a firm diagnosis or detects the presence of severe health problems that require specialized care (e.g. cancer problems, cardiac problems, etc.). The planning component must select task knowledge describing how to refer a patient to an appropriate health-care specialist based on the determined or suspected diagnosis (see Figure 5.9). For example, a spot on an X-ray of the patient's lung or a lump in the patient's breast would prompt the physician to refer the patient to a cancer specialist, while prolonged chest pains may necessitate a referal to a heart specialist. In some instances, the planning component may have to interrogate several different specialists to determine which one is best suited to the health problems that the patient is experiencing.

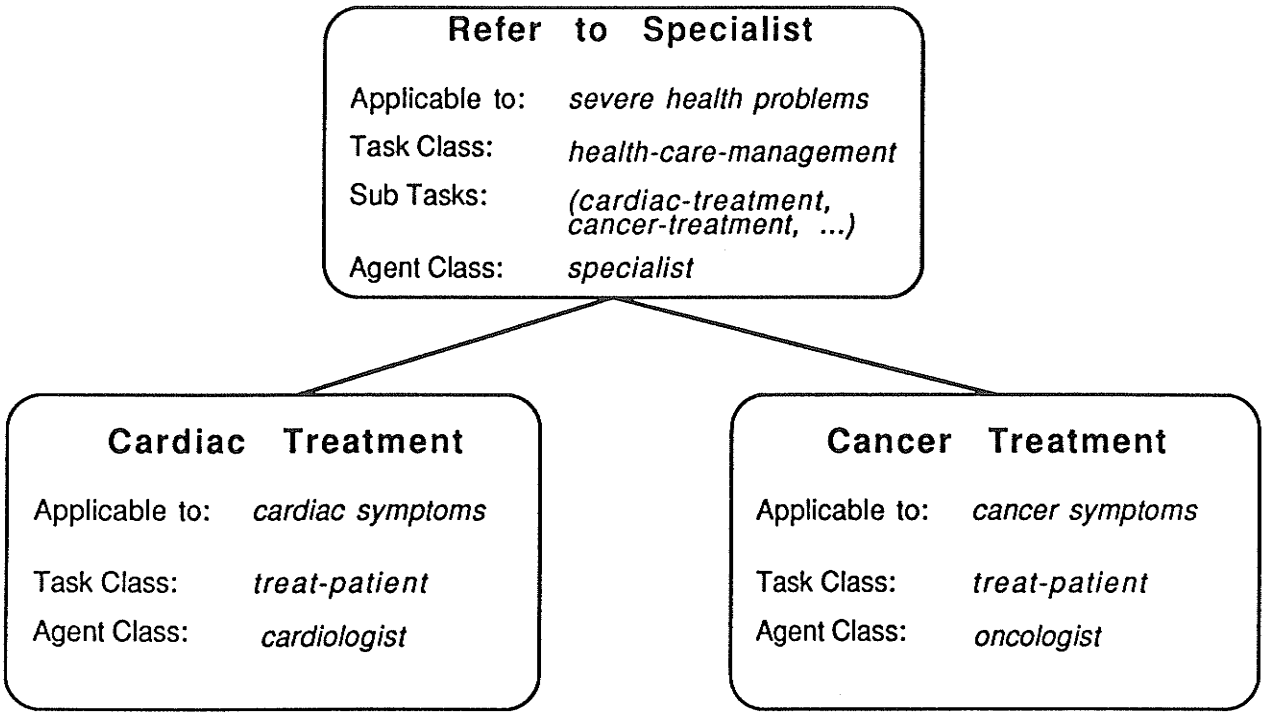


Figure 5.9: Snapshot of a Physician's Task Knowledge

The patient's health status can be used to determine a suitable appointment with the specialist. For example, the preliminary diagnosis may indicate that the patient is in a potentially dangerous state of health and requires the earliest possible treatment. In such cases, the specialist may be able to juggle his schedule to accommodate the patient immediately. The physician's planning component must interact with the specialist's planning component in order to plan and co-ordinate the referal and negotiate an acceptable appointment.

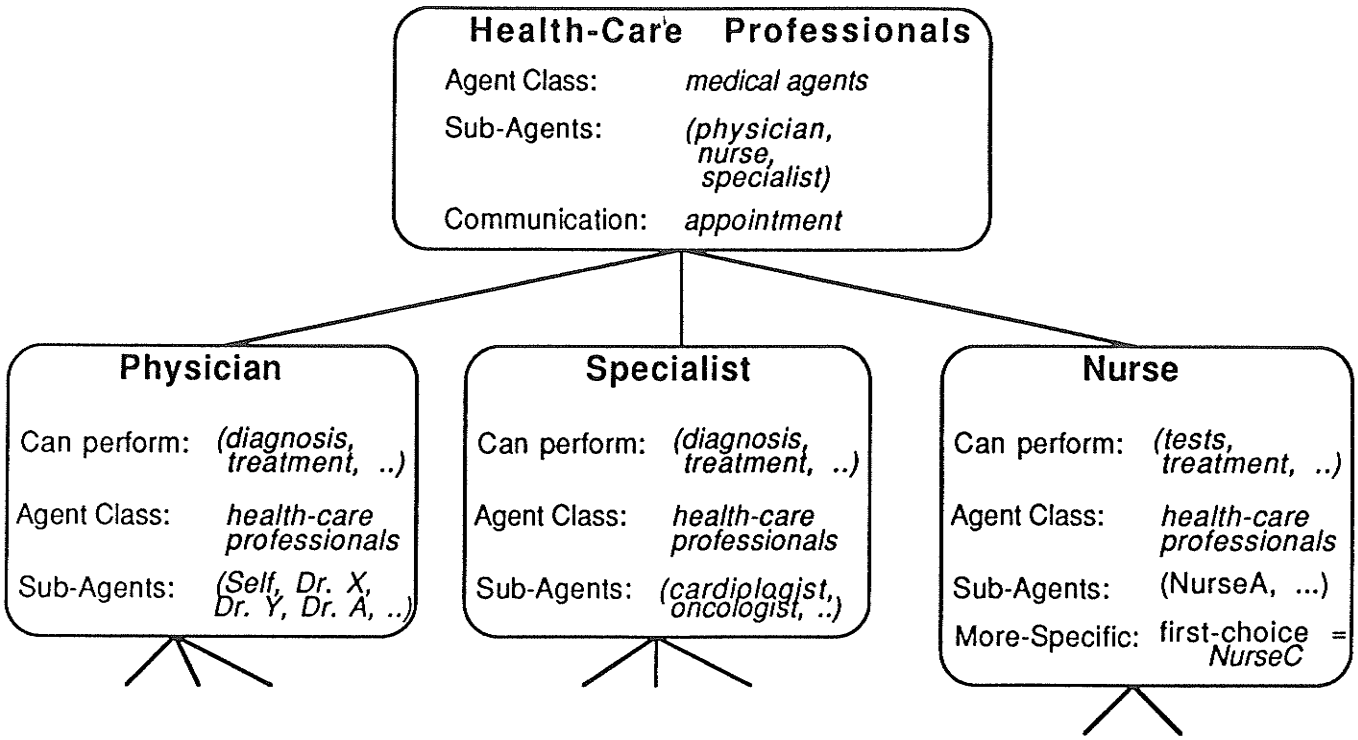


Figure 5.10: Snapshot of a Physician's Agent Knowledge

The agent information stored in the task knowledge scripts specifies classes of agents (AGENT CLASS) or specific agents (AGENT ID) that can execute each task (perform tests, interpret test, select diagnosis, etc.). Figure 5.10 represents the agent knowledge maintained by the physician's planning component describing the health-care professionals in the environment. The planning component uses this agent knowledge to determine specific agents belonging to specified agent classes and to determine how to interact with the agents (e.g. addresses, telephone numbers, etc.) to request that they perform tasks.

5.5.3 Cancer Diagnosis

An oncologist assumes responsibility for diagnosing and treating a patient after the patient is referred to a cancer centre. Figure 5.11 illustrates a snapshot of the plan knowledge that the oncologist's planning component uses to plan and co-ordinate patient care. The plan steps in this example correspond to those used by a family physician, the activities that are performed within each step will, however, reflect the oncologist's special expertise and abilities and those of the agents that the oncologist may co-operate with. In addition, the oncologist's problem-solving component will contain specialized knowledge and reasoning strategies that are designed to deal with cancer problems. Thus, the structure of an oncologist's plan and resource knowledge is similar to that of a family physician's; it is the activities and resources described by the knowledge that differ, thus representing the different skills and abilities of each.

Consider the oncologist's task of selecting a diagnosis. The plan knowledge in Figure 5.11 describes the tasks that an oncologist's planning component must co-ordinate to provide cancer care for a patient. The first part of this plan specifies that a set of standard tests should be performed and interpreted, followed by the selection of a diagnosis. The planning component must plan and co-ordinate the execution of these tasks using the appropriate task and agent knowledge (see Figures 5.12 and 5.13). The planning component must arrange for the necessary tests to be performed by qualified agents (oncology nurses, surgeons, technicians, etc.) and to have the test results interpreted, again by qualified agents (the oncologist's problem-solving component, pathologists, etc.).

Manage Care For Cancer Patients

Applicable-to:
  Operation: cancer care
  Object: (?a patient)
  Symptoms: constraints = *cancer problems*

Plan class: *health-care management*
Plan:
  DO (perform-tests (**symptoms**);
    interpret-test-results (***test-results***);
    select-diagnosis (**symptoms, *test-analysis***)
    IF diagnosis indicates special care THEN
      select-diagnosis (*specialist* **patient-data**)
    select-treatment-plan (**diagnosis**);
    manage-therapy (**treatment-plan**));

Figure 5.11: Snapshot of an Oncologist's Plan Script for Co-ordinating Cancer Care

Planning and co-ordinating the execution of these tasks may require the planning component to interrogate and negotiate with other agents in order to overcome constraints, ambiguities in interpretations, and any other difficulties that may arise. In addition, the agents who are asked to perform the various tasks may interrogate the oncologist's planning component to request that additional tasks be performed or that additional information be forwarded (e.g. perform additional tests,

forward additional patient history, etc.). The oncologist's planning component must plan and co-ordinate co-operative activities that carry out such requests (e.g. schedule additional tests, retrieve patient history, etc.).

The task of selecting a diagnosis is normally carried out by the oncologist's problem-solving component (SELF) as indicated in the task knowledge in Figure 5.12. However, in some cases the oncologist may not be able to make a firm diagnosis, in which case the planning component must plan and co-ordinate interactions with one or more other oncologists in order to construct a complete diagnosis co-operatively. The planning component uses the agent knowledge in Figure 5.13 to determine the oncologists who should be consulted based on the partial diagnosis provided by its problem-solving component. The partial diagnosis will normally specify the type of cancer (breast, lung, bone, etc.) that is present. The planning component uses this information to select an oncologist who specializes in the care of the specified or suspected cancer. This agent knowledge represents the oncologist's partial knowledge of the abilities of some of the other agents in the centre (and possibly agents in other cancer centres or cancer institutes) and enables the planning component to plan and co-ordinate interactions with other oncologists in order to utilize their expertise when dealing with problems that stretch past the boundaries of the oncologist's own expertise.

This consultation process is a good example of multiple agents (two or more oncologists) collaborating to attain a solution (complete a diagnosis). The oncologists' planning components must plan and

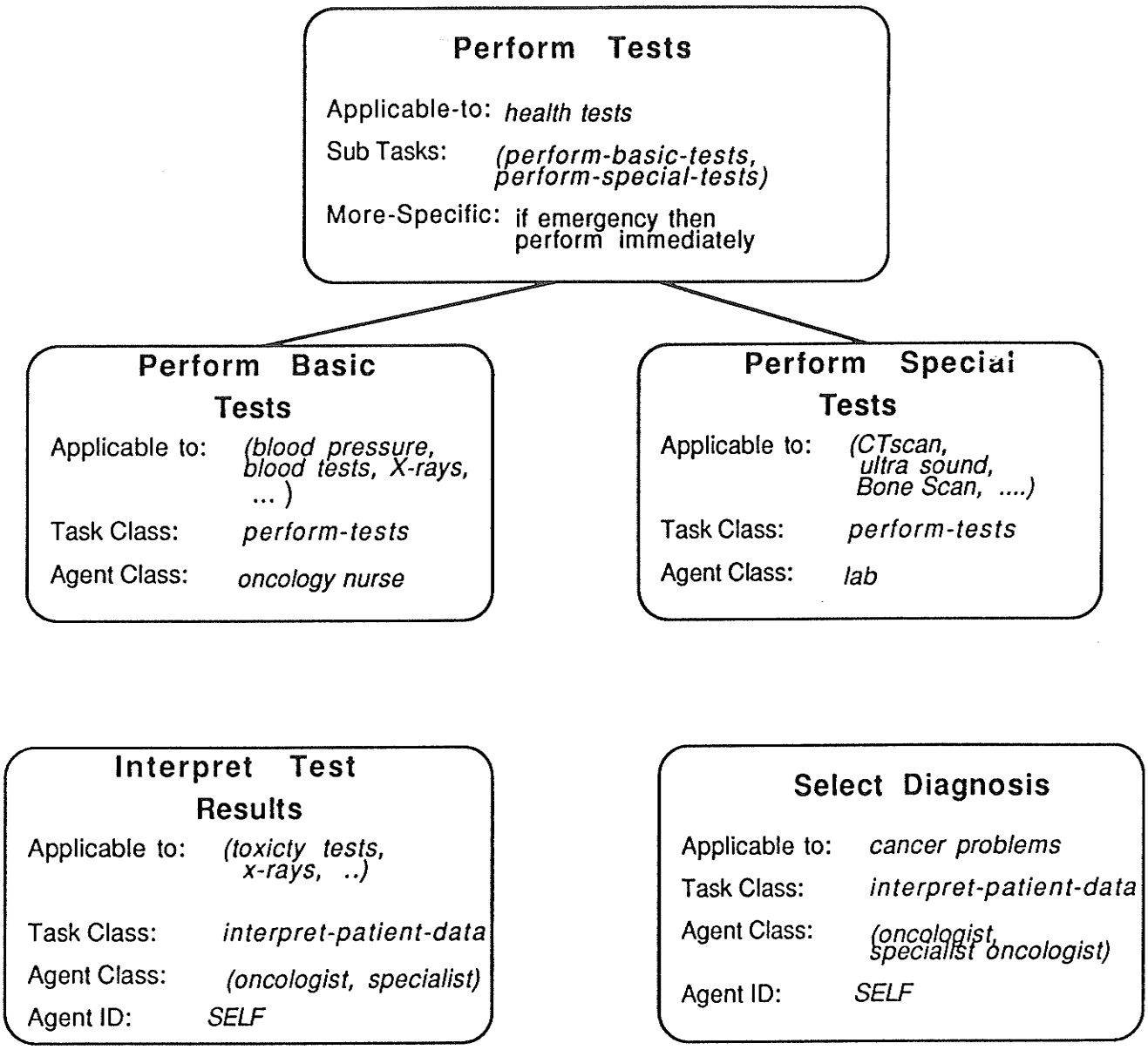


Figure 5.12: Snapshot of an Oncologist's Task Knowledge

co-ordinate interactions among one another in order to exchange information describing the problem context (patient's status, symptoms, potential diagnoses) and negotiate an acceptable diagnosis based on this

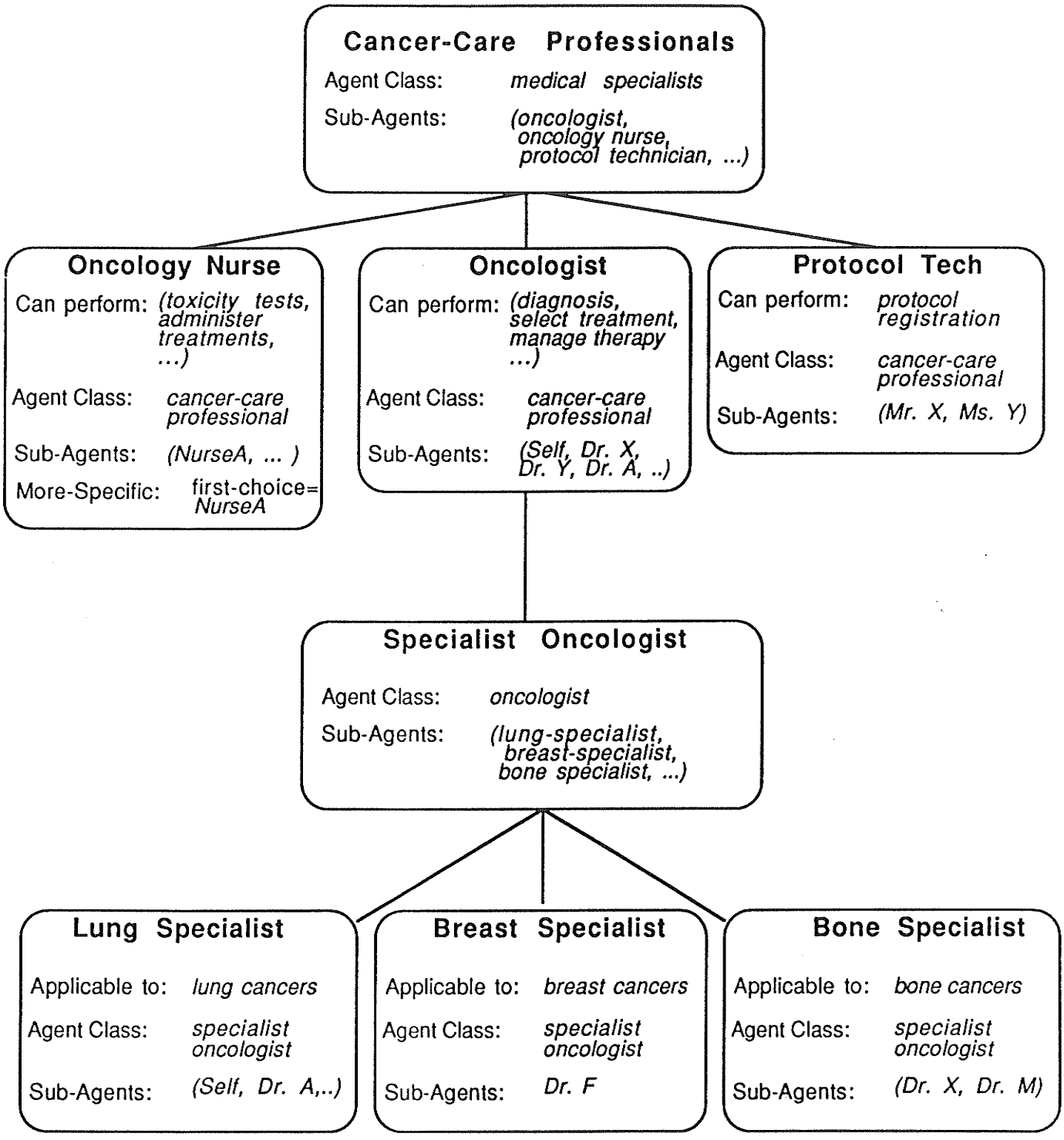


Figure 5.13: Snapshot of an Oncologist's Agent Knowledge

information and their expertise and experience (contained in their respective problem-solving components). The oncologists must co-operate to integrate their preliminary diagnoses into a final diagnosis, a process which may require several interactions and refinements to intermediate diagnoses. The negotiation and integration processes result in a synergy of the expertise of several oncologists which is often essential in solving the complex task of diagnosing cancer problems. The planning component of each oncologist acts a liaison between its corresponding problem-solving component and the problem-solving components of the other oncologists involved in the consultation in order to co-ordinate co-operative diagnosis activities effectively.

5.5.4 Establishing a Treatment Plan

The next step in the patient management plan (Figure 5.11) is to select an appropriate treatment plan based on the diagnosis. As illustrated in the snapshot of the plan knowledge hierarchy in Figure 5.14, selecting a treatment plan can be carried out in one of two ways: the patient can be placed in a standard treatment plan (and may or may not participate in a clinical trial), or the oncologist can select a suitable non-standard plan to treat the patient. Standard plans are normally used to treat patients; however, there may be some characteristics of the patient (age, health history, or progression of the disease) that dictate the need for a non-standard plan (i.e. a treatment plan that can treat a patient's special needs effectively).

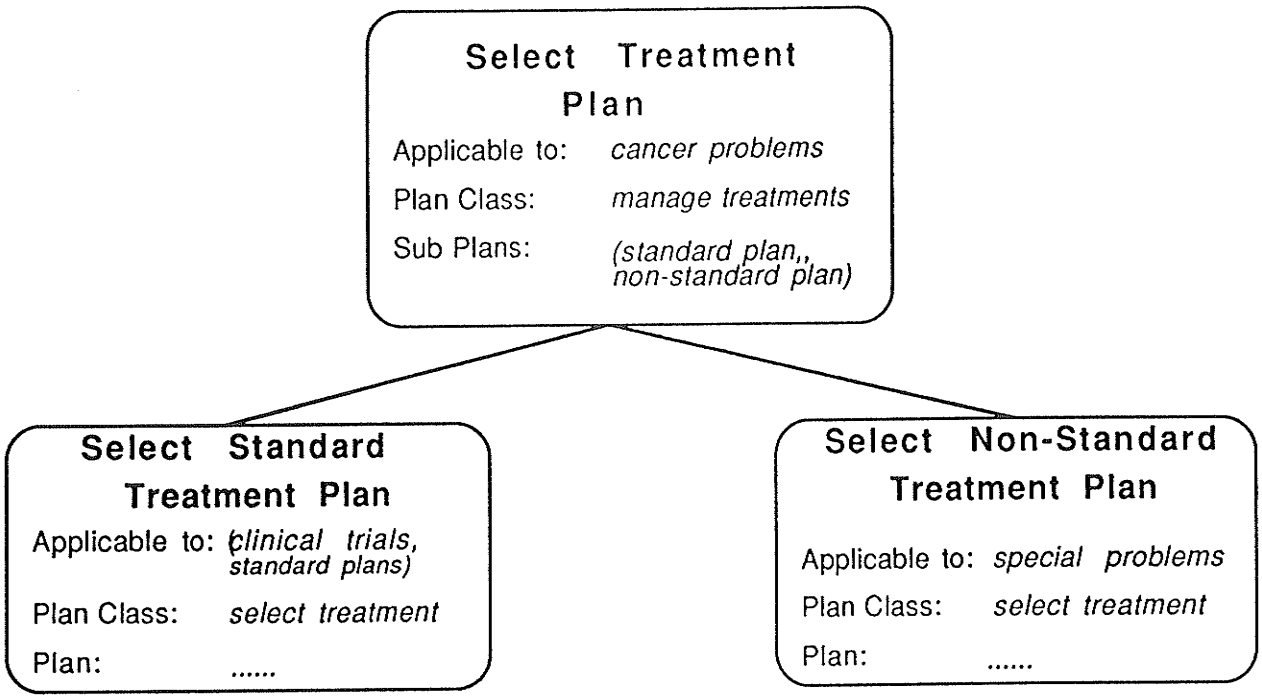


Figure 5.14: Snapshot of an Oncologist's Plan Knowledge Hierarchy

Select Standard Plan

Applicable-to:
Operation: select treatment plan
Object: (?a patient)
Symptoms: constraints = *standard cancer treatment*

Plan class: *select-treatment-plan*

Plan:
DO (select-treatment-plan (***diagnosis***);
IF undetermined-plan Then
select-protocol (*specialist,* ***diagnosis***);
Fail then abort this plan
IF clinical trial then
register-patient (***protocol***))

Figure 5.15: Snapshot of an Oncologist's Plan Script for Selecting a Standard Treatment Plan

The oncologist's planning component uses the diagnosis information and interrogations of the agents who participated in the diagnosis task in order to determine which category of treatment plan is appropriate (standard or non-standard treatment). In order to select a treatment plan, the oncologist's planning component must interpret and apply the chosen plan knowledge. (Note that the criteria in the MORE-SPECIFIC slot of the general treatment selection plan script indicates that the oncologist should attempt to establish a standard treatment plan before using a non-standard treatment plan).

Figure 5.15 illustrates an expanded version of the plan script for placing a patient in a standard treatment plan. First, the planning component must co-ordinate the selection of an appropriate protocol. Task knowledge in Figure 5.16 corresponding to this step indicates that the oncologist's problem-solving component (SELF) can perform this task (or another oncologist can perform the task). The oncologist selects a protocol based on the diagnosis and various eligibility requirements stipulated by the protocols such as patient age, health history, and diagnosis (type and extent of cancer to be treated). The second plan step specifies that the planning component can plan and co-ordinate consultations with other oncologists to determine an appropriate protocol. This is an optional step which would only be used when the oncologist's problem-solving component indicates that it requires help in selecting a treatment plan (lack of confidence or experience, incomplete information, etc.). Agent knowledge in Figure 5.13 would again be used to determine an appropriate specialist for the type of cancer to be treated. If a standard treatment plan is not applicable,

then the planning component can apply the plan and resource knowledge for selecting a non-standard plan (not shown) in order to co-ordinate the activities required to select an appropriate non-standard treatment plan.

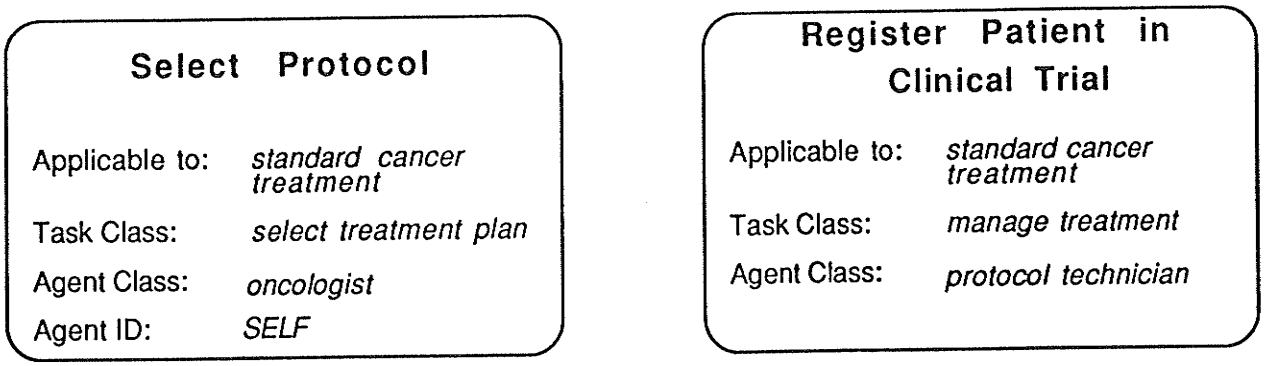


Figure 5.16: Snapshot of an Oncologist's Task Knowledge

If a protocol is selected, then the oncologist arranges to have the patient registered in the clinical trial of the cancer institute that sponsors the protocol (third step of the plan script in Figure 5.15). The task knowledge in Figure 5.16 specifies that this task can be distributed to the protocol technician at the cancer centre. The protocol technician's planning component uses its own plan and resource knowledge (not shown) to plan and co-ordinate the execution of this task. During the execution of this task, the protocol technician can interrogate the oncologist and possibly the patient through their respective planning components in order to obtain necessary patient information (e.g. age, eligibility information, protocol, patient id number). Next, the protocol technician must plan and co-ordinate

interactions with a representative of the cancer institute that sponsors the protocol in order to register the patient in the appropriate clinical trial. The protocol technician will relay the results of the registration process to the patient's oncologist. If the sponsoring institute rejects the patient, then a treatment plan from the protocol may still be used, but the patient won't be part of the clinical trial.

5.5.5 Administering Treatment

Figure 5.17 illustrates a snapshot of the plan knowledge used by the oncologist to co-ordinate the tasks involved in managing the administration of cancer therapies. The plan information specifies that the oncologist can co-ordinate the administration of three types of therapy: surgical therapy, chemotherapy, and radiotherapy (a treatment plan may specify all or some of these therapies). The oncologist selects the appropriate plan knowledge script corresponding to the type of therapy to be administered. The plan script in Figure 5.18 specifies that managing the administration of therapies consists of four tasks: performing toxicity tests, interpreting test results, determining treatment plan modifications, and administering prescribed treatments (taking any modifications into consideration). The planning component must select and apply the task knowledge in Figures 5.19 and 5.20 to determine how each task in the plan can be carried out. The task knowledge in this example indicates that an oncology nurse can perform the toxicity tests. The nurse would have to co-ordinate sending to the lab any test samples that have to be analyzed. The results would then be integrated and returned to oncologist (by updating the patient's chart) for interpretation (the task and agent knowledge that the nurse would use to co-ordinate interactions with a lab is not shown).

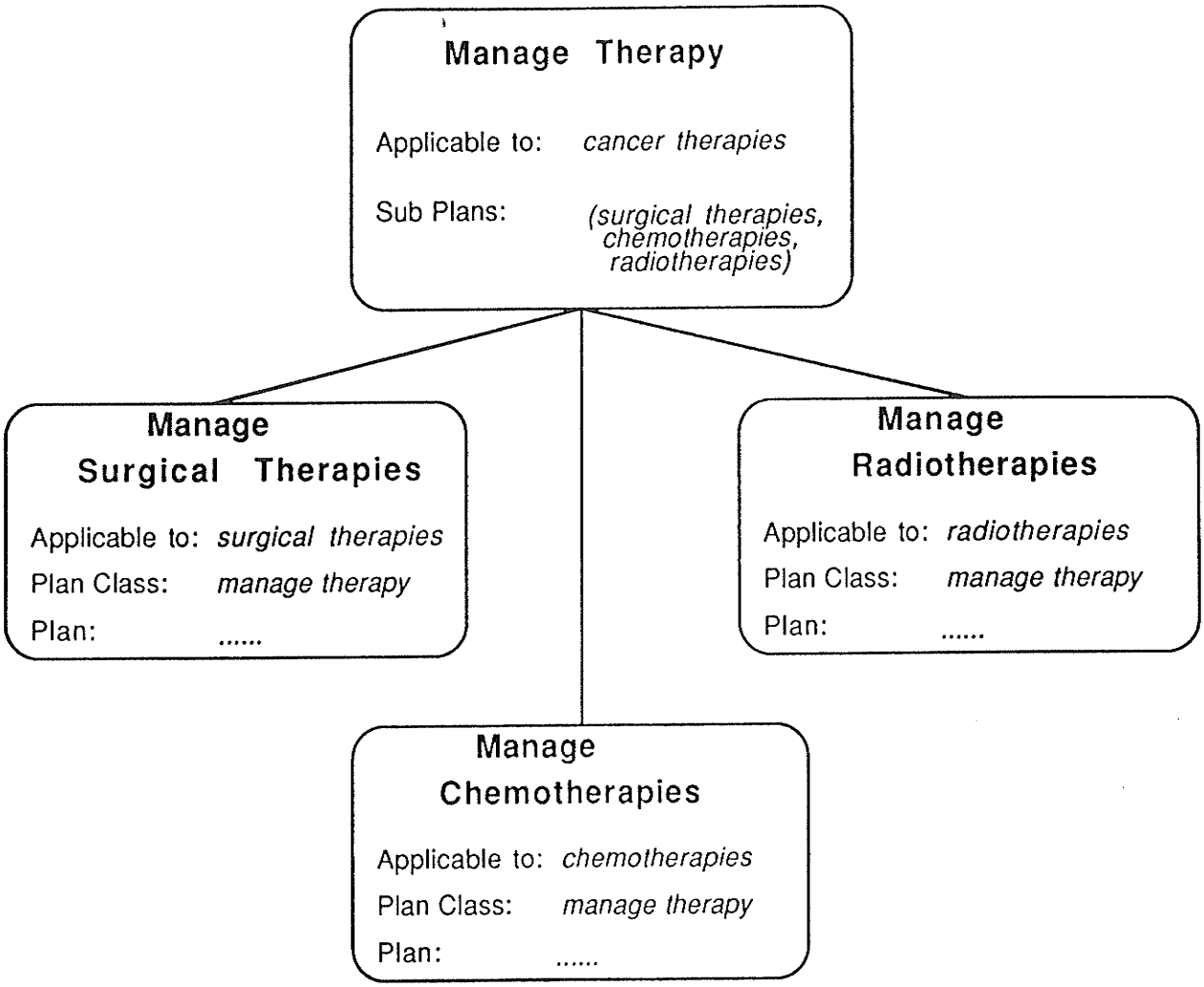


Figure 5.17: Snapshot of an Oncologist's Plan Knowledge Hierarchy for Administering Therapies

The oncologist will review the patient's chart and interpret the tests in order to determine whether modifications to the treatment plan are necessary. Task knowledge in Figure 5.19 indicates that the oncologist is capable of determining plan modifications when toxicity problems arise. The oncologist's planning component would co-ordinate

Manage Chemotherapies

Applicable-to:
Operation: administer therapy
Object: (?a patient)
Symptoms: (?a symptom-list)
Treatment Plan: (?a treatment-plan)
Therapy Type: constraints = chemotherapies
Plan class: *health-care management*
Plan:
DO (perform-toxicity-tests (***toxicity-tests***);
interpret-test-results (***test-results***);
modify-treatment (***test-results, treatment-plan***);
administer-treatment (***treatment-plan, modifications***))

Figure 5.18: Snapshot of an Oncologist's Plan Script for Co-ordinating Chemotherapy

interrogations of its problem-solving component or other oncologists in order to determine the necessary modifications. The task knowledge in Figure 5.19 also indicates that minor toxicity problems may be handled by the oncology nurse if necessary (e.g. the oncologist is very busy), while more severe problems may require consultations with experts (i.e. co-operative problem solving).

After determining any applicable plan modifications, the oncologist arranges to have treatments administered, reducing dosages and omitting or delaying doses when necessary. The oncologist's planning component again uses task knowledge to determine that chemotherapy treatments can be administered by an oncology nurse (see Figure 5.20). The oncology

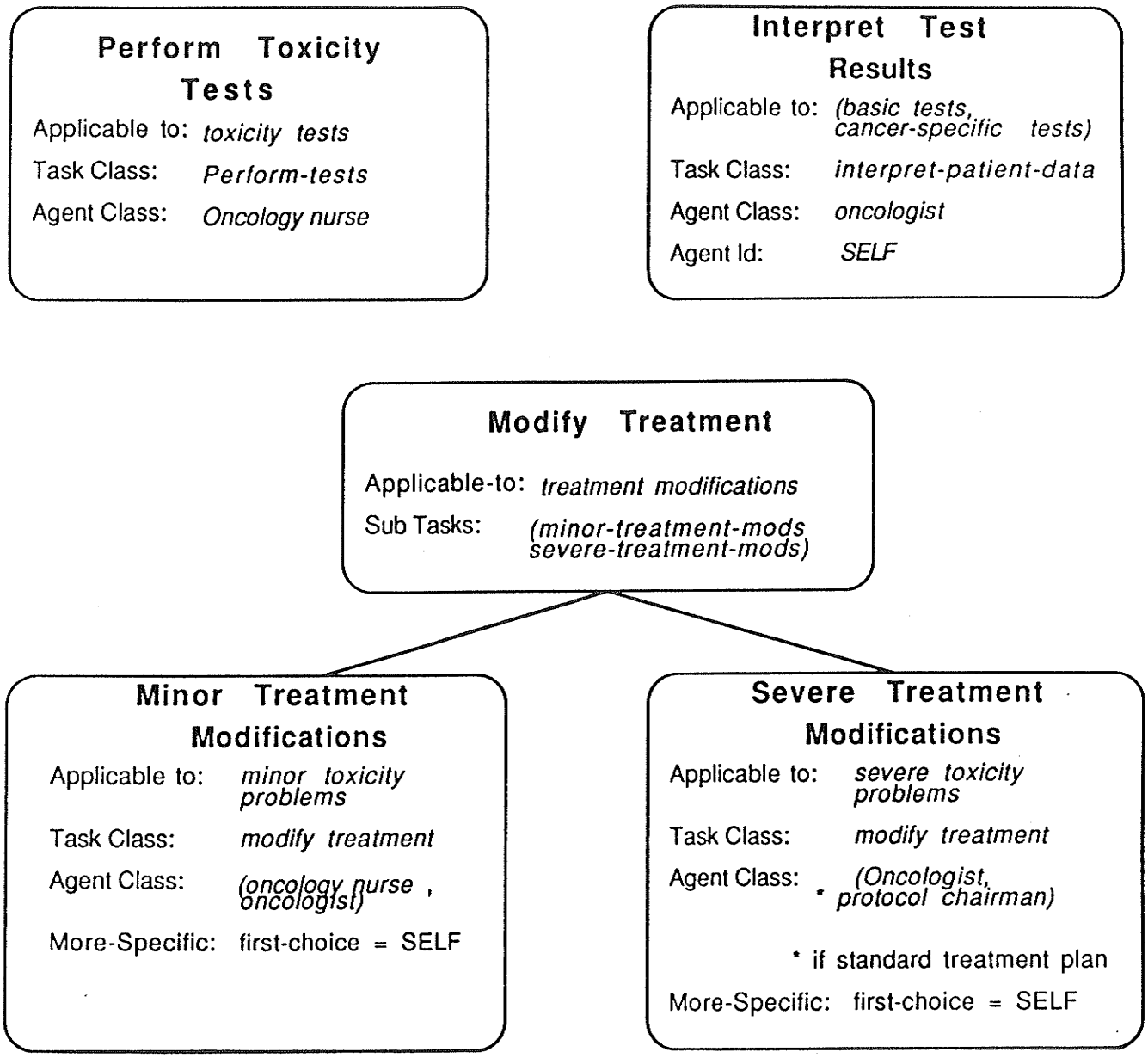


Figure 5.19: Snapshot of an Oncologist's Task Knowledge

nurse's planning component will contain plan knowledge for co-ordinating the tasks involved in administering chemotherapy treatments. In addition, the nurse's planning component will possess a task knowledge script indicating that another oncology nurse is capable of

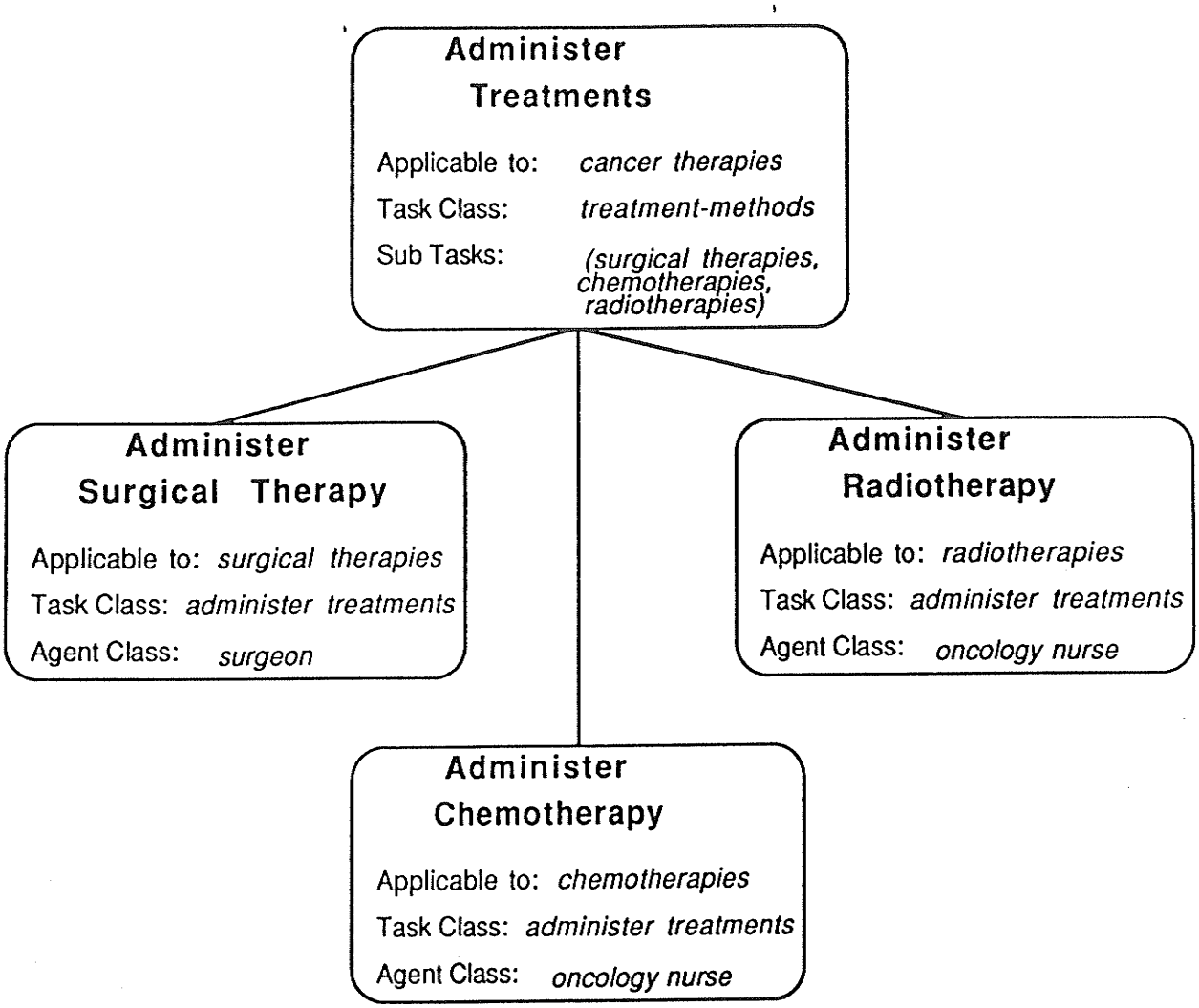


Figure 5.20: Snapshot of an Oncologist's Task Knowledge

administering chemotherapy treatments (see Figure 5.21). This enables the oncology nurse to represent explicitly that any qualified oncology nurse can administer chemotherapy treatments (i.e. partial knowledge of the general capabilities of the agent class to which nurse belongs). As a result, if the nurse is unable to administer a patient's therapy because of illness or schedule conflicts, then the nurse can negotiate

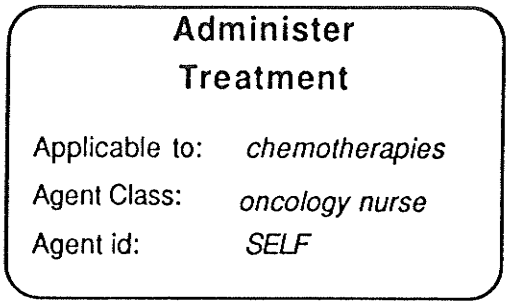


Figure 5.21: Snapshot of an Oncology Nurse's Task Knowledge

with other oncology nurses to have one or more of them take over all or some of the treatment tasks. (For example, an oncology nurse who is pregnant could perform the initial consultation, but would not administer the chemotherapy). The nurse would have to forward the treatment plan, plan modifications, and any other information that the other nurse(s) need to administer treatment appropriately. A nursing supervisor would co-ordinate the rescheduling of the nurses' tasks to ensure that the overall schedule remains balanced. Thus, the nurse typically must either negotiate directly with the nursing supervior who would adjust the schedule and inform the other nurses affected, or with another nurse as described above and then obtain verification from the nursing supervisor who would update the schedule accordingly.

After administering treatments for the current treatment period, the nurse will update the patient's chart and schedule the next treatment visit, interacting with other agents as required (i.e. patient, oncologist, etc.). If problems were detected during the toxicity tests, then the nurse may arrange special visits at which time toxicity levels

will be checked and delayed therapies administered. The nurse can negotiate with the patient to determine an acceptable time and date for special visits within the boundaries of the constraints prescribed by the treatment plan.

5.6 CONTROL OF DISTRIBUTED PROBLEM SOLVING

5.6.1 Overview

The previous section examined how the model of distributed problem solving presented in Chapter 4 can be applied to create artificial agents capable of performing the co-operative problem-solving activities necessary in diagnosing and treating cancer problems. The focus of this discussion was the plan, task, and agent knowledge that the planning component of each agent used to plan and co-ordinate problem-solving activities, many of which require co-operation among several agents. This knowledge enables the planning components to reason about the problem-solving tasks that must be performed to solve given problems and to determine how the tasks can be distributed to agents who are capable of performing them (tasks can be carried out by an agent itself or in co-operation with other agents).

The control strategy employed by the planning components uses several control functions to co-ordinate co-operative problem solving: introspection, interrogation, negotiation, integration, and plan execution. The remainder of this section discusses each of these control functions and illustrates how they are used to plan and co-ordinate some of the co-operative problem-solving activities discussed in the previous section.

5.6.2 Introspection

An agent's planning component performs introspection to select plan, task, and agent knowledge. Given a problem-solving request, a planning component performs introspection to determine the plan knowledge specifying the tasks required to fulfill the request. This selection process is based on various aspects of the given problem-solving request and the preconditions specified in the APPLICABLE-TO slot of the knowledge scripts; this includes descriptions of the problems to which each script (plan, task, agent) is applicable as well as control information for determining when a script is, in fact, applicable and ways to reformulate initially inapplicable problem descriptions so that plans, tasks, and agents become applicable. For example, when a patient develops health problems, its planning component performs introspection to select plan knowledge describing various ways of treating the given problems. In this case, introspection is based on the symptoms that the patient is experiencing. These symptoms are used to select one or more plan scripts that are applicable to solving (treating) the patient's health problems (e.g. self-treatment methods or obtaining professional health care).

A planning component also performs introspection to select task and agent knowledge. Selecting task knowledge is based on the task descriptions specified by the plan knowledge (type of task, task attributes, etc.). For example, if a patient is attempting to obtain professional health care, then its planning component must perform introspection to determine the type of care that is required based on the health problems (symptoms) that the patient is experiencing (i.e.

general care, emergency care, specialized care). Thus, a search of the task knowledge describing professional health care is performed to determine the type of care that is required.

Once the appropriate task knowledge has been selected, a planning component performs introspection to select the agent knowledge describing one or more agents that can perform the specified tasks. In the example above, the patient's planning component would perform introspection to determine the agents that can provide the chosen category of professional health care (e.g. family physician, emergency clinic, specialists, etc.). This information is used to co-ordinate interactions with one or more agents to request that the necessary tasks be carried out (i.e. obtain treatment from an appropriate health-care professional).

5.6.3 Interrogation

The interrogation operation is used by a planning component to make inquires of agents in order to obtain information relevant to co-ordinating problem-solving activities. A planning component can also interrogate its problem-solving component, in which case it is actually performing a form of introspection: it is requesting additional information about its own problem-solving capabilities or needs. For example, a patient's planning component may interrogate its problem-solving component to obtain additional information about the health problems that it is experiencing in order to determine the type of health care that is required (i.e. select the appropriate plan knowledge for treating the health problems). On the other hand, a

planning component may have to interrogate other agents. An example of this type of interrogation is an oncologist's planning component interrogating the planning component of a family physician in order to obtain additional information about a patient's health history. A planning component can also perform interrogations to determine whether one or more agents can or will perform a given task. This form of interrogation is used to request that agents perform tasks specified by plan and task knowledge (i.e. send one or more agents a problem-solving request and have the agents indicate whether or not they can carry out the request).

Thus, a planning component performs interrogation to interact with agents, either its own problem-solving component or other agents in the environment, to obtain additional information and to distribute tasks. The information represented in the agent knowledge is used to determine how to interact with a given agent. For example, if a patient determines that he needs to visit an emergency clinic, then the agent knowledge corresponding to this type of health-care agency is used to determine the address of an appropriate clinic.

5.6.4 Negotiation

Planning components perform negotiation to co-ordinate co-operative interrogations in order to resolve conflicts and clear up discrepancies and misunderstandings surrounding distributed problem-solving activities. The planning components of agents engaged in or attempting to engage in co-operative problem solving must negotiate with one another to overcome constraints and differences in opinions that prevent

tasks from being completed successfully. For example, if an oncologist determines that a patient requires immediate surgery to alleviate a life-threatening problem (e.g. brain tumor), its planning component must co-ordinate negotiations with one or more surgeons to have the surgery scheduled as soon as possible. The oncologist's planning component must co-ordinate interrogations of the surgeons (via their planning components) to determine the available times and dates that surgery can be performed. It must then determine if the times and dates are acceptable, possibly by interrogating its problem-solving component. If the oncologist is not satisfied with the scheduling, then the planning component will co-ordinate further interactions with the surgeons, passing on any information that is important to the decision, until a mutually acceptable time and date is agreed upon. For example, the oncologist may inform the surgeon that surgery must be performed within two weeks, otherwise the patient's health will be compromised. The surgeon can then use this information to determine if he can rearrange his schedule to accommodate the patient, or recommend another surgeon who may be able to perform the surgery within the requested time period. This process may require several iterations and the planning components of the agents must co-ordinate the exchange of information (justifications and explanations) among the respective problem-solving components.

In some cases a planning component may have sufficient knowledge of its problem-solving component's capabilities or of its work load to make negotiation decisions. For example, a surgeon's planning component can use the surgeon's schedule (i.e. surgeries) to determine if a requested

date of surgery can be accommodated. If the surgeon's schedule is full, then the surgeon's planning component may have to interrogate its problem-solving component to determine if the schedule can be rearranged to accommodate the oncologist's request. Ideally, the planning components will possess sufficient knowledge to make effective negotiation decisions (however, in practice, the planning component may have to interrogate its problem-solving components to evaluate what-if situations). Much of this knowledge will be gathered over time as the planning components obtain knowledge (constraints, ways to relax constraints, etc.) through experience (knowledge describing negotiable aspects of an agent's own problem-solving abilities and those of other agents is represented in the appropriate task and agent knowledge scripts).

5.6.5 Integration

A planning component performs integration to take knowledge generated by agents and combine it with existing knowledge. This enables the results of executed tasks to be integrated and used in the execution of subsequent tasks or to produce solutions to problems. For example, a physician's planning component must integrate the test results it receives before interpretation and diagnosis can be performed. In other words, as agents that perform the required tests respond with results, the planning component must integrate the results so that they may be examined in subsequent steps (i.e. interpretation and diagnosis).

In some instances, a planning component may not possess sufficient knowledge to enable it to perform the integration functions itself.

Instead, it must distribute the task of integrating the information to its problem-solving component or another agent. For example, an oncologist's planning component will manage the integration of the diagnoses received during a co-operative diagnosis session with other oncologists. The actual integration functions are performed by the oncologist's problem-solving component which possesses knowledge enabling it to summarize, merge, or augment the various diagnoses appropriately.

5.6.6 Plan Execution

A planning component performs plan execution to carry out problem-solving plans specified by plan knowledge. Executing plans involves planning and co-ordinating several other operations including: introspections, interrogations, negotiations, and integrations. Introspections are used to refine composite plan steps using additional plan knowledge, and to determine how and by whom each specified task can be performed. Interrogations are used to ask agents whether they will perform specified tasks and to request additional information about problem-solving requests that the planning component has received. Negotiations may be necessary to resolve conflicts or reformulate task descriptions so that required activities can be performed, and integrations may be needed to combine results received from one or more agents after the completion of intermediate tasks.

Consider the plan knowledge used by an oncologist to manage the administration of chemotherapies (Figure 5.18). This plan specifies four tasks: perform toxicity tests, interpret toxicity tests, modify

treatments, and administer treatments. Each of these plan steps represents a primitive task and therefore does not require refinement using other plan knowledge (recall that a primitive task may be decomposed into several other tasks by the agent to which it is distributed). The plan is executed by performing introspection to select appropriate task and agent knowledge describing how and by whom each specified task can be performed. This resource knowledge is interpreted and applied in order to perform interrogations of agents who can execute each task (e.g. SELF, other oncologists, oncology nurses, etc.). These interrogations request that the agents carry out the specified tasks.

Responses from these interrogations may necessitate negotiations with one or more of the agents in order to resolve conflicts (e.g. time conflicts, missing information, misinterpretations) or reformulate task descriptions co-operatively. The execution of this plan requires intermediate integrations of the results of planned tasks. The results of the toxicity tests must be integrated and passed on to the agents (possibly several) who will perform the interpretations. The interpretations must then be integrated and used to determine any necessary treatment modifications.

In addition, plan execution may involve reasoning about alternative plans or alternative ways of executing planned tasks. For example, an oncologist has plan knowledge describing two ways to select a treatment plan, either by using a plan that describes how to select a standard treatment plan or by using a plan that specifies how to select a special-purpose treatment plan. Using the criteria specified in the

MORE-SPECIFIC slot of the plan scripts, the plan knowledge indicates that the primary method of selecting a treatment plan is to choose an appropriate standard treatment plan, but if standard treatment plans are not applicable, then the alternative method of selecting a treatment plan can be used. In addition to alternative plans, a planning component can also reason about alternative ways of executing the tasks specified by a plan. For example, if the oncology nurse becomes ill and cannot perform specified tasks (perform toxicity tests, administer treatment, etc.), then plan execution must determine an alternative way of having these tasks performed (e.g. negotiate with other oncology nurses). This enables a planning component to reason about alternative ways to execute tasks when the primary means of executing them are unavailable.

5.6.7 Co-ordinating Control

A planning component's control strategy is responsible for managing the construction and execution of problem-solving plans. It accomplishes this by monitoring incoming messages, deciphering them, and performing meta-problem solving using its available control functions and meta-problem knowledge to plan and co-ordinate interactions with agents who are capable of carrying out the necessary tasks.

This may involve co-ordinating activities for several problems which the agent has been requested to solve (e.g. an oncologist managing care for several patients). In this case, the planning component must co-ordinate several active plans and manage the co-operative problem solving required to execute all or some of these plans.

Appendix A presents a more detailed example of the application of the knowledge scripts and control functions to a patient experiencing particular cancer symptoms.

5.7 SUMMARY

This chapter illustrates the application of the model of distributed problem solving presented in Chapter 4 to problem solving in the medical domain of cancer diagnosis and treatment. The complexity inherent in the domain necessitates co-operative problem solving among several agents (health-care professionals). A model was presented in which each agent is viewed as consisting of a planning component and a problem-solving component. The problem-solving component performs the actual problem-solving functions, while the planning component is responsible for planning and co-ordinating both autonomous and co-operative problem solving. The knowledge manipulated by the planning component of the various agents was examined in order to illustrate that each agent must contain three basic types of knowledge to facilitate co-operative problem-solving: plan knowledge representing the problem-solving tasks that must be performed to solve given problems; task knowledge representing the tasks that an agent knows can be performed either by itself (its problem-solving component) or by some other agents in the environment; and agent knowledge describing some of the other agents in the environment and how to communicate with them (the knowledge presented in the examples represents only a portion of the knowledge that each agent would possess). The examples presented demonstrate how the agents use each of these three types of knowledge to plan and co-ordinate co-operative problem solving. As a result, the

chapter illustrates how the proposed model of distributed problem solving can be used to enable individual agents to co-operate with one another in order to produce a synergy of their expertise.

Chapter VI

CONCLUSIONS

6.1 SUMMARY OF DISTRIBUTED PROBLEM-SOLVING CHARACTERISTICS

A distributed problem-solving system consists of a collection of semi-autonomous agents that are capable of performing problem-solving tasks individually and, more importantly, are capable of interacting to solve problems co-operatively. Each agent is assigned a specific set of problem-solving responsibilities and is programed with local expertise enabling it to carry out these responsibilities. The agents are also provided with a mechanism that enables them to plan and co-ordinate co-operative interactions with one another when they require assistance from other agents in order to carry out their assigned tasks. The agents therefore function as a team, resulting in a synergy of their problem-solving abilities which enables them to solve problems that could not be solved by any one individual agent.

Distributed problem solving combines the research interests and efforts of artificial intelligence and distributed processing. The majority of AI research has concentrated on applying knowledge-based programming techniques to create individual agents which are programmed with knowledge that enables them to solve difficult problems. These agents work well with problems that are within the scope of their knowledge, but their performance degrades swiftly when they encounter problems that stretch past the boundaries of their expertise.

Distributed processing systems, on the other hand, are comprised of multiple, disparate tasks which are executed concurrently. The goal of these systems is to divide a problem into a set of independent tasks and distribute them to several agents (processors). The systems are designed so that an agent rarely needs assistance from another agent to carry out its problem-solving functions. In a distributed processing environment, the developers predefine the co-operative activities of the agents. Co-operative actions are not managed by the agents themselves, but by a global controller which decides how a particular problem is to be subdivided and distributed to appropriate agents. Often the agents do not know that other agents exist; interactions among the agents are carried out indirectly through accesses to shared informational resources which are managed by the global controller. The goal is to preserve the illusion that each agent (task) is executing alone on a dedicated system by having the network-operating system (global controller) hide the resource-sharing interactions and conflicts among the agents in the network [Shapiro 87].

Distributed problem-solving research concentrates on integrating the problem-solving characteristics that these two research areas provide in order to create agents which are capable of performing knowledgeable, if not expert, problem solving and are capable of planning and co-ordinating interactions with one another to solve problems co-operatively. Distributed problem-solving methodologies provide the ability to interconnect individual problem-solving agents to form a problem-solving network, and the ability to have the agents themselves co-ordinate co-operative problem-solving activities in order to produce

a synergy of their problem-solving expertise. Agents in a distributed problem-solving system are explicitly aware of the distribution of network components (and expertise) and can make informed interaction decisions based on that information [Shapiro 88].

6.2 THE MODEL OF DISTRIBUTED PROBLEM SOLVING

Researchers have developed many criteria for modelling the competence of computer systems. Competence is evaluated and analyzed with two major classes of performance measurement: effectiveness and efficiency. Effectiveness is the degree to which the goals of a system are achieved. Efficiency is a measure of the resources used to achieve the desired goals. Effectiveness does not necessarily imply efficiency. A system can be very effective in attaining its goals, but very inefficient if it expends tremendous resources to attain its goals. Conversely, a system may be efficient in its use of resources, but ineffective in that it doesn't achieve its goals. A competent system is both effective and efficient, although the emphasis on effectiveness and efficiency typically depends on the target application.

Newell introduced a methodology for analyzing the competence of single-agent systems in which a computer system is stratified and analyzed in two distinct levels: the symbol level which deals with how the system is represented using symbols and symbol structures, and the knowledge level which is above the symbol level and which investigates the competence of the system (i.e. what it can do) [Newell 81]. These two levels are sufficient for modelling single agent systems, but further research has indicated the need for a third level, known as the

organizational level, in order to model the competence of distributed problem-solving systems [Fox 87]. The organizational level provides a means of studying multi-agent systems and addresses issues such as the organization of agents into groups, the relationships between agents, the distribution of task and control responsibilities throughout the organization, methods for interaction and negotiation to allow the agents to solve problems co-operatively, the impact of inconsistent and incomplete knowledge, the impact of conflicting goals and perspectives among the agents, the effective utilization of resources, and the impact of resource contention.

This dissertation proposes that a fundamental requirement of a model of distributed problem solving is the ability to provide each agent with an explicit, knowledge-based mechanism for co-ordinating problem-solving functions at the organizational level. This premise is examined using a model of distributed problem solving which satisfies the proposed requirement by facilitating the division of problem-solving agents into two knowledge-based components: a problem-solving component and a planning component. Each agent's local expertise is contained in its problem-solving component which is responsible for performing the agent's assigned tasks. Problem-solving components, however, do not perform functions at the organizational level. Organizational knowledge is maintained and applied by the agent's planning component in order to co-ordinate an agent's actions at the organizational level. The planning components are responsible for planning and co-ordinating distributed problem-solving activities which ultimately enable the agents to work together to solve problems co-operatively.

For the most part the agents function independently, with each agent's problem-solving component performing its assigned tasks. However, when an agent's problem-solving component requires assistance to carry out a task, it can interact with its planning component to request that it arrange to have other agents provide the necessary assistance. The planning component uses explicit knowledge of the agent's environment to plan and co-ordinate co-operative interactions with other agents to request that they carry out the necessary problem solving tasks. The planning component manages the distribution of tasks and the co-ordination of co-operative problem solving and returns the results produced to the problem-solving component as they are received, integrating results when necessary. An agent's planning component also co-ordinates activities for problem-solving requests that the agent receives from other agents, distributing tasks to its problem-solving component and other agents as necessary. Maintaining two separate components with distinct problem-solving responsibilities enables an agent to reason about several aspects of its operation: how it can perform autonomous problem-solving functions, when it should perform problem-solving functions for other agents, and when it should have other agents perform problem-solving functions for it as well as which agents should perform this problem solving.

The planning component of each agent maintains a knowledge-based description of the agent's perspective of the organizational level, including its own problem-solving abilities and those of some of the other agents in the environment. This knowledge base consists of three distinct types of knowledge, each of which can be used to co-ordinate

particular types of activities required to carry out distributed problem solving effectively. These categories of knowledge are referred to as plan knowledge, task knowledge, and agent knowledge. Separating the knowledge into explicit categories leads to a more maintainable and flexible description of the problem-solving environment.

Plan knowledge describes meta-problem-solving plans which specify the tasks to be performed to solve various problems. These plans represent meta-level algorithms which specify problem-solving steps (tasks) that are to be carried out, either by the agent's problem-solving component or in co-operation with other agents. The plan knowledge, however, does not specify which agents are to perform each step. Instead, task knowledge describes the problem-solving functions that are available in the environment, both those available in its problem-solving component and those potentially available in other agents. Plan knowledge is applied to determine the tasks that are to be performed to fulfill problem-solving requests, and task knowledge is applied to determine which agents can perform each specified task (task knowledge can also specify additional task attributes and preferences as to which agents should perform a task). In addition to plan and task knowledge, a planning component's knowledge base contains agent knowledge which describes the agents in the environment, the relationships among the agents, and the communication protocols that are required to interact with these agents effectively. This knowledge is used to co-ordinate interactions with other agents in order to distribute tasks appropriately.

Maintaining explicit categories of knowledge enables the control mechanism of a planning component to process problem-solving requests in three stages: determine the tasks to be performed by selecting and applying the appropriate problem-solving plans; determine which agents or class of agents can perform each task; and determine specific agents within the environment that belong to the specified agent classes and the protocols necessary to communicate with these agents to request that they perform the specified tasks. Control in a planning component therefore involves selecting and applying plan, task, and agent knowledge to co-ordinate an agent's actions at the organizational level. In order to select and apply this knowledge effectively, a planning component's control strategy can interrogate agents to obtain additional information about problem-solving requests, task descriptions, and communication protocols and can negotiate with agents when incompatiblities arise that require the parties involved to determine mutually acceptable compromises. The ability to interrogate and negotiate enables the agents to overcome incomplete or incompatible problem solving specifications co-operatively and thus increase their ability to produce a synergy of their expertise.

6.3 EVALUATING THE MODEL

This section examines the model of distributed problem solving defined in this thesis using Hewitt's criteria from Chapter 3. This examination discusses the extent to which the model incorporates the major features required to create effective co-operative problem-solving environments.

One of the most important features of a distributed problem-solving system is the ability to have its member agents engage in decentralized decision making. The model facilitates decentralized control by allowing control functions and responsibilities to be distributed to each agent in the environment. When an agent wishes to engage in co-operative problem solving with other agents, its planning component sends problem-solving requests to the agents which are capable of carrying out the desired problem solving. These problem-solving requests specify the tasks to be performed, but do not specify how the recipients are to perform the tasks (a request can include constraints and other problem specifications which the recipients are requested to use in performing the tasks). Consequently, all interactions between agents are planned and co-ordinated by the agents' respective planning components and the system as a whole functions as a product of the actions of its member agents and the interactions among the agents. No single agent or group of agents can control other agents directly; responsibility for controlling the system's actions is distributed throughout the environment and the agents must co-operatively determine suitable interactions that will achieve the desired synergy of their expertise.

In addition, the model provides a developer with the ability to experiment with the distribution of control because each agent's distributed problem-solving functions are explicit and programmable. For example, agents can be organized into agencies in which one agent functions as the manager, co-ordinating the activities of its subordinates and access to their pool of expertise by agents outside of

the agency. The degree of control that the managing agent wields over its subordinates is dictated by the application at hand, not by predefined organizational constraints imposed by the model. Consequently, the developer is able to distribute the necessary control functions and responsibilities to the appropriate agents and can program explicitly the agent-to-agent control relationships. Furthermore, the distribution of control throughout the environment is represented explicitly (in the knowledge base of each agent's planning component), making it easier to understand and maintain the organizational level of the environment.

In conjunction with decentralized control, the model provides the agents with the ability to negotiate. The planning component of each agent is capable of performing negotiations with other agents in order to resolve difficulties that occur during distributed problem-solving activities. When an agent requests assistance from other agents, its planning component will initiate co-operative problem-solving activities by sending problem-solving requests to the agents which can provide the necessary assistance. If these agents determine that the requests are incomplete or are incompatible with their problem-solving abilities or schedules, then they can respond with messages describing the difficulties that prevent them from carrying out the requests. The planning component of the agent which sent the requests can reason about these responses and co-ordinate further interactions with the agents to reformulate the requests co-operatively. As a result, interactions among the agents are not restricted to predefined agreements which impose undue constraints on the ways in which the agents can interact to solve problems co-operatively; the agents would be restricted to

all-or-nothing interactions, each of which must be anticipated in advance and accounted for in each participant. Instead, the agents can negotiate mutually acceptable interactions based on their local needs and abilities and those of the other agents in the environment.

The planning component of each agent is responsible for co-ordinating negotiations based on the information exchanged between the agents during the negotiation process; this process is driven by knowledge maintained by the agent's planning component describing the ways in which problem-solving functions can be reformulated when incompatibilities occur. A planning component may interact with its problem-solving component to obtain additional information describing its problem-solving abilities (e.g. is a reformulated problem-solving request acceptable?), but the problem-solving components are not directly involved in the negotiation process. Negotiation allows the agents to co-operate in a significantly more flexible and robust manner which ultimately results in a greater synergy of their problem-solving expertise. Negotiation can become counter productive if the agents are unable to determine the most cost effective means of negotiation or are unable to detect when further negotiations are inappropriate. The model provides an explicit mechanism for specifying these types of knowledge in the form of relaxation methods associated with the various knowledge scripts.

Another important feature of the model is the ability to maintain the individual perspectives of the agents of a distributed problem-solving environment. Each agent is able to maintain explicit perspectives of its own problem-solving abilities and roles and those of some of the

other agents in the environment. Using decentralized control enables the agents to interact based on their individual perspectives, and the ability to negotiate during decentralized decision making enables the agents to overcome conflicts and inconsistencies that may exist among their perspectives. Inconsistent or conflicting perspectives are not meaningless, rather they play a major role in complex applications which require the integration of several views of problems. Rarely does an agent or set of agents have all of the information that is required to produce a complete and infallible solution. Instead, the agents must contribute their views and abilities to the solution, while at the same time respecting the views and abilities of the other agents in order to weigh evidence for and against potential solutions and to compromise for the good of the global environment when their views conflict. By maintaining the individual perspectives of the member agents, the model provides the ability to capture the fundamental characteristics of complex applications, while decentralized control and negotiation enable the agents to co-ordinate a synergy of the problem-solving expertise, even when faced with incomplete, inconsistent, or conflicting information.

In many cases the agents will share perspectives, resulting in a natural form of redundant problem-solving resources which, when used wisely, can produce a significantly more robust environment. The knowledge base of an agent's planning component can describe several alternative plans that can be used to solve a problem (or class of problems) and several agents (or classes of agents) which are capable of performing the tasks specified by the various plans. The result is a

significantly more robust and flexible problem-solving environment in which agents are able to cope with bottlenecks and disabled portions of the environment by applying alternative methods of co-ordinating activities when primary methods are not applicable (e.g. alternative plans or alternative resources). Furthermore, if the agents in the environment share some common problem-solving abilities, then an agent which possesses scarce skills and expertise can distribute some of its less demanding tasks to other less skilled agents so that its advanced skills are used effectively.

Distributed problem-solving systems by nature must be able to evolve continuously through modifications and extensions to their member agents and the ability of the agents to interact to solve problems co-operatively. By separating an agent's distributed problem-solving functions from its basic problem-solving functions, the model is able to satisfy this requirement. Each agent's planning component is able to maintain, manipulate, and extend an explicit knowledge-based model of the agent's environment which describes the agent's perspective of the organizational level and of its knowledge level. Initially, an agent's planning component will be programmed with knowledge describing the agent's own problem-solving capabilities and partial knowledge of some of the problem-solving abilities of other agents. As agents interact to solve problems co-operatively, the planning components can extend and modify their knowledge of the abilities of other agents and the protocols required to communicate with the agents effectively. For example, an agent's planning component can interrogate other agents to determine whether they can perform a given task and can use the

resulting responses to extend its knowledge of the abilities of the agents. Thus, in theory, each agent is able to strengthen its connections with other agents through experience by refining and extending its knowledge of their abilities and needs. Most importantly, an agent can extend and modify its knowledge of the problem-solving resources available in the environment without having to determine when and where these new or modified resources should be used. The agent simply updates its knowledge of the organizational level appropriately and subsequent distributed problem-solving functions that its planning component must perform can utilize the new or modified resources when they are applicable. In practice, a planning component will require a learning mechanism that can examine newly acquired knowledge and determine whether or not it should be used to modify or extend its knowledge base and, if so, how these modifications and extensions should be carried out.

The model also provides for the incorporation of new agents into the environment in a smooth and orderly manner. The task knowledge maintained by the planning components describes the problem-solving tasks that can be performed and the class of agents (and possibly specific agents) that can perform each task (a task can be carried out by an agent itself or in co-operation with other agents). Representing the class of agents that can perform each task enables a planning component to reason abstractly about the problem-solving abilities of other agents. As a result, simply knowing the class of agents to which a new agent belongs allows the other agents in the environment to infer an extensive description of the new agent's problem-solving abilities (or

potential abilities). Some of these inferred abilities may have to be refined after subsequent interactions with the new agent reveal additional problem-solving abilities or the agent's inability to perform some of the tasks attributed to the class of agents to which it belongs. Nevertheless, this facility enables additional agents to be incorporated into the environment without undue burden on the existing agents. Furthermore, a newly added agent can use the same facility to automatically infer a large amount of information describing the problem-solving abilities of the other agents in the environment (assuming that the new agent already has task knowledge describing the problem-solving abilities of the various classes of agents). In the medical domain described in Chapter 5, this would be equivalent to an oncologist moving from one city to another: the oncologist's agent knowledge of specific agents would no longer be appropriate, but the task knowledge and agent knowledge describing agent classess would need no, or only minor, changes. In both cases, the agents would simply extend their agent knowledge to describe the existence of the new agent or agents. An agent's planning component will take these new agents into consideration when it is attempting to distribute tasks which the agent's class can perform.

6.4 ANALYSIS AND FUTURE WORK

The model of distributed problem solving presented in this thesis provides a facility for studying the organizational level of distributed problem-solving environments. Although the model provides a framework for constructing co-operative problem-solving systems, a developer must

address specific details of the target application in order to realize a computational entity. The model provides a representation paradigm for encoding the basic types of knowledge needed at the organizational level, but a developer must determine the appropriate knowledge to be represented at this level in each agent. The model outlines the general control strategy required at the organizational level, but an actual implementation will require the developer to tailor the control mechanism to the characteristics of the target application. Furthermore, an application-specific learning mechanism for determining when the agent's organizational knowledge should be extended as new plan or resource knowledge is acquired through experience is also required. Most importantly, the developer must determine the distribution of the task and control responsibilities among the agents in the target application and the distribution of responsibilities between the organizational and knowledge levels of each agent. These decisions can have a dramatic impact on the effectiveness of the ensuing system. The model does, however, provide a natural framework for experimenting with the distribution of task and control functions within an agent and among the agents. Thus, the model provides a flexible and explicit methodology for empirically evaluating the needs of the organizational level of distributed problem-solving applications.

According to Shapiro, the applications to which distributed problem solving is applicable can be divided into three basic categories: distributed interpretation applications which require the interpretation and analysis of distributed data to generate a model of the data (e.g. distributed sensor networks and network-fault diagnosis); distributed

planning and control applications which involve co-ordinating the actions of a number of agents to perform desired tasks (e.g. groups of co-operating robots, multi-user project co-ordination, intelligent command and control systems, and co-operative resource allocation systems); and distributed expert system applications in which expert system technology is applied to larger, more complex problem domains by developing co-operative interaction mechanisms that allow multiple expert systems to work together to produce a synergy of their problem-solving expertise [Shapiro 87, p. 246]. The model of distributed problem solving described in this dissertation is geared primarily towards the latter class of applications. This class of applications is illustrated in Chapter 5 in which the problem of cancer diagnosis and treatment is examined using a co-operative environment of artificial agents each of which specializes in a particular subset of the domain. The model is, to a lesser extent, applicable to the two other classes of applications.

The model, however, is not applicable to applications in which the agents are generally uncooperative. Such applications require additional research to study the effects of uncooperative activities such as agents who deliberately provide incorrect information during interactions or who maliciously tamper with shared resources. The model assumes that the agents are willing to co-operate, and provides each agent with an explicit component to co-ordinate its activities at the organizational level. Each agent is able to co-operate with other agents using a knowledge-based model of its own abilities and of those of the other agents. As a result, the agents are able to co-operate

more effectively to produce a greater synergy of their problem-solving abilities.

The major contributions of the research presented in this dissertation are:

- A description of a conceptual model of distributed problem-solving which provides a framework for constructing agents containing an explicit, knowledge-based component which can co-ordinate problem-solving functions at the organizational level;
- Identification of the major types of knowledge required at the organizational level and the major control functions required to apply the knowledge in order to carry out co-operative problem solving;
- A specification of the representation mechanisms required to represent the various types of organizational knowledge, including control knowledge that can be used to determine when particular problem-solving methods are applicable. A mechanism for employing relaxation methods to reformulate incompatible interactions through co-operative negotiations is also specified;
- A specification of the basic control functions which co-ordinate an agent's actions at the organizational level. This specification describes a control strategy based on a multi-agent constraint-satisfaction planning mechanism;
- A description of a methodology for enabling the agents to modify and extend their knowledge of the organizational level through co-operative interactions.

Future work to be addressed will involve prototyping various applications in order to further refine and extend the representation and control mechanisms specified in the model. This work will enable issues in distributed problem solving (such as the organization of agents into groups, the distribution of task and control responsibilities among the agents, methodologies for enabling the agents to interact and negotiate, etc.) to be examined. This research will also lead to a better understanding of the conceptual needs of

distributed problem-solving systems and of the engineering issues involved in creating operational systems based on the conceptual framework.

Appendix A

SPECIFICATIONS AND GUIDELINES FOR IMPLEMENTING THE MODEL

A.1 OVERVIEW

This appendix addresses the issues involved in creating a computational entity based on the model of distributed problem solving presented in Chapter 4. The description of the model outlines the organizational principles that can facilitate co-operative problem solving, but it does not provide a detailed specification of how an operational system can actually be constructed, that is, the model is a conceptual entity, not a computational specification. Consider the following statements by Nii:

> "Problem-solving models are conceptual frameworks for formulating solutions to problems. The models do not address the details of designing and building operational systems. How a piece of knowledge is represented, as rules, objects, or procedures, is an engineering decision. It involves such pragmatic considerations as "naturalness", availability of a knowledge representation language, and the skill of the implementers, to name a few. What control mechanisms are needed depends on the complexity and the natural of the application task." [Nii 1986]

What is evident in Nii's comments is that a more detailed specification of the engineering details required to build an implementation based on the model is necessary. The very nature of the model prevents the specification of an all-encompassing implementation; however, we can attempt to narrow the gap between the model and an operational system. In the following sections, I address the engineering issues surrounding the construction of a functional

implementation of the model. The specifications presented in these sections identify the representation and control mechanisms required to realize an operational system based on the conceptual model of distributed problem solving. It is important to understand that these discussions outline representation and control mechanisms that address the engineering level of the model. A target application dictates the knowledge that is to be represented and used to drive the control activities of an agent. Thus, a particular application dictates many of the lower-level details of the the representation and control mechanisms required to create an operational system.

A.2 KNOWLEDGE REPRESENTATION

A.2.1 Overview

In the framework of the model, agents co-ordinate co-operative problem solving by creating, distributing, and interpreting problem-solving requests. Problem-solving requests are structures which describe an operation (or set of operations) to be performed, an object (or set of objects) to which the operation is to be applied, and possibly an agent (or set of agents) which are to carry out the operation. Each agent is divided into a planning component and one or more problem-solving components. An agent's planning component interprets problem-solving requests that the agent receives and determines the problem solving required to fulfill a request and the agents capable of performing this problem solving (a request may be performed by one or more of the agent's problem-solving components, other agents, or some combination thereof).

In the remainder of this section, I examine the form and content of problem-solving requests and of the various types of knowledge a planning component requires to process requests effectively.

A.2.2 Representing Problem-Solving Requests

In order to represent problem-solving request structures, we require a language that provides the following facilities:

- A facility for specifying generic frames representing generic objects and requests built from these objects. Generic frames specify abstract characteristics of objects and requests such as generic values, defaults, demons, and constraints. In addition, this facility may also provide the ability to define specializations of generic objects and requests. In this case, it must provide an inheritance mechanism which enables specializations to inherit all or some of the form and content of their parent frames. Specializations should be able to override inherited characteristics or append additional characteristics.

- A mechanism for specifying specific instances of generic objects and requests and their specializations. An instance will represent a copy of a generic frame with all or some of the components filled in with actual object instances. Instances should be able to override or append particular generic characteristics when necessary.

- A mechanism for defining the form and content of the slot structures of a frame which specify the following:

  - Actual values and/or default values. These values can be constants, generic objects, or object instances.

  - Demons in the form of arbitrarily complex expressions which can be evaluated to generate a slot value or to fill in values for other slots based on a given value (these expressions can include references to the values of other slots, function invocations, etc.). For example,

    Appointment: IF-NEEDED (first-available-time)

    The function FIRST-AVAILABLE-TIME examines the agent's agenda and returns the first available time period. This example illustrates that it may be necessary for functions to access internal databases, tables, etc.

```
Accounting-cost: (?a number)
Legal-cost: (?a number)
Total-cost: IF-NEEDED ( (accounting-cost FROM SELF) +
                        (legal-cost FROM SELF))
```

This expression stipulates that total cost is the sum of the accounting cost and legal cost (note: FROM SELF denotes an access to a particular slot value in the current frame).

- Constraints in the form of arbitrarily complex logical expressions that evaluate to TRUE or FALSE. Logical expressions can include: functions, relational operators, expressions, etc. For example,

```
Location: CONSTRAINT ( distance(nearest-school) < 1)
```

Where nearest-school is a function that returns the location of the school nearest to the location specified as the value of the slot. This function would search a database containing locations of schools. The distance function will generate the distance between two locations.

```
Price:  CONSTRAINT ( (< 100), (> 500))
```

A conjunctive expression stating that the slot value must be less than 100 AND greater than 500.

```
Style:  CONSTRAINT ( (= BUNGALOW); (= SPLIT-LEVEL))
```

A disjunctive expression stating that the slot value must be either BUNGALOW or SPLIT-LEVEL.

- Inheritance specifications denoting the type of inheritance to apply for the specific slot (e.g. APPEND, OVERRIDE, etc.).

The example in Figure A.1 illustrates a sample set of request structures constructed using a generic representation language (the actual language is not formally defined, it simply serves to illustrate the type of mechanisms that are required). This example contains a generic frame (PURCHASE), a specialization of that frame (HOUSE-PURCHASE), and an instance of the specialization (REQUEST-101). In addition, the example contains a generic object frame (HOUSE) which

```
purchase    REQUEST  TYPE  generic

    Operation:  purchase
    Object: (?a general-object)
    Buyer: ( (?a person); (?a organization))
    Seller: ((?a person); (?a organization))
    Payment: (cash;credit-card;mortgage)
```

```
house-purchase     REQUEST  TYPE  specialization

       purchase   WITH

    Object: (?a house)     Inherit(OVERRIDE)
    Payment: (cash;mortgage)   Inherit(OVERRIDE)
```

```
request-101   REQUEST  TYPE  instance

       house-purchase   WITH

     House.Style:  Constraint( (= BUNGALOW))
     House.Price:  Constraint( (< 100000))
     Buyer: PERSON-37
     Payment: MORTGAGE     Inherit(OVERRIDE)
```

```
house     OBJECT  TYPE   generic

     Color: (?a color)
     Style:  (BUNGALOW;SPLIT-LEVEL;...)
     Location: (?a address)
     City:  (?a city)
     .
     .
```

Figure A.1: Sample Generic Frames, Specializations, and Instances

is used to describe a component of the HOUSE-PURCHASE frame. As evident in this example, a specific request instance does not need to specify a complete instantiation of a generic request. Instead, instances typically specify the characteristics of various objects that will constitute an acceptable solution to the request. The recipient of the request (e.g. a real-estate agent) must fulfill the request by filling in the frame with specific values that satisfy all of the specified characteristics. For example, the request instance in Figure A.1 specifies that the buyer is PERSON-37, an object instance of a generic object (PERSON), and that the payment will be made via a mortgage loan. In addition, two constraints are associated with part of the request:

- the value assigned to the style slot of a prospective house structure must be BUNGALOW, and
- the value assigned to the price slot must not exceed 100000.

Therefore, in order to fulfill this request, the recipient must find a house which matches the characteristics specified in the request - a BUNGALOW, priced at or below 100,000.

A.2.3 Representing the Knowledge

In order to co-ordinate co-operative problem solving activities, a planning component requires extensive knowledge describing the types of problem-solving requests that it can process, the methods it can use to process each type of request, and the agents that can carry out the problem-solving specified by a particular method, including the requests that it can distribute to its problem-solving components and those it can distribute to other agents in the environment. The types of requests that a planning component may be required to process can be divided into three categories:

- COMPOSITE REQUESTS: these are requests that can be transformed into a set of simpler requests which can be solved individually. The results of solving this set of requests typically must be integrated to produce a solution to the original request.

- PRIMITIVE REQUESTS: these requests can be distributed directly to appropriate agents. The recipient of a primitive request may execute it immediately (e.g. a request distributed to a problem-solving component), or if the recipient is another planning component, then the recipient may view the request as a composite request and transform it, distribute a simpler set of requests, and integrate the results.

- FLEXIBLE REQUESTS: these are requests that can be placed in both of the previous categories: they can be processed either as composite requests or as primitive requests. The method used to process this type of request typically depends on the planning component's workload, the priority of the request, and the availability of other agents. For example, a planning component may decide to distribute a low priority request to a subordinate rather than apply available transformation methods because the planning component's workload cannot accommodate the time and effort involved in co-ordinating a solution using the transformation methods.

The following are the three basic types of knowledge required by a planning component:

- PLAN knowledge specifying how to process composite requests. This knowledge must include a definition of the requests that are considered composite, a plan definition specifying how to transform a particular composite request into a set of simpler requests, and control information that dictates dependencies and constraints among these simpler requests as well as the integration functions that must be performed to produce a final solution from the results of the individual requests.

- TASK knowledge specifying the processing required to solve primitive requests. This knowledge must specify a definition of the requests that can be processed as primitive requests, and the class of agents (or specific agents) that are capable of executing a particular primitive request (i.e. a problem-solving component or other agents).

- AGENT knowledge specifying the processing required to communicate with the various agents in the environment. This knowledge must specify the address of each agent in the environment, the communication protocols needed to interact with each agent and typically a definition of the class or classes of agents to which each agent belongs. In addition, this knowledge can associate agent-specific task knowledge with each agent. This task knowledge

typically defines additional information describing specific characteristics of the requests that each agent can perform. Thus, general task knowledge identifies the agents that can perform particular requests, and agent-specific task knowledge describes specific requirements that individual agents may impose on these requests.

Let us now consider a knowledge representation that can be used to encode these types of knowledge. Chapter 4 suggested a representation scheme which requires a language that can express the three types of knowledge in the form of knowledge scripts. Figures A.2 and A.3 illustrate the type of information that such a language must be capable of expressing, including:

- TYPE IDENTIFIER: indicates the type of entity represented in the script (PLAN, TASK, AGENT).
- PRECONDITIONS: A precondition is simply a particular request instance which defines the type of problem-solving requests to which the plan, task, or agent information described by the script is applicable. A particular problem-solving request is said to match the precondition of a knowledge script if the two structures are instances of the same generic request (or compatible specializations thereof) and the components of the two structures are compatible. Two components are considered compatible if:
  - they both specify the same object instance;
  - they both specify generic objects (or specializations) and the constraints associated with these objects are consistent;
  - one component is an object instance that satisfies all of the constraints associated with the other component (which is an generic object);
- DESCRIPTION: this structure specifies the plan, task, or agent information represented in the script.
  - For plan scripts this includes:
    - A set of plan steps that must be executed to fulfill the request. Typically, these steps specify problem-solving requests that represent the tasks involved in satisfying the entire request. A plan may also specify function calls as a plan step. For example, the second step of the plan in Figure A.2 specifies a function (CHOOSE-HOUSE) which takes the results from the first step (a set of houses that match the

```
Type:  PLAN
Applicable-to:    house-purchase    {precondition}
-------------------------------------------------------
Plan:  DO
        (step-1  find-house  WITH
             Object:  FROM precondition)

        (step-2  (choose-house  (object FROM step-1))

        (step-3  arrange-a-deal WITH
              Object: FROM step-2
              Seller: FROM step-2
              Buyer: FROM precondition)

    Solution =   house-purchase  WITH

              Object: FROM step-3   {house with arranged price}
              Seller: FROM step-2
              Buyer: FROM Precondition
              Payment: FROM Precondition
```

```
Type:  TASK
Applicable-to:   find-house    {precondition}
-------------------------------------------------------
Agent-Class:   real-estate agent
Agent-id:    SELF
```

Figure A.2: Sample Plan and Task Script

given criteria) and selects a particular house to pursue (based on specific criteria embodied in the function). Backtracking may be necessary if the chosen house conflicts with the rest of the plan (e.g. a price cannot be negotiated).

- A control specification describing the processing involved in executing the plan. This includes control structures for specifying temporal orderings among tasks, iteration, conditional tasks, etc.

- A definition of the integration functions required to obtain a complete solution from the results of the plan steps. In the sample plan, a solution is defined as a house-purchase request structure which specifies how the values for the various components are to be obtained (e.g. from the precondition or from a specific plan step).

```
Type:  AGENT
Applicable-to:    {no specific preconditions}
-------------------------------------------------
Network-id: agent-101
Address:  SELF  {the problem-solving component}
Can-Perform:  (find-house, arrange-a-deal, ...)
```

```
Type: AGENT-SPECIFIC-TASK
Agent: SELF   {which agent does the script belong to}
Applicable-to:  find-house  WITH

          House.City: CONSTRAINT ( (= WINNIPEG))

            {this agent can only find houses in
             Winnipeg}
```

Figure A.3: Sample Agent and Agent-Specific Task Scripts

- A set of constraints that must be satisfied in order for the plan to remain valid. These constraints can be specified in the solution definition to represent dependencies among the plan steps that must be obeyed as the solution evolves.

- For task scripts this includes:
  - A list of the agents or classes of agents that can perform the task represented by the script.
  - An optional request structure that defines additional characteristics of the request. This information is merged with that of the request structure in the precondition of the script. If additional information is not required, then the request structure specified in the precondition is used.

- For agent scripts this includes:
  - A definition of the information required to communicate with a specific agent (e.g. address, routing protocols, etc.)
  - A list of agent-specific task scripts which define specific requirements that an agent imposes on one or more of the

requests (tasks) that it is capable of performing. If a particular task that the agent can perform does not require an agent-specific task script, then the information in the general task script will be used to construct the appropriate request to be sent to the agent.

- OTHER INFORMATION: in addition to the two basic structures, a script can have any number of additional structures representing knowledge describing relationships among scripts, including heuristic knowledge that can be used determine which of a set of applicable plans, tasks, and agents are most applicable to a particular problem-solving request.

A.2.4 Organization of the Knowledge Base

Throughout this section I have assumed that a structured representation is used to represent request structures and knowledge scripts. It is feasible, however, to use a linear representation in which the knowledge base is viewed as set of knowledge scripts, each of which completely describes a particular plan, task, or agent. The only structuring of the knowledge base is the partitioning of the knowledge scripts according to the category of knowledge to which they belong. For example, each script within the plan category will completely describe a particular plan: a precondition (request structure) and a plan definition. All request structures and knowledge scripts must be completely specified because an inheritance mechanism is not available in a linear representation. This approach is feasible if the number of scripts is small and easy to maintain. However, as the number of scripts increases and the relationships among the scripts grows, a structured approach is required. Consider the following advantages inherent in structured representations:

- Early pruning of entire classes of scripts during introspection. If the precondition of a high-level script is not met, then its subordinates can be pruned (provided they do not override the precondition in question).

- Incorporation of control knowledge in the high-level scripts that can direct the selection or pruning of subordinates (e.g. the MORE-SPECIFIC slot described in Chapter 4).

- The knowledge is represented more explicitly and naturally thus enabling large collections of scripts to be developed and maintained with greater ease (although consistency problems inherent in structured representations must be addressed as outlined in [Levesque 84]).

A.3 BASIC CONTROL ISSUES

A.3.1 Overview

In order to formalize the control specification outlined in Chapter 4, it helps to view control as a constraint-directed search problem. Fox states that:

> "Domain knowledge, encapsulated in the form of constraints, can be used to develop a better understanding of the structure of the problem space which will lend to more efficient problem solvers whose:
>
> - architectures take advantage of known problem space structures, and
> - focus of attention methods reduce the amount of search." [Fox 84]

From this perspective, co-operative problem-solving can be viewed as a multi-agent constraint-satisfaction planning problem in which agents use constraints to direct and co-ordinate co-operative problem solving activities. The constraints that must be manipulated during co-operative problem-solving can be divided into five basic categories:

- Organizational Goals: maintaining an effective use of available resources (particularly agent resources) by constraining their use. For example, preventing scarce resources (e.g. highly skilled agents) from being used on low priority tasks, preventing bottlenecks during co-operative work (deadlocks), and balancing the use of idle resources effectively.

- Physical Restrictions: identifying physical limitations of resources such as the processing time involved in and the information required to perform particular tasks.

- Temporal Restrictions: identifying temporal orderings among the individual problem-solving requests an agent receives and among the tasks required to carry out individual requests (e.g. the order in which the steps of a plan must be performed).

- Availability Restrictions: identifying resource availability and the implications of unavailable resources (e.g. task X cannot be performed unless agent A is available).

- Preferences: identifying preferences for plans, tasks, and agents as well as preferences for negotiation methods that may be available.

The knowledge representation described in the previous section provides a natural framework for representing constraints and for exploiting constraint-directed control strategies in order to co-ordinate co-operative problem-solving activities. The representation of constraints in the knowledge scripts can be summarized in the following manner:

- The preconditions of plan scripts identify the type of problems that a planning component can process using divide-and-conquer methods.

- The preconditions of task scripts identify the type of problems that a planning component can process as primitive requests.

- The preconditions of agent scripts (agent-specific task scripts) identify additional constraints that individual agents impose on specific tasks (primitive requests). For example, one or more of the agents identified by a task script as being capable of performing a particular primitive request may only be able to perform specific configurations of the request.

- In addition, the constraints specified in a particular problem-solving request that an agent receives must be integrated with the constraints in the knowledge scripts. Specific constraints in a particular request may conflict with the preconditions imposed by the knowledge scripts thus further constraining the plans, tasks, and agents that can be used.

Therefore, all constraints and, as we shall see, all methods available for relaxing constraints are explicitly represented in the appropriate knowledge scripts (and problem-solving requests). A planning component requires a control strategy that can process problem-solving requests by manipulating these constraints in order to perform the following steps:

- Determine the problem-solving methods that can be used to process a particular request. There may be several methods available including several plans, several tasks, or some combination of the two.
- Select a method capable of fulfilling a particular request. This typically involves determining the most suitable method.
- Determine the agents that can perform the problem-solving tasks specified by the chosen method (there may be more than one agent capable of performing a task).
- Select an agent to carry out each task. This typically involves determining the agent that is best suited to each task.
- Determine the problem-solving requests that must be sent to these agents to have them execute the tasks (it is through these problem-solving requests that co-operative problem-solving is initiated).
- Co-ordinate the integration of the results produced by these agents (at some point the results must be returned to the source of the original problem-solving request).

In the remainder of this section, I describe an algorithm which addresses the basic control issues outlined in the model. Essentially this algorithm is a constraint-directed search with dependency-directed backtracking. Thus, a programming tool that provides these mechanisms can be employed to build an operational system based on this algorithm. (AI programming languages such as KEE and KnowledgeCraft provide dependency-directed backtracking facilities.) Subsequent sections address engineering issues that arise when attempting to extend this

algorithm to facilitate more robust forms of co-operative problem solving (i.e. negotiation). The goal is to provide a general-purpose control specification which can be applied to create the various planning mechanisms necessary to address the needs of co-operative problem-solving systems. One must remember, however, that much of an operational system will still require extensive tailoring in order to capture the specific knowledge and control characteristics of a particular application.

A.3.2 Basic Control Algorithm

1. Monitor the communications link for the arrival of problem-solving requests.

   - Represent the requests as entries on the problems partition of the activity blackboard.

2. Select a request to process and perform introspection to find applicable plan scripts (i.e. determine whether the request can be processed using a divide-and-conquer method).

   - Perform a constraint-satisfaction search which retrieves the plan scripts that are compatible with the request.

     - A script is only selected if its precondition is completely compatible with the given request (negotiation methods for processing scripts whose preconditions are not compatible are discussed in the next section). Store these scripts on a plan agenda.

   - If no applicable plans were found, then go to step 5 (the request may correspond to a primitive request).

3. Select a plan from the plan agenda and interpret the plan definition (plan execution).

   - Create a data structure describing the tasks to be performed and the control information associated with executing these tasks (i.e. an executable plan).

4. Distribute the tasks as specified in plan definition.

   - Perform steps 5 to 7 for each task that is to be distributed (may require maintaining a temporal ordering of task execution).

5. Introspect to determine the task scripts that are applicable to the specified primitive request (a primitive request received directly from an agent or specified in a plan step).

   - Applicable task scripts are stored on a task agenda. If the task agenda is empty, then the request cannot be distributed; backtrack to step 3 and select another plan, or inform the source of the request that the request cannot be processed.

6. Select a task interpretation from the task agenda and introspect to determine the applicable agent scripts.

   - Applicable agent scripts are stored on an agent agenda. If the agent agenda is empty, then the request cannot be distributed; select another task interpretation from the task agenda, or backtrack to Step 3 and select another plan.

7. Select an agent from the agent agenda and distribute the request (task) to the agent using interrogation.

   - The communication information needed to create and route an appropriate message containing the request is described in the chosen agent script.

   - Store outgoing requests on the agent activity partition of the activity blackboard. Include a request identification tag with each outgoing request. This tag is used to identify subsequent incoming messages that represent responses to distributed requests.

8. Go back to step 1 and select another request to process.

9. When responses to distributed tasks arrive:

   - Determine the request corresponding to the response by examining the request identification tag in the response.

   - If the response indicates successful completion of the request, then do one or more of the following:

     - Integrate the results with results of other completed tasks as defined in the plan definition;

     - Wait for the completion of other distributed requests;

     - Distribute other requests that were waiting for the response;

     - Route results to the destination (if processing is complete);

   - If the response indicates that the request was not completed successfully, then do one or more of the following:

     - Backtrack to step 7 and determine whether the request can be distributed to another agent;

     - Backtrack to step 6 and determine whether another interpretation of the request can be used;

     - Backtrack to step 3 and determine whether another plan can be used;

     - Terminate processing and send a response to the source of the request indicating that the request cannot be completed;

A.3.3 Additional Issues to Consider

There are several additional control issues to be addressed in the algorithm:

- In order to co-ordinate the actions of the planning component effectively and efficiently, the control algorithm requires the following facilities:
  - A mechanism for assigning each request a priority based on the type of request, the contents of the request, the source of the request, or some combination of this information. When faced with several requests to process, the planning component can use these priorities to focus its efforts towards the requests that are the most important. Furthermore, the planning component can use these priorities to determine when to suspend processing of low priority requests in favor of more important requests that subsequently arrive.
  - A mechanism for associating a quality measure with each plan, task, and agent script. This measure will indicate the quality of a particular plan, task, or agent. The planning component can use this measure to focus its efforts towards selecting the highest quality resources. These measures can be predefined constants or can be expressed as a function of particular characteristics of a given problem-solving request. For example, criteria in the MORE-SPECIFIC slot of the parent of a set of scripts can be used to determine which of its subordinates is best suited to a particular request.
- It may be necessary to interrogate the source of a request during introspection in order to obtain values for missing component information that is needed to determine whether a particular script is compatible to the request.
- A plan definition may include one or more steps which specify function calls rather than problem-solving requests. The control strategy must recognize these steps and invoke the appropriate internal functions.
- Interpreting a plan typically requires the instantiation of all or part of the specified plan steps. This instantiation involves integrating values for components of a step, either from other steps or from the precondition of the script, as illustrated in Figure A.2.
- Instead of processing the tasks specified by a plan one at a time, it may be necessary to perform a preliminary scheduling of all the tasks to ensure that the plan is feasible. A plan may prove to be infeasible for several reasons:

- The required agents are not available (they are busy or disabled), or available agents impose additional constraints that are incompatible with the requests specified in the plan steps.

- The constraints imposed among one or more of the planned steps cannot be satisfied (e.g. the results of one step are not compatible with that required by a subsequent step);

- Preliminary scheduling may involve interrogating agents to ensure that they can and will perform the requests specified by the plan.

  - Preliminary interrogations may ask one or more of the agents to provide an estimate in order to determine whether or not the intertask constraints can be met (i.e. total cost of all steps cannot exceed a predefined limit; time required to perform one step must be within the constraints specified by another step)

    - Thus, a planning component should be able to apply a heuristic function to generate an estimated solution to a request (assuming the request is compatible with the planning component's knowledge). These heuristic functions can be represented in the appropriate knowledge scripts. Generating an estimate may entail obtaining and integrating estimates from other agents (i.e. the heuristic function associated with a plan requires estimates from the agents that are to carry out the requests specified by the plan steps).

    - Estimates are not guaranteed to be exact. A plan may fail when tasks are actually performed and estimates are exceeded.

A.4 INCREASING ROBUSTNESS THROUGH NEGOTIATION

Using the control strategy outlined in the previous section, agents are able to co-operate if they can create the necessary problem-solving requests and distribute these requests to the appropriate agents. Rarely, however, is it possible to create a distributed problem-solving system in which the agents are able to interact without conflicts or incompatibilities. Rather, the power of these systems is inherent in the diversity of the beliefs, abilities, and needs of their member agents. As a result, interactions among agents often will contain incompatibilities: the constraints associated with the methods (plans,

tasks, agents) that an agent can employ conflict with the information (values and constraints) specified in a particular problem-solving request that the agent has been asked to fulfill. In many cases, however, incompatibilities can be overcome if the agents can negotiate with one another to reformulate the interactions. Negotiation can be viewed as the process of relaxing constraints to propose alternative problem configurations which are satisficing and feasible. In order to facilitate negotiation, a planning component must be able to perform the following functions:

- Employ relaxation methods that can be used to relax constraints.
- Select the focus of negotiation when incompatibilities occur.
- Decide when negotiation is successful.
- Recognize when further negotiation is inappropriate or unnecessary.

In the remainder of this section, I examine the facilities that are required to extend the basic control algorithm to incorporate these negotiation functions.

A.4.1 Representing Relaxation Methods

One way of specifying relaxation methods is to have each constraint define how it can be satisfied [Mittal 86]. This can be accomplished by extending the the representation language to provide the following facilities:

- A facility for associating a relaxation method or set of relaxation methods with a particular constraint. For Example:

```
Slot: CONSTRAINT(expression)
          UNLESS( relax(expression); relax(expression);...)

Cost: (?a number)
       CONSTRAINT( (< 100) )
                  UNLESS( relax( (< 125) ))
```

- A facility for specifying arbitrarily complex logical expressions that constrain the use of a relaxation method. For example:

```
Slot: CONSTRAINT(expression)
          UNLESS( (expression relax(expression)))

Due-Date: (?a date)
          CONSTRAINT( (>= current-date))

Work-Period: (?a number)
             IF-NEEDED((due-date FROM SELF) - current-date)

Price:  (?a number)
        CONSTRAINT( (>= 1000) )
              UNLESS (relax( (>= 900) );
                      ((work-period FROM SELF) > 7)
                           relax( (>= 500) ))
```

In this example, the request structure specifies that the due date of a particular request must be greater than or equal to the current date. The work period is calculated as the number of days between the current date and the due date. The price that is specified in the request must be greater than or equal to 1,000. Two relaxation methods are also specified. The first indicates that the price constraint can be relaxed by 100 (unconditionally), while the second indicates that the constraint can be cut in half if the work period is longer than seven days.

- A mechanism for determining whether further negotiation should be applied to satisfy the logical expression associated with a relaxation method. In the example above, the control strategy could attempt to negotiate a longer due date if the current work period is not long enough to warrant the 50% reduction in the price.

- A mechanism for verifying that a relaxation method can be used based on other constraints that the method may affect. A particular relaxation of one constraint may adversely effect the ability to satisfy another. Thus, the control strategy must propagate local relaxations to determine whether other constraints at the same level or at higher levels are invalidated, and if so, whether these constraints propose relaxation methods that can rectify the incompatibilities. In many cases, verifying a potential relaxation of a constraint will require the planning component to interrogate another agent to determine if the proposed relaxation is feasible. For instance, extending the due date in

the example above may prevent the agent that sent the request from meeting other deadlines. However, that agent may also know of ways to relax these deadlines (by rescheduling other tasks) so that the extended due date is acceptable.

A.4.2 Manipulating Relaxation Methods

In the absence of negotiation methods, a knowledge script is not considered applicable unless it is completely compatible with a particular problem-solving request. Knowledge scripts which are partially compatible are ignored, thus limiting substantially the resources that can be used to process the request. When control is extended to facilitate negotiations, introspection becomes significantly more complicated: the selection process can no longer ignore all partially compatible knowledge scripts. These scripts may become applicable if relaxation methods are available and are applied successfully. As a result, introspection requires a form of hypothesize-and-test search: when a script is only partially compatible with a particular request, the control strategy must hypothesize reformulations of the problem based on relaxation methods specified in the script or in the request and then test these hypotheses to ensure they are valid.

Because this process has the potential to become very lengthy and unwieldy (there may be multiple scripts and multiple reformulations that can be applied to each script), we require a mechanism to focus the planning component's attention towards the most promising hypotheses. Such a mechanism would extend the control strategy so that introspection retrieves the knowledge scripts that are either completely or partially compatible to a particular problem-solving request. For example, if the

planning component is searching for applicable plan knowledge, then introspection will retrieve all completely or partially compatible plan scripts. The planning component would then determine whether a completely compatible script is available and, if so, whether it should be used or whether additional effort should be expended to negotiate the removal of the incompatibilities present in one or more of the partially compatible scripts. In order to make this decision, the representation language and control strategy must be extended to provide the following mechanisms:

- A facility for determining the extent of incompatibility between a problem-solving request and a particular knowledge script. This facility must take into account the following factors:
  - The number of constraints that are unsatisfied, both in the knowledge script and in the problem-solving request.
  - The importance of the unsatisfied constraints. This requires the representation language to provide a mechanism for associating an importance measure with each constraint. It should be possible to express this measure as a constant or as a function of the values of particular components in a request.
  - The utility and cost of relaxation methods. This requires the representation language to provide a means of associating measures with each relaxation method indicating the following:
    - A difficulty measure indicating the amount of effort that typically must be expended to apply this method successfully. It should be possible to express this measure as a constant or as a function of the degree to which the associated constraint has been broken.
    - A utility measure which indicates the likelihood of the method succeeding, if applied. Again, this measure should be expressable as a constant or as a function of the degree to which the constraint was broken.
- A mechanism for specifying combination functions which manipulate these measures in order to produce an compatibility measure that reflects the degree of compatibility of a script. Compatibility measures range from 0 (completely incompatible) to 1 (completely compatible). A typical combination function averages the various measures. A combination function also can apply different weightings to the various measures in order to reflect a greater

reliance on a particular measure or combination of measures. The actual combination functions that are used are application dependent (e.g. rules, functions, etc.). The SOLVED-WHEN slot of a problem-solving request and the APPLICABLE-WHEN slot of a knowledge script can define specific combination functions, or the control strategy can employ a general-purpose combination function.

While introspecting, the planning component uses the mechanisms described above to compute a compatibility measure for each script that it examines. As a result, each script will have a compatibility measure and a quality measure (as outlined in section A.3.3) associated with it. Based on these measures, the planning component can perform any of the following actions:

- Select a completely compatible script. The remaining completely compatible and/or partially compatible scripts are placed on an agenda. These scripts can be used if the planning component determines after further processing that the chosen script is not applicable because of lower-level incompatibilities. For example, a request may be completely compatible with the precondition of a particular plan script, but the planning component may subsequently determine that one or more of the planned steps cannot be executed. For example, after instantiating the plan steps using the information in the request, the planning component may determine that the step conflicts with the preconditions imposed by the agent knowledge. Thus the step cannot be distributed. Moreover, a plan can fail if one or more of the agents designated to carry out the plan are unavailable (busy or disabled).

- Prune those scripts with an compatibility measure below a predefined threshold. A low compatibility measure indicates that a script has too many incompatibilities, the incompatibilities are too severe, or the methods available for resolving the incompatibilities require too much effort or provide little chance of succeeding. The script itself can define this threshold (i.e. in the APPLICABLE-WHEN slot of the precondition), or the control strategy can use a general-purpose compatibility threshold.

- Choose a partially compatible script and apply negotiation to determine whether the application of relaxation methods can resolve the existing incompatibilities. The planning component can choose this approach even if a completely compatible script is available. If a script's compatibility measure is high enough and its quality measure is very high, significantly higher than that of any completely compatible scripts, then the planning component can choose to expend additional effort to resolve the incompatiblities so that a higher quality script (plan, task, agent) can be employed.

- The control strategy should use the importance measures associated with the constraints that need to be satisfied, and utility and difficulty measures associated with the various relaxations that can be applied for each constraint in order to determine which negotiations should be applied first. In other words, it should pursue negotiations of the constraint with the highest importance measure using the relaxation methods with the highest utility measure and the lowest difficulty measure.

If the planning component determines that it should perform negotiations to resolve incompatibilities, it must be able to determine when it should conclude negotiations. There are several ways that the planning component can determine that further negotiations are unnecessary or unfruitful:

- The compatibility measure reaches 1, indicating all incompatibilities between the request and the preconditions of the script have been resolved. Negotiation is therefore considered complete.

- The compatibility measure is deemed high enough that the script can be considered applicable. Incompatibilities may still exist, but these incompatibilities involve constraints with very low importance measures. An applicability threshold can be specified in each script defining the level of compatibility that must be present before the script is considered applicable. The impact of the remaining incompatibilities may also be defined in the script. For example, the quality measure of the script can be reduced by some factor of the amount of incompatibility remaining. The planning component can use this information in conjunction with the utility and cost of available relaxation methods to balance the benefits of further negotiation (i.e. obtaining a higher quality measure) with the effort that is required to obtain those benefits.

- The compatibility measure falls below the compatibility threshold. This occurs when relaxation methods fail to resolve incompatibilities in constraints carrying a high importance measure. The planning component must prune this script from the list and backtrack to select another course of action (choose another script from the agenda, or backtrack to a higher-level and select another alternative.

A.4.3 Co-ordinating Efforts During Negotiation

In order to function effectively, a planning component must be able to focus its efforts towards processing the most important problem-solving requests that it receives. One way of achieving this is to assign each request a priority value as outlined in the previous section. The planning component then focuses the majority of its efforts towards high priority requests. Building on this concept, we can provide a planning component with a mechanism that enables it to dictate the maximum and minimum amount of effort that should be expended while processing a particular request. This mechanism requires the following facilities:

- A facility that enables the planning component to dictate the emphasis that is to be placed on quality and compatibility while processing a particular request. The planning component must be able to define a quality and compatibility threshold for each request based on the importance of the request. For example, specifying a low value for the quality threshold and a high value for the compatibility threshold signifies that the planning component is not to emphasize the quality of the methods (scripts) that it can use. Thus, the planning component will focus its efforts towards scripts with the least incompatibilities. This enables the planning component to process low priority requests as expediently as possible. On the other hand, specifying a high value for the quality threshold and a low value for the compatiblity threshold signifies that the planning component should, when necessary, expend additional effort in resolving incompatibilities in order to use the highest quality resources. Thus, the emphasis is placed on quality, not compatibility.

- A facility for specifying request-specific thresholds for the utility and difficulty measures associated with available relaxation methods. This facility enables the planning component to dictate the amount of effort that should be expended during any negotiations involving a particular request. This enables the planning component to constrain the relaxation methods that can be used in negotiating low priority requests. The relaxation methods that are considered applicable must possess a utility measure above the specified utility threshold and a difficulty measure below the difficulty threshold. By specifying these thresholds appropriately, a planning component can focus negotiation efforts towards high quality, low cost relaxation methods for low priority requests. On the other hand, a high priority request may warrant additional efforts in which case the planning component can specify lower utility and difficulty thresholds so that alternative relaxation methods can be used.

A planning component also can use the mechanism described above to indicate the amount of effort that should be expended in fulfilling a particular request that it distributes to another agent. Nevertheless, decentralized control is maintained in that the sender does not dictate how the recipient is to process a request; the sender can, however, indicate the relative importance that it places in the request. The recipient of the request uses this information in determining the processing efforts it should expend to fulfill the request. The recipient typically can expend more or less effort than is indicated by the thresholds specified by the sender, depending on its own workload and the importance that it places in the request.

This facility is particularly useful in co-ordinating the processing efforts within an agency of agents. The managing agent can distribute requests that the agency receives to its subordinates with an indication of the resources that should be used to fulfill the request. The managing agent typically will embodied knowledge describing the relative importance of each request that the agency as a whole can process. It therefore is better equipped to judge the efforts that should be expended in processing particular requests, especially in view of other requests that the agency is currently processing. Therefore, in this scenario the subordinate agents typically will adhere more closely to the thresholds associated with a request so that the agency's resources are used effectively.

A.5 SUMMARY

In this appendix, I have addressed the engineering issues involved in creating an operational system based on the conceptual model of distributed problem solving outlined in Chapter 4. I identified and discussed the representation and control mechanisms required to narrow the gap between the conceptual and computational aspects of the model. A particular application will, however, dictate many of the specific representation and control characteristics that must be implemented. The engineering issues that were examined provide a general-purpose framework for tackling specific applications. AI programming languages such as KEE and KnowledgeCraft can be used to provide a vehicle for rapid prototyping of target applications. Typically, however, extensive tailoring of the representation and control mechanisms provided by these tools will be required to capture the true essence of co-operative problem solving in complex applications.

Appendix B

A SAMPLE DIAGNOSIS SESSION

B.1 CO-OPERATIVE PROBLEM-SOLVING ACTIVITIES

In this appendix, the mechanics of the interactions carried out among the artificial health-care agents described in Chapter 5 in diagnosing a particular patient's health problems are examined. The control functions that the various agents use to co-ordinate co-operative problem solving are discussed as is the meta-level knowledge that the agents use to determine the form and extent of this co-operation.

The problem-solving process begins after a patient has initiated a consultation with a family physician (by sending the physician the problem-solving request in Figure A.1). The physician's planning component performs introspection to select the plan script in Figure A.2 describing the tasks involved in providing the appropriate health care. After a nurse has interrogated the patient to obtain some basic soci-demographic information, the physician arranges to have some basic tests performed based on the patient's (subjective) symptoms by applying the appropriate task knowledge (not shown); some of the tests may require the physician to interrogate the patient in order to obtain a more detailed description of the symptoms. The physician's planning component will integrate the results of the tests (signs), and the patient information received from the nurse, to extend the initial problem-solving request (see Figure A.3).

The diagnosis process is applied to a female patient, 40 years of age, with a lump in her left breast. The results of a mammogram (breast X-ray) that was performed indicate that the tumor is less than 2 cm in size.

```
TYPE:      Request

Operation:    patient care
Object:       Sue Jones
Symptoms:     lump in left breast
```

Figure B.1: Initial Problem-Solving Request Sent by the Patient

```
TYPE:   Plan   (manage patient care)

Applicable-to:
  Operation:  patient care
  Object:     (?a patient)
  Symptoms:   (?a general health problem)

Plan Class:  health-care management

Plan:
     DO (perform-test (symptoms);
         interpret-test-results (signs);
         select-diagnosis (symptoms, signs);
         IF  diagnosis indicates special care THEN
            refer-to-specialist (patient-data);
         ELSE DO
            (select-treatment-plan (diagnosis);
             manage-therapy (treatment-plan));
```

Figure B.2: Snapshot of a Physician's Plan Knowledge for Managing Health Care

The physician's next actions involve interpreting the test results and selecting a diagnosis. Task knowledge maintained by the physician's planning component (not shown) indicates that these tasks can be

```
TYPE:    Request

Operation:    patient care
Object:       Sue Jones
Age:          40
Sex:          female
Symptoms:     lump in left breast
Mammography-results:   (a tumor in left breast,
                        upper inner quadrant,
                        size is < 2 cm)
```

Figure B.3: Updated Problem-Solving Request Describing the Patient

```
TYPE:    Request

Operation:    patient care
Object:       Sue Jones
Age:          40
Sex:          female
Symptoms:     lump in left breast
Mammography-results:   (a tumor in left breast,
                        size is < 2 cm)
Diagnosis:      (?a diagnosis)

Solved-When:  a diagnosis has been determined
```

Figure B.4: Problem-Solving Request Sent to the Physician's Problem-Solving Component

```
TYPE:    Request

Operation:    patient care
Object:       Sue Jones
Age:          40
Sex:          female
Symptoms:     lump in left breast
Mammography-results:   (a tumor in left breast,
                        upper inner quadrant,
                        size is < 2 cm)
Diagnosis:    (Suggestive evidence of breast cancer
               in left breast,
               upper inner quadrant)
Treatment:     specialized care is indicated
```

Figure B.5: Problem-Solving Request with Intermediate Diagnosis

performed by the physician's problem-solving component. As a result, the planning component interrogates its problem-solving component to request that it determine an appropriate diagnosis. The problem-solving request in Figure A.4 is sent to the problem-solving component during this interrogation; the Solved-When slot specifies that a diagnosis is to be generated. The problem-solving component processes the request and responds by returning the request with the diagnosis slot filled in appropriately (see Figure A.5). This information indicates that the problem-solving component was able to suggest the potential presence of a malignant tumor in the patient's breast. As a result, the next step (refer-to-specialist) in the general plan script for managing patient care is executed because the diagnosis indicates that specialized care is required to confirm or disconfirm the diagnosis (see Figure A.2).

The physician's planning component then applies the introspection operator to select task knowledge describing the type of specialized care which is applicable to the patient's tentative diagnosis. The introspection operation uses the characteristics of the patient's health problems, thus in this case the task script describing cancer care is retrieved (see Figure A.6). This task script indicates that an oncologist is capable of providing the required care. In addition, the script specifies a list of subordinates that describe specific requirements that must be met before particular types of cancer problems can be distributed to an oncologist (this knowledge was gathered by the physician from previous interactions with oncologists). The planning component performs additional introspection to select the most applicable subordinate task script. In this example, the script in Figure A.7 is selected since it describes breast cancer care.

```
TYPE:   Task    (cancer care)

Applicable-to:
  Operation:    patient care
  Object:       (?a patient)
  Diagnosis:    (?a cancer problem)

Task Class:     specialized health care

Sub Tasks:      (breast cancer care,
                 lung cancer care,
                 liver cancer care, ...)

Agent Class:    Oncologist
```

Figure B.6: Physician's Task Script Describing Cancer Care

```
TYPE:   Task    (breast cancer care)

Applicable-to:
  Operation:    patient care
  Object:       (?a patient)
  Diagnosis:    (?a breast cancer problem)

Task Class:     cancer care

Task Description:
     Biopsy Report: (?a breast biopsy)
     Location:      (?a breast)

     Solved-When:
         IF   tumor is malignant THEN
              (refer patient to a specialist,
              arrange to have the tumor
              surgically removed)
```

Figure B.7: Physician's Task Script Describing Breast Cancer Care

The physician's planning component interprets this knowledge and determines that a biopsy of the breast must be performed before the patient can be referred to an oncologist; this information is specified by the task description slot of the script, while other information is

inherited from the general task script and the current problem-solving request (e.g. that an oncologist can provide breast cancer care). The physician's planning component again performs introspection to determine how the biopsy can be performed. Figure A.8 illustrates the task script that is selected for this purpose. This script indicates that surgical procedures can be performed by a surgeon (a biopsy is a subordinate type of surgery along with surgical removal of tumors). The planning component co-ordinates additional introspections to select the task script corresponding to biopsies (see Figure A.9) and integrates this information with the current patient data to create a problem-solving request to be sent to the surgeon (see Figure A.10); additional introspections are required to retrieve agent knowledge that can be used to determine a specific surgeon with which to interact.

```
TYPE:  Task  (surgical procedures)

Applicable-to:
  Operation:     (?a surgery)
  Object:        (?a patient)

Sub Tasks:      (biopsy, surgical removal)

Agent Class:    Surgeon
```

Figure B.8: Physician's Task Script Describing Surgical Procedures

Figure A.11 illustrates a portion of the plan knowledge that the surgeon employs to co-ordinate surgical procedures. Information in the Applicable-When slot of this script indicates that surgeries which require less than thirty minutes can be carried out the same day, while more extensive surgeries require at least two weeks notice. Information

```
TYPE:   Task   (biopsy)

Applicable-to:
  Operation:    (?a biopsy)
  Object:       (?a patient)

Task Class:     surgical procedures

Task Description:
     Location:       (?a body location)
     Length of Procedure:   < 30 minutes
```

Figure B.9: Physician's Task Script Describing a Biopsy Procedure

```
TYPE:     Request

Operation:     breast biopsy
Object:        Sue Jones
Age:           40
Sex:           female
Location:      left breast
Length of procedure:  < 30 minutes
Biopsy Report: (?a pathology report)

Solved-When:  a pathology report has been generated
```

Figure B.10: Problem-Solving Request Sent to a Surgeon to Request a Breast Biopsy

in the Applicable-When slot also indicates that the notification period can be adjusted if the surgeon's schedule permits or the patient's status is life threatening. The biopsy request that the surgeon receives from the physician satisfies the prerequisite for performing the surgery immediately therefore the surgeon's planning component notifies the physician that the procedure can be performed immediately (the problem-solving request is sent back to the physician with the notification period filled in appropriately).

```
TYPE:    Plan   (manage surgical procedures)

Applicable-to:
  Operation:  (?a surgery)
  Object:     (?a patient)
  Location:   (?a body location)
  Length of Procedure:  (?a time period)

  Notification Period:  (?a time period)
       IF-NEEDED (IF length of procedure < 30 minutes
                  THEN  notification period = immediately
                  ELSE  notification period = 2 weeks)

        RELAX (negotiate an acceptable period
               based on the patient's health status
               and the schedule of surgeries)


Plan:   .....
```

Figure B.11: Snapshot of a Surgeon's Plan Knowledge for Managing Surgical Procedures

Once the surgery has been completed and a pathologist has analyzed the specimen, the pathology report indicating whether or not the tumor is malignant is sent to the physician. The physician applies the information in the Solved-When slot of the task description structure in the task script in Figure A.7 to determine the next course of action. Since the tumor has been found to be malignant, the information indicates that the physician should interrogate an oncologist to have him take over the diagnosis and treatment tasks. Figure A.12 illustrates the problem-solving request that the physician's planning component sends to the oncologist; a specific oncologist is selected by applying introspection to select the appropriate agent knowledge (not shown).

```
TYPE:    Request

Operation:    cancer care
Object:       Sue Jones
Age:          40
Sex:          female
Symptoms:     lump in left breast
Mammography-results:   (a tumor in left breast,
                        upper inner quadrant,
                        size is < 2 cm)
Diagnosis:    (Suggestive evidence of breast cancer
               in left breast,
               upper inner quadrant)
Treatment:    specialized care is indicated
Biopsy Report:   Malignant
Surgical Procedure:  removal is scheduled

Solved-When:  apply diagnosis and treatment procedures
```

Figure B.12: Problem-Solving Request Sent to Oncologist by the Physician

The oncologist applies the appropriate control functions to select plan and task knowledge that describe the tasks involved in co-ordinating cancer care, including detailed diagnosis of the stage of the breast cancer and the establishment and management of an appropriate treatment plan.

In addition to referring the patient to an oncologist, the information in the Solved-When slot specifies that the physician should arrange to have the tumor surgically removed. The physician's planning component applies introspection to select the task script in Figure A.13 describing surgical removal tasks (this is the other task script subordinate to the surgical procedures script in Figure A.8). The planning component uses this knowledge and the patient data to send to a surgeon the problem-solving request in Figure A.14 indicating that the

tumor in the patient's breast should be removed; the physician may have to interrogate the oncologist to determine the extent of the mastectomy (i.e. partial or total).

```
TYPE:   Task   (surgical removal)

Applicable-to:
  Operation:    (?a removal operation)
  Object:       (?a patient)
  Notification Period:  (?a time period)

  Applicable-When:  IF notification period is unacceptable
                    THEN  initiate negotiation or interrogate
                          another agent

Task Class:    surgical procedures

Task Description:
     Location:       (?a body location)
     Extent:         (?a description)
     Length of Procedure:    > 30 minutes
```

Figure B.13: Physician's Task Script Describing Surgical Removal Procedures

```
TYPE:    Request

Operation:    Mastectomy
Object:       Sue Jones
Age:          40
Sex:          female
Location:     left breast
Length of procedure:  > 30 minutes
Extent:         partial mastectomy
Notification Period:  (?a time period)
```

Figure B.14: Problem-Solving Request Sent to a Surgeon to Request Removal of the Tumor

The Surgeon's planning component performs introspection to select the plan script in Figure A.11. Because the length of the surgical procedure specified in the request is greater than thirty minutes, the surgeon will respond by sending the physician the problem-solving request structure with a slot indicating the notification period is two weeks (default time for more extensive procedures). If necessary, the physician's planning component can co-ordinate negotiations with the surgeon to have the notification period shortened (the information in the Applicable-When slot of the surgical removals task script indicates that the notification period can be negotiated). If the physician initiates negotiations over the notification period, the surgeon's planning component interprets the information in the plan script for managing surgical procedures in order to determine that the notification time period can be adjusted if the patient's health status is life threatening or the schedule of surgery permits moving the date of surgery forward.

If the surgeon is able to shorten the notification period, its planning component negotiates with the physician to determine whether the new period is acceptable. The physician's planning component can indicate that the period is acceptable or can co-ordinate additional negotiations to try to shorten the period further. In some instances, the surgeon may not be able to modify the notification period in which case the physician will have to accept the time period or interrogate other surgeons.

B.2 SUMMARY

The sample diagnosis session presented in this appendix illustrates the knowledge-based processing that the agents perform to co-ordinate co-operative interactions necessary to diagnose health problems. It is important to note that the knowledge used in the example is the communication knowledge of each specialist, it does not include the problem-solving knowledge of the specialists. The fundamental principle that the example illustrates is that all interactions are co-ordinated using explicit knowledge which is stratified into context-specific knowledge sources rather than grouped into a large and cumbersome procedural implementation of a co-operative problem-solving algorithm. The co-operative problem-solving capabilities of each agent are therefore explicit and modifiable which results in a more understandable and flexible means of co-ordinating co-operative problem solving.

BIBLIOGRAPHY

[Agha 86]
Agha, G. **Actors: A Model of Concurrent Computation in Distributed Systems,** MIT Press, London, 1986.

[Aikins 83]
Aikins, J. S. "Prototypical Knowledge for Expert Systems", **Artificial Intelligence (20),** 1983, pp. 163-210.

[Allison 71] Allison, G. **Essence of Decision: Explaining the Cuban Missile Crisis,** Little, Brown, and Company, Boston, 1971.

[Axelrod 84]
Axelrod, R. **The Evolution of Cooperation,** Basic Books, New York, 1984.

[Barr 81]
Barr, A., Feigenbaum, E.A. **The Handbook of AI Volume 1,** William Kaufmann, Inc., Los Altos, 1981.

[Barr 81a]
Barr, A., Fiegenbaum, E.A. **The Handbook of AI Volume 2,** William Kaufmann, Inc., Los Altos, 1981.

[Beck 85]
Beck, W. S. (editor) **Hematology,** MIT Press, Cambridge, 1985.

[Bobrow 75]
Bobrow, D., Collins, A. (eds.) **Representation and Understanding: Studies in Cognitive Science,** Academic Press, New York, 1975.

[Bobrow 86]
Bobrow, D., Stefik, M. "Object-Oriented Programming: Themes and Variations", **The AI Magazine,** Winter, 1986, pp. 40-62.

[Brachman 85]
Brachman, R., Pigman-Gilbert, V., Levesque, H. "An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of Krypton", **Proceedings of the IJCAI,** 1985, pp. 532-539.

[Brodie 84]
Brodie, M., Mylopoulos, J., Schmidt, J. **On Conceptual Modelling,** Springer-Verlag, New York, 1984.

[Brownston 85]
Brownston, L., Farrell, R., Kant, E., Martin, N. **Programming Expert Systems in OPS5,** Addison-Wesley, Don Mills, 1985.

[Buchanan 84]
Buchanan, B., Shortliffe, E. **Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project,** Addison-Wesley, Don Mills, 1984.

[Burkowski 88]
Burkowski, F., Cormack, G. "Distributed Synchronous Process Communication", **Congressus Numerantium,** Volume 62, May 1988, pp. 125-132.

[Cammarata 83]
Cammarata, S., McArthur, D., Steeb, R. "Strategies of Cooperation in Distributed Problem Solving", **Proceedings of the IJCAI,** 1983, pp. 767-770.

[Cercone 83]
Cercone, N., McCalla, G. "Approaches to Knowledge Representation", **IEEE Computer,** October 1983, pp. 12-18.

[Chang 86] Chang, E. "Participant Systems", **Future Computing Systems,** vol. 1(3), 1986.

[Chang 88]
Chang, E. "Report on the 1988 Distributed Artificial Intelligence Workshop", **Canadian Artificial Intelligence,** July 1988, pp. 32-34.

[Charniak 85]
Charniak, E., McDermott, D. **Introduction to Artificial Intelligence,** Addison-Wesley, Don Mills, 1985.

[Cohen 72] Cohen, M., March, J., Olsen, J. "A Garbage Can Model of Organizational Choice", **Administrative Studies Quarterly,** (17:1), 1972, pp 1-25.

[Corkill 79]
Corkill, D. "Hierarchical Planning in a Distributed Environment", **Proceedings of the IJCAI,** 1979, pp. 168-175.

[Corkill 82]
Corkill, D., Lesser, V., Hudlicka, E. "Unifying Data-Directed and Goal-Directed Control: An Example and Experiments", **Proceedings of the AAAI,** 1982, pp. 143-147.

[Corkill 83]
Corkill, D., Lesser, V. "The Use of Meta-Level Control for Coordination in a Distributed Problem-Solving Network", **Proceedings of the IJCAI,** 1983, pp. 748-756.

[Cyert 63] Cyert, R., March, J. **A Behavioral Theory of the Firm,** Prentice-Hall, Englewood Cliffs, New Jersey, 1963.

[Davis 80]
Davis, R. "Report on the Second Workshop on Distributed Artificial Intelligence", **SIGART Newsletter (73),** pp. 43-52.

[Davis 82]
Davis, R., Lenat, D. **Knowledge-Based Systems in Artificial Intelligence,** McGraw-Hill, Toronto, 1982.

[Davis 83]
Davis, R., Smith, R. "Negotiation as a Metaphor for Distributed Problem Solving", **Artificial Intelligence (20),** 1983, pp. 63-109.

[Devita 85]
Devita, V., Hellman, S., Rosenberg, S. **Cancer Principles and Practice of Oncology,** (2nd Edition), J.B. Lippincott Company, Philadelphia, 1985.

[Dreyfus 86]
Dreyfus, H., Dreyfus, S. "Why Expert Systems Do Not Exhibit Expertise", **IEEE Expert,** Summer 1986, pp. 86-90.

[Durfee 85]
Durfee, E.H., Lesser, V., Corkill, D. "Increasing Coherence in a Distributed Problem-Solving Network", **Proceedings of the IJCAI,** 1985, pp. 1025-1030.

[Eliot 87]
Eliot, L. "Neural Networks, Part 1: What are They and Why is Everybody so Interested in Them Now?", **IEEE Expert,** Winter 1987, pp. 10-14.

[Ensor 85]
Ensor, J. R., Gabbe, J. "Transactional Blackboards", **Proceedings of the IJCAI,** 1985, pp. 340-344.

[Erman 75]
Erman, L., Lesser, V. "A Multi-Level Organization for Problem Solving Using Many, Diverse, Cooperating Sources of Knowledge", **Proceedings of the IJCAI,** 1975, pp. 483-490.

[Eswaran 76]
Eswaran, K. P. "The Notions of Consistency and Predicate Locks in a Database System", **Communications of the ACM,** November 1976, pp. 624-633.

[Evans 87]
Evans, M. "Knowledge Representation: The Crux of AI", **Congressus Numerantium,** Volume 57, June 1987, pp. 63-87.

[Evans 87a]
Evans, M., Scuse, D.H. "A Multi-Level Model For Knowledge Representation", **Proceedings of the CIPS Congress'87,** pp. 367-375.

[Fikes 85]
Fikes, R., Kehler, T. "The Role of Frame-Based Representation in Reasoning", **Communications of the ACM Vol. 29 No. 2,** September 1985, pp. 904-920.

[Fischler 87]
Fischler, M.A., Firschein, O. **Intelligence: The Eye, the Brain, and the Computer,** Addison-Wesley, Don Mills, 1987.

[Fowler 64]
Fowler, H. W., Fowler, F. G. (eds.) **The Concise OXFORD DICTIONARY of Current English,** Oxford University Press, Oxford, 1964.

[Fox 84] Fox, M., Smith, S. "ISIS: A Knowledge-Based System for Factory Scheduling", **International Journal of Expert Systems,** Vol. 1, 1984, pp. 25-49.

[Fox 87]
Fox, M. "Beyond the Knowledge Level", **Conference on Expert Database Systems,** 1987, pp. 455-463.

[Gelernter 87]
Gelernter, D. "Programming for Advanced Computing", **Scientific American,** October 1987, pp. 91-99.

[Genesereth 86].
Genesereth, M., Ginsberg, M., Rosenschein, J. "Cooperation Without Communication", **The Proceedings of the AAAI,** 1986, pp. 51-57.

[Georgeff 83]
Georgeff, M. "Communication and Interaction in Multi-Agent Planning", **Proceedings of the AAAI,** 1983, pp. 125-129.

[Georgeff 84]
Georgeff, M. "A Theory of Action For Multi-Agent Planning", **Proceedings of the AAAI,** 1984, pp. 121-125.

[Ginsberg 87]
Ginsberg, M. "Decision Procedures", in **Distributed Artificial Intelligence,** Morgan Kaufmann Publishers, California, 1987.

[Hayes-Roth 83]
Hayes-Roth, F., Waterman, D., Lenat, D. **Building Expert Systems,** Addison-Wesley, Don Mills, 1983.

[Hecht-Nielsen 88]
Hecht-Nielsen, R. "Neurocomputing: Picking the Human Brain", **IEEE Spectrum,** March 1988, pp. 36-41.

[Hewitt 73]
Hewitt, C., Steiger, R. "A Universal Modular ACTOR Formalism for Artificial Intelligence", **Proceedings of the IJCAI,** 1973, pp. 235-245.

[Hewitt 77]
Hewitt, C. "Control Structure as Patterns of Passing Messages", **Artificial Intelligence,** Volume 8, 1977, pp. 323-363.

[Hewitt 84]
Hewitt, C., de Jong, P. "Open Systems", in [Brodie 84], pp. 147-164.

[Hewitt 85]
Hewitt, C. "The Challenge of Open Systems", **Byte Magazine,** April 1985, pp. 223-242.

[Hillis 85]
Hillis, W. **The Connection Machine,** MIT Press, Cambridge, 1985.

[Hofstadter 79]
Hofstadter, D. **Godel, Escher, Bach: An Eternal Golden Braid,** Vintage Books, New York, 1979.

[Huhns 87] Huhns, M. **Distributed Artificial Intelligence,** Morgan Kaufmann Publishers, Inc., California, 1987.

[Jackson 86]
Jackson, P. **Introduction to Expert Systems,** Addison-Wesley, Don Mills, 1986.

[Jagnnathan 86]
Jagannathan, V., Dodhiawala, R. "Distributed Artificial Intelligence: An Annotated Bibliography", **SIGART Newsletter No. 95,** January 1986, pp. 44-56.

[Keller 87]
Keller, R. **Expert System Technology: Development and Application,** Prentice-Hall, New Jersey, 1987.

[Kornfeld 79]
Kornfeld, N. "ETHER -- A Parallel Problem-Solving System", **Proceedings of the IJCAI,** 1979, pp. 490-492.

[Kuo 81]
Kuo, F. (Editor) **Protocols and Techniques for Data Communication Networks,** Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[Lesser 80]
Lesser, V. "Cooperative Distributed Problem Solving and Organizational Self-Design", **Sigart Newsletter,** October 1980, p. 46.

[Lesser 79]
Lesser, V., Corkill, D. "The Application of Artificial Intelligence Techniques to Cooperative Distributed Processing", **Proceedings of the IJCAI,** 1979, pp. 537-540.

[Lesser 81]
Lesser, V., Corkill, D. "Functionally-Accurate, Co-operative Distributed Systems", **IEEE Transactions on Man, Systems, and Cybernetics,** SMC-11(1), January 1981, pp. 81-96.

[Lesser 83]
Lesser, V., Corkill, D. "The Distributed Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks", **The AI Magazine,** Fall 1983, pp. 15-33.

[Levesque 84]
Levesque, H. "A Fundamental Tradeoff in Knowledge Representation and REasoning", **Proceedings of CSCSI-84,** 1984, pp. 141-152.

[Liebowitz 88]
Liebowitz, J. **Introduction to Expert Systems,** Mitchell Publishing, California, 1988.

[McDermott 81]
McDermott, J. "R1: The Formative Years", **The AI Magazine,** Vol. 2 No. 2, 1981, pp. 21-29.

[Minsky 86]
Minksy, M. **The Society of Mind,** Simon and Schuster, New York, 1986.

[Minsky 88]
Minksy, M., Papert, S. **Perceptrons,** MIT Press, Cambridge, 1988.

[Mittal 86] Mittal, S., Dym, C., Jorjaria, M. "PRIDE: An Expert System for Design of Paper Handling Systems", **COMPUTER,** July 1986, pp. 102-114.

[Moulin 87]
Moulin, B. "A Knowledge Engineering Perspective of System Modelling and Design", **Proceedings of the CIPS Congress'87,** pp. 377-384.

[Newell 81]
Newell, A. "The Knowledge level", **The AI Magazine,** Vol. 2 No. 2, 1981, pp. 1-20.

[Newell 76]
Newell, A., Simon, H. "Computer Science as Empirical Inquiry: Symbols and Search", **Communication of the ACM,** March 1976, pp. 113-126.

[Nii 86]
Nii, H. P., "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures", **The AI Magazine,** Summer 1986, pp. 38-53.

[Nilsson 80]
Nilsson, N. **Principles of Artificial Intelligence,** Tioga Publishing, Palo Alto, 1980.

[Old 88]
Old, L. J. "Tumor Necrosis Factor", **Scientific American,** May 1988, pp. 59-75.

[O'Shea 87]
O'Shea, T., Self, J., Thomas, G., (eds.) **Intelligent Knowledge-Based Systems: An Introduction,** Harper & Row, London, 1987.

[Pearl 84]
Pearl, J. **Heuristics,** Addison-Wesley, Don Mills, 1984.

[Quillian 86]
Quillian, M. R. **Semantic Memory,** (Ed. Minsky), 1968, pp. 227-270.

[Rauch-Hindin 88]
Rauch-Hindin, W. **A Guide to Commercial Artificial Intelligence,** Prentice-Hall, New Jersey, 1988.

[Restak 79]
Restak, R. **The Brain: the Last Frontier,** Warner Books, New York, 1979.

[Rich 83]
Rich, E. **Artificial Intelligence,** McGraw Hill, Toronto, 1983.

[Rich 84]
Rich, E. "Natural Language Interfaces", **IEEE Computer,** September 1984, pp. 39-47.

[Rosenschein 85]
Rosenschein, J. **Rational Interaction: Cooperation Among Intelligent Agents,** Ph.D. thesis, Standford University, 1985.

[Rosenthal 87]
Rosenthal, S., Carignan, J. R., Smith, B. D. **Medical Care of the Cancer Patient,** W.B. Saunders Company, Toronto, 1987.

[Rubin 83]
Rubin, P. (editor) **Clinical Oncology for Medical Students and Physicians,** American Cancer Society, 1983.

[Rumelhart 86]
Rumelhart, D., McClelland, J. **Parallel Distributed Processing,** MIT Press, Cambridge, 1986.

[Scuse 88]
Scuse, D. H. **"Artificial Intelligence in Medicine and the Mantioba Cancer Treatment and Research Foundation Programme",** Technical Report, Department of Computer Science, University of Manitoba, 1988.

[Shapiro 87]
Shapiro, S. (editor) **Encyclopedia of Artificial Intelligence,** Volumes 1-2, John Wiley & Sons, Toronto, 1987.

[Singer 86]
Singer, J **1986 Annual Report: The Manitoba Cancer Treatment and Research Foundation,** Manitoba Cancer Foundation, Winnipeg, 1986.

[Smith 79]
Smith, R. "A Framework for Distributed Problem Solving", **Proceedings of the IJCAI,** 1979, pp. 836-841.

[Smith 78]
Smith, R., Davis, R. "Distributed Problem Solving: The Contract Net Approach", **Proceedings of the Second National Conference of the Canadian Society for Computational Studies of Intelligence,** 1978.

[Sridharan 87]
Sridharan, N. "1986 Workshop on Distributed Artificial Intelligence", **The AI Magazine,** Fall 1987, p. 75-85.

[Stanfill 86]
Stanfill, C., Waltz, D. "Towards Memory-Based Reasoning", **Communications of the ACM,** December 1986, pp. 1213-1228.

[Sternberg 82]
Sternberg, R. (editor) **Handbook of Human Intelligence,** Cambridge University Press, New York, 1982.

[Tank 87]
Tank, D., Hopfield, J. "Collective Computation in Neuronlike Circuits", **Scientific American,** December 1987, pp. 104-114.

[Tsotsos 85]
Tsotsos, J. "Expert Systems Overview: Where Does the Phrase Fit?", **CIPS Review,** September/October 1985, pp. 12-14.

[Turban 88]
Turban, E. **Decision Support and Expert Systems: Managerial Perspectives,** MacMillan Publishing Company, New York, 1988.

[Wasserman 88]
Wasserman, P., Schwartz, T. "Neural Networks, Part 2: What are They and Why is Everybody so Interested in Them Now?", **IEEE Expert,** Spring 1988, pp. 10-16.

[Waterman 86]
Waterman, D. **A Guide to Expert Systems,** Addison-Wesley, Don Mills, 1986.

[Winston 84]
Winston, P. **Artificial Intelligence,** Addison-Wesley, Don Mills, 1984.

LIST OF FIGURES

Figure		page
1.1.	The Hijacker Example	6
1.2.	A Corporate Hierarchy	9
2.1.	AI Model of Problem Solving	17
2.2.	The Waterjug Problem	19
2.3.	Algorithm for Solving the Waterjug Problem	20
2.4.	Nondeterministic Solution to the Waterjug Problem	20
2.5.	An OR Graph	24
2.6.	An AND Tree and an AND/OR Tree	24
2.7.	The 8-Puzzle Problem	28
2.8.	Representation in Predicate Logic	38
2.9.	Sample Rules	40
2.10.	A Semantic Network	43
2.11.	A Frame-Based Network	46
2.12.	Knowledge System Embodying a Metaknowledge Component	51
2.13.	Sample Meta-Rules For Justifying Actions	52
2.14.	Sample Meta-Rules for Controlling Reasoning	53
3.1.	Hierarchy of Agents and Agencies	75
3.2.	Contract Net Announcement, Bid, and Award Messages [Smith 78]	83
4.1.	A Basic Agent in a Distributed Problem-Solving System	97
4.2.	An Agency of Agents	98
4.3.	Communications Link in the Components	101

4.4. Structure of a Planning Component 102

4.5. Types of Knowledge in a Planning Component 105

4.6. Frame Structure for Representing Problem Descriptions . . . 106

4.7. Frame Structure for Representing Plan Scripts 110

4.8. Frame Structure for Representing Task Scripts 114

4.9. Frame Structure for Representing Agent Scripts 116

4.10. Problem-Solving Request for a House Purchase 119

4.11. Partial Description of the Real-Estate Agent's Plan Knowledge 120

4.12. Organization of an Activity Blackboard 122

4.13. Components of the Control Strategy 126

5.1. Organization of Agents Involved in Cancer Diagnosis and Treatment 144

5.2. Basic Cancer Diagnosis and Treatment Activities 148

5.3. Snapshot of Patient's Plan Knowledge Hierarchy 157

5.4. Snapshot of a Patient's Plan Script for Treating Severe Health Problems 157

5.5. Snapshot of a Patient's Task Knowledge For Obtaining Professional Health Care 159

5.6. Snapshot of a Patient's Agent Knowledge Describing Health-Care Professionals 160

5.7. Snapshot of a Physician's Plan Script for Co-ordinating Health Care 162

5.8. Snapshot of a Physician's Task Knowledge 164

5.9. Snapshot of a Physician's Task Knowledge 166

5.10. Snapshot of a Physician's Agent Knowledge 167

5.11. Snapshot of an Oncologist's Plan Script for Co-ordinating Cancer Care 169

5.12. Snapshot of an Oncologist's Task Knowledge 171

5.13. Snapshot of an Oncologist's Agent Knowledge 172

5.14. Snapshot of an Oncologist's Plan Knowledge Hierarchy 174

5.15. Snapshot of an Oncologist's Plan Script for Selecting a Standard Treatment Plan 174

5.16. Snapshot of an Oncologist's Task Knowledge 176

5.17. Snapshot of an Oncologist's Plan Knowledge Hierarchy for Administering Therapies 178

5.18. Snapshot of an Oncologist's Plan Script for Co-ordinating Chemotherapy 179

5.19. Snapshot of an Oncologist's Task Knowledge 180

5.20. Snapshot of an Oncologist's Task Knowledge 181

5.21. Snapshot of an Oncology Nurse's Task Knowledge 182

A.1. Sample Generic Frames, Specializations, and Instances . . . 216

A.2. Sample Plan and Task Script 220

A.3. Sample Agent and Agent-Specific Task Scripts 221

B.1. Initial Problem-Solving Request Sent by the Patient 241

B.2. Snapshot of a Physician's Plan Knowledge for Managing Health Care 241

B.3. Updated Problem-Solving Request Describing the Patient . . . 242

B.4. Problem-Solving Request Sent to the Physician's Problem-Solving Component 242

B.5. Problem-Solving Request with Intermediate Diagnosis 242

B.6. Physician's Task Script Describing Cancer Care 244

B.7. Physician's Task Script Describing Breast Cancer Care . . . 244

B.8. Physician's Task Script Describing Surgical Procedures . . . 245

B.9. Physician's Task Script Describing a Biopsy Procedure . . . 246

B.10. Problem-Solving Request Sent to a Surgeon to Request a Breast Biopsy 246

B.11. Snapshot of a Surgeon's Plan Knowledge for Managing Surgical Procedures 247

B.12. Problem-Solving Request Sent to Oncologist by the Physician 248

B.13. Physician's Task Script Describing Surgical Removal Procedures 249

B.14. Problem-Solving Request Sent to a Surgeon to Request Removal of the Tumor 249