National Library of Canada

Bibliothèque nationale du Canada

Acquisitions and Bibliographic Services Branch

Direction des acquisitions et des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Cooperative Problem Solving
and the Game of Distributed Blackbox

Kristina Pitula

A Thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

June 1990

© Kristina Pitula, 1990

National Library of Canada

Acquisitions and Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada

Direction des acquisitions et des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

THE AUTHOR HAS GRANTED AN IRREVOCABLE NON-EXCLUSIVE LICENCE ALLOWING THE NATIONAL LIBRARY OF CANADA TO REPRODUCE, LOAN, DISTRIBUTE OR SELL COPIES OF HIS/HER THESIS BY ANY MEANS AND IN ANY FORM OR FORMAT, MAKING THIS THESIS AVAILABLE TO INTERESTED PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE IRREVOCABLE ET NON EXCLUSIVE PERMETTANT A LA BIBLIOTHEQUE NATIONALE DU CANADA DE REPRODUIRE, PRETER, DISTRIBUER OU VENDRE DES COPIES DE SA THESE DE QUELQUE MANIERE ET SOUS QUELQUE FORME QUE CE SOIT POUR METTRE DES EXEMPLAIRES DE CETTE THESE A LA DISPOSITION DES PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP OF THE COPYRIGHT IN HIS/HER THESIS. NEITHER THE THESIS NOR SUBSTANTIAL EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT HIS/HER PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE DU DROIT D'AUTEUR QUI PROTEGE SA THESE. NI LA THESE NI DES EXTRAITS SUBSTANTIELS DE CELLE-CI NE DOIVENT ETRE IMPRIMES OU AUTREMENT REPRODUITS SANS SON AUTORISATION.

ISBN 0-315-97704-3

Canada

Abstract

Cooperative Problem Solving and the Game of Distributed Blackbox

Kristina Pitula

Distributed Problem Solving (DPS) consists of a set of distributed expert systems or agents, that cooperate with each other to solve a single, complex problem. Cooperative problem solving studies how agents in a DPS system coordinate their activity by pursuing a common goal, and thus cooperate with each other. The difficulty of studying coordination and cooperation in an abstract context make experimental work an essential aspect of DPS research. In this thesis we examine the issues involved in applying a cooperative problem solving approach through the agency of a game called Distributed Blackbox (DBB). DBB's suitability as a testbed for experimental research is brought out through an extensive comparison with the well known Distributed Vehicle Monitoring Testbed (DVMT). The comparison reveals that DBB is much simpler than DVMT, but provides a rich variety of problem situations in which cooperation can be studied.

As a forerunner of DBB, we have designed, implemented, and evaluated a single expert system to solve the game of Blackbox. In the course of the design, we have developed an appropriate knowledge representation for the problem. We analyse the DBB problem, and propose a design for a DBB prototype wherein agents coordinate their activity in a way analogous to that used by people in human organisations. The design incorporates a number of other features to facilitate experiments. The characteristics of the DBB problem and the features of the proposed design, make DDB appear eminently suitable for studying many current issues in Distributed Problem Solving.

Acknowledgements

I would like to thank everyone who aided, abetted, and otherwise contributed in getting this production out. I would like to give particular thanks to Dr. Radakrishnan, for his patient guidance, pertinent questions, and for reading all the first drafts. To John, who had the flush of inspiration, and is still hot on the trail. To Carol and Thomas, who both got swept into DBB with no forewarning. To Cliff, who was there from the beginning, passed through all the intermediate turmoil, and who may still have to live with Blackbox. To Chef Mbage and Professor Hutton, who explained why, one crocodile who got the whole thing rolling, and Mary, who says she wants to read this for entertainment??? To the future Dr. B., who enjoyed playing the game (a personalised copy in colour), and to Capt. P, who learnt everything in a much tougher school. To the family, who kept up the chicken soup when the wolves were howling, (am I educated now?). And a special *ACK* to Fre, who lived with all this for the past few years even though it wasn't in the contract.

"The researches of many commentators have already thrown much darkness on this subject, and it is probable that, if they continue, we shall soon know nothing at all about it."

Mark Twain

Contents

List of Figures and Tables

Chapter 1
An Expert System Approach to Problem Solving

Artificial Intelligence is concerned with creating models of the human reasoning process that can be employed to solve problems using computers. The problems considered generally involve searches over very large search spaces. The combinatorial explosion that can occur within even the most seemingly trivial search space has lead to the development of special techniques to handle these searches, based on the methods people employ when reasoning in similar situations. These methods rely on knowledge about the problem domain, and domain related strategies to guide their searches. The methods generally succeed, but they may sometimes fail. This is a consequence of the ill-defined nature of the problems, and the incompleteness of the reasoning models.

The symbolic knowledge employed by these systems is often difficult to formulate using conventional approaches. This has lead to the development of specialised AI languages, and specialised software environments known as *expert systems*. Both provide facilities for representing and manipulating symbolic knowledge in a non-procedural fashion. Rather than using mathematical computations and relationships, they use the associations between knowledge and action to transform the current state of the problem solver into a state which "appears" closer to that of the desired solution. The resulting systems differ significantly from non-AI systems in the way they behave, in the way they are described, in the type of problems they address, and in the way their performance is evaluated.

1.1 Outline of the Thesis

In this thesis we examine how an expert system approach can be applied in a distributed environment in order to increase the complexity of the problems that can be solved. Such Distributed Problem Solving (DPS) systems would consist of a set of semi-autonomous agents that cooperate with each other to solve a single problem. The difficulty of studying cooperation in a purely theoretical context makes experimental work an essential aspect of DPS research. We propose the game of Distributed Blackbox (DBB) as a suitable testbed for experimenting with many current DPS issues. DBB's advantages are brought out through an extensive comparison with the well known Distributed Vehicle Monitoring Testbed (DVMT). The comparison reveals DBB's relative simplicity, and a richness of problem situations in which cooperation can be studied. A single expert system to solve the game of Blackbox is presented. A design for a DBB prototype is then proposed. These topics are covered in the following chapters.

Chapter one briefly describes how an expert system operates, the different ways in which knowledge can be represented, the reasoning strategies with which an expert system handles uncertainty, and the work that the development of an expert system entails. The latter covers the acquisition of knowledge, the development of reasoning strategies, and performance evaluation.

Chapter two explores the concept of cooperative problem solving, and draws an analogy to the paradigms that people use when reasoning about complex problems. The roles of *control*, *coordination*, *cooperation*, and *communication* within a DPS system are considered, with an emphasis on cooperation. The various coordination schemes that have been proposed for DPS systems are briefly surveyed, and a scheme in which DPS agents employ "organisational" knowledge is then examined more closely.

Chapter three describes the game of DBB, and the reasons which make DBB appear a suitable problem for DPS research. DBB's advantages and the DPS issues for which it is suitable are brought out by comparing DBB to DVMT, and determining DBB's similarities and differences.

Chapter four describes the game of Blackbox, and the expert system that has been implemented to solve the problem. The description covers the representation of knowledge, the solution process, the facilities for performance evaluation, and the actual performance measurements obtained by the Blackbox expert system.

The DBB problem is analysed in chapter five. A design for a DBB prototype is proposed wherein agents employ "organisational" knowledge to coordinate their activity and cooperate with each other. The design includes a number of other features that promise to facilitate development and experimentation with DBB. Chapter six contains the conclusion.

1.2 Expert Systems

Expert systems are a distinct class among AI systems because they rely on domain knowledge in addition to general purpose inference techniques. Rather than follow a fixed processing sequence that systematically examines the entire search space, the "solution process" varies according to the current state of knowledge, choosing those actions which appear the most promising. It is here that domain knowledge and human expertise enter the picture, as the most appropriate action to take may not be apparent from the available information. Uncertainty arises because the problem is ill-defined, and the knowledge to solve it is incomplete. The ability to vary the solution process according to the state of knowledge, and to make decisions in spite of uncertainty about that state, distinguishes expert systems from other approaches. A consequence of this ability is that expert systems have highly variable and unpredictable processing patterns, related to the amount and type of uncertainty in the problem.

Uncertainty is an inherent characteristic of the problems addressed. The problems may be either *ill-defined*, or *well-defined* but intractable due to processing complexity. A problem is ill-defined because of: (a) inadequate or inaccurate problem descriptions, (b) incomplete knowledge about how to derive solutions, and (c) possibly no way of determining whether the solutions actually found are acceptable. Medical diagnosis is an application domain that encompasses all three. In the case of well-defined but intractable problems, the excessive processing requirements of the solution model make the model effectively useless. The travelling salesman problem is a classic example of this.

How this uncertainty is dealt with is highly dependent upon the nature of the problem, and what end users consider an acceptable solution. Many problems will have more than one path to arrive at a solution. Certain problems will have more than one solution, with no means of determining which is the optimal one. Some way of approximating which solution or solution path is the best is necessary in all cases. This introduces inaccuracy at various points in both the solution process and in the solution itself. Reducing inaccuracy will increase processing complexity. As a consequence, acceptability generally involves a tradeoff between timeliness and accuracy.

How successful the solutions are will depend upon how well the developer's definition of acceptability corresponds to the user's expectations. This is highly dependent on both the application domain, and its intended use. Given that the number of problem instances is generally open-ended, there is no a priori way of predicting the optimal solution path for all cases. The reasoning model used by the expert system may only be suitable for a subset of problems within the problem domain. Establishing how well the system's performance fulfills the user's expectations is part of evaluation. In this case, evaluation must not only verify the logical correctness of the implementation, but must also establish that the chosen subset of problems is representative of the domain, and that the system will perform consistently for all problems that lie within this subset.

1.3 The Structure of an Expert System

An expert system consists of three basic components: a *knowledge base*, a *working memory*, and an *inference engine*, that operate in the following way. The knowledge base contains the knowledge pertaining to the problem domain. The working memory contains a description of the current problem solving state. The inference engine drives the reasoning process. It analyses working memory, and applies the corresponding knowledge in the knowledge base. The application of knowledge transforms the contents of working memory, leading to its reanalysis, and the application of more knowledge, or termination. The state of the problem at any stage determines the set of applicable knowledge. The structure of the domain knowledge, combined with the order of access defined by the inference engine, determines what specific knowledge is applied at any given point during the reasoning process.

The basic operating structure of an expert system is in itself highly procedural. All non-procedural aspects to its reasoning are supplied entirely by the developer. The system's reasoning capabilities reside in its ability to choose the next action from a set of possible alternatives, and to undo the effect of this action and attempt an alternative action if the first one is considered unsuccessful. It is the developer's responsibility to determine what actions are valid alternatives, and to define the order in which they are attempted. Specifying this flow requires an understanding of how the knowledge encoded by the developer interacts with the system's inference mechanisms when the system chooses an action.

The *knowledge base* contains domain specific knowledge, structured according to the needs of the solution process. This knowledge consists of descriptions, procedures, and relationships that identify, transform, and otherwise link the symbols representing the problem. A generic unit of knowledge can be described as a description associated with an action that advances the solution in some way. The description contains attributes that describe a problem state, and may include preconditions that define when it can be applied. An action may itself be composed of descriptions pertaining to other actions at different levels. The structuring of knowledge within the knowledge base corresponds to the problem's decomposition for processing.

Working memory is the area in which the description of the current problem solving state is maintained. This area is a central data repository for the system, and is constantly accessed during execution. The data it maintains consist of a representation of the evolving solution, along with all its intermediate states, and the information necessary to restore any of these intermediate states if the current solution proves to be wrong. All these representations must necessarily relate to the descriptions defined in the knowledge base.

The *inference engine* drives the reasoning process by applying the knowledge that corresponds to the problem state maintained in working memory. Its basic functions consist of *pattern matching* and *scheduling*. Pattern matching is the process by which data in working memory is matched to descriptions in the knowledge base in order to establish the set of applicable knowledge. This involves constraint testing when descriptions incorporate preconditions for their activation. Scheduling consists of ordering the actions within the set of valid alternatives. The order may be explicit within the knowledge, or involve conflict resolution if more than one action is equally applicable. Conflict resolution may devolve to a system default, or lead to the application of more knowledge. And when some chosen action fails, the engine "*backtracks*" to the last decision point, restoring all previous states along the way. It then schedules an alternative action from the set of actions available at that point.

The above provides a brief description of how an expert system reasons. In addition to this, operational systems will have a man-machine interface through which problem descriptions can be entered, and solutions displayed. The display may become relatively complex if system requirements include a trace of the reasoning behind the solution, justifying its correctness. Other support functions interpret the actions, manage the knowledge base, and provide a development environment. A development environment can be considered an integral part of most systems, as they are generally developed incrementally. These are the accessories with which the expert system approach to problem solving is applied.

1.4 Knowledge Representation

Knowledge representation consists of expressing a human expert's knowledge in a form that is suitable for processing by an expert system. Given the many forms that human knowledge can assume, no single form has proven adequate. This has lead to the development of a large number of formalisms. Each domain of application has its own, preferred method of expression. The suitability of any knowledge representation method will depend upon how easy it is to model the domain knowledge using that method. It will also depend upon the developer's skill in structuring the knowledge to obtain the maximum benefit from the inference mechanisms available in the expert system.

Broadly speaking, knowledge representation consists of (1) assigning symbols to describe some body of knowledge, and (2) defining the relationships between these symbols that are necessary to solve the problem. The symbols are then inserted into a knowledge base, whose access is controlled by the system's inference rules. Within the knowledge base, the symbols are stored in well-defined data structures associated with specific operations and interdependencies. Access to any structure leads to the corresponding associations. The knowledge representation methods provide a framework for structuring different types of knowledge into a form suitable for creating a knowledge base.

Two general categories have emerged for classifying knowledge: *descriptive* and *procedural* [1,2]. The former emphasises the use of descriptions to represent such things as concepts, facts, and problem states, and their inter-relationships. The latter emphasises the procedural aspect of reasoning, expressing knowledge as the process by which "intelligent" operations transform a problem into a solution. Most approaches incorporate aspects from both.

Systems that are considered descriptive employ constructs such as "*frames*" and "*semantic nets*". They describe conceptual objects in terms of attributes, component objects, or interdependencies with other objects. The descriptions may be associated with operations, and possibly include information about the required order of access. The order may be implicit within the interrelationships, or explicitly provided.

When descriptions relate to operations, they can be considered procedural. "*Scripts*" incorporate aspects from both categories, as they generally provide plans for interpreting the objects associated with them. "*Production systems*" express knowledge as "*rules*". These incorporate descriptions, preconditions, and operations, all of which identify when and how they should be applied, and what they will produce. Rules are further subdivided into "*goals*" and "*data*", which correspond to descriptions and actions.

The large amount of knowledge and complex interdependencies within a knowledge base must be structured to promote comprehension and efficiency. This structure can be provided by "*levels of abstraction*" or "*areas of applicability*" that focus, constrain, or otherwise limit the amount of knowledge that must be considered at any moment. When knowledge relates to knowledge, it is known as "*meta-level*" knowledge [1]. Meta-levels can be built onto any level of knowledge, by using layers to deal with successive levels of detail. Thus the system may reason about the appropriateness of some action before actually pursuing it, and that action may itself concern reasoning about other more detailed aspects.

While levels of abstraction correspond to decomposing the representation, areas of applicability relate to the division of tasks. In both cases, modularity makes the knowledge easier to express, easier to understand, and may result in a more compact representation.

The data structures within a knowledge base correspond to the conceptual objects used to represent the problem domain. The interdependencies among these data structures express the associations between knowledge that transform the objects and drive the reasoning process. One aspect of knowledge representation is concerned with choosing a set of symbols that adequately expresses the domain's conceptual objects and operations. This is the "knowledge engineering" problem, which is described more fully later. The second aspect is concerned with defining appropriate data structures and relationships to store the objects in, and link them together into a coherent whole.

1.5 Reasoning Models and Strategies

Reasoning is the process by which new facts or premises are inferred from those that are believed true. This simple definition of purpose glosses over much of the perplexity associated with the reasoning process, as our knowledge about how people actually reason is extremely inconclusive and often controversial [3,4]. The "reasoning" performed by an expert system is a highly stylised version of this process. Expert systems employ only a limited subset of the possible reasoning models, and these are principally concerned with devising stratagems that reduce the total amount of time spent on searches. The nature of the problem dictates what stratagems are suitable, while the mechanisms provided by the system determine how they are implemented.

People reason in order to integrate their experience into a world view, which then provides a model for "rational" interaction with the world. Several broad divisions have been established to classify the forms that reasoning can assume. These divisions are based on the subject's *awareness* of the process, and the *methodology* by which it is realised [5]. Our discussion will not enter the debate about the nature of awareness, and whether a computer program can attain it [6]. Nor will we examine the reasoning methodologies in detail. Our principal interest resides in the convenient classification labels these divisions provide us with.

Awareness is the quality of being cognizant of one's actions. It is measured from its two extremes, with *introspective* at one end and *reflexive* at the other. They describe to what extent the subject is aware of the stimuli to which he is responding. With *introspective* reasoning, the subject integrates stimuli into his world view, and his response is a deliberate consequence of that integration. At the other extreme, the subject's response is reduced to a *reflexive* reaction to the stimuli received, with no deliberation involved. In an expert system, the degree of awareness describes to what extent the system "deliberates" (integrates its stimuli into a world view) before choosing an action.

All reasoning methodologies assume some initial premise(s) from which the conclusion is inferred. The methods differ in what they assume, and the way their conclusions are derived. *Intuitive* reasoning derives conclusions by insight rather than formal logic. Its conclusions must therefore be accepted on faith. Both *deductive* and *inductive* methods rely on logical inference to support their conclusions, but differ in the nature of the conclusions drawn. Conclusions inferred by deduction convey no additional knowledge beyond that advanced in their premises. Their validity depends on the validity of the individual premises, and that of the inference rules. Conclusions inferred by induction draw general propositions from the available evidence. Although the premises support the conclusion, it does not necessarily follow from them, and its implications exceed those that could be drawn from the premises themselves.

Most human knowledge comes from inductive methods operating in inconsistent systems, and is often guided by intuitive leaps of inspiration [7]. This is particularly true of much of the knowledge that expert systems attempt to model. Current expert systems have extremely limited learning capabilities, which effectively curtails their ability to reason inductively (i.e., formulate general propositions from a given set of premises). However, this does not prevent them from employing inductive techniques -- in this case reasoning from the particular to the general -- once the propositions have been established and are assumed true. Deductive techniques are straightforward to envision as the chaining of premises according to inference rules. And although no documented system has ever been "inspired", certain types of intuitive knowledge can be captured in the form of heuristics.

Many problem domains involve more than one type of reasoning interwoven throughout the problem solving process. In others, any one of the reasoning methods could be applied with equal success. In most domains, the knowledge does not come prepackaged and ready for use, but must be shaped into the desired form. Nor does the knowledge neatly fit into the models described. In many cases, establishing what reasoning model is to be used will depend upon the perspective from which the system is viewed. Whatever the chosen reasoning methodolgy is, in most expert systems its implementation is based on deductive techniques applied in a reflexive manner.

In an expert system, reasoning can be reduced to the process of searching through its knowledge base to find a sequence of actions that transform the initial problem state into a solution state. Under most circumstances an exhaustive search is infeasible, and therefore heuristic strategies are used to prune the search space. The heuristic strategies attempt to decrease the length of the search by (1) reasoning in the direction of the least number of possibilities; (2) eliminating impossible alternatives at an early stage of the search; and (3) following the most promising paths first. Each of the three increases the likelihood of finding a solution quickly, but does not guarantee success. This is due to the fact that even after pruning, the system may still retain a high number of alternatives.

The number of possibilities that the system must consider often depends on the direction in which the search is performed. Possible directions consist of *top-down*, *bottom-up*, and *bidirectional* [1]. The first two define two distinct approaches, whereas the last combines both in an opportunistic fashion. They describe the reasoning methodology for solving the problem, and the way it is implemented by the system's inference engine. Each may employ a different approach within the same system. However, the suitability of any approach is highly dependent upon the problem itself.

With a *top-down* strategy (otherwise known as *goal driven*, or *backward chaining*), the system reasons backwards from some final, desired "goal". It establishes whether it can reach that goal by determining if it can satisfy the conditions associated with it. These conditions may themselves be subgoals, which require the satisfaction of further subgoals in a recursive fashion that brings them back to the original problem statement.

With a *bottom-up* strategy (also known as *data* or *event driven*, and *forward chaining*), the system reasons forward from the current problem state to the solution. It examines the current state of data or events to determine what conditions have been satified, then activates the corresponding rules in the knowledge base. This process is repeated until the solution is found.

For many problems a single strategy is insufficient. Instead, they combine both top-down and bottom-up reasoning in a *bidirectional* fashion, employing either one according to which seems the most promising. This can be applied either during the problem's decomposition, or during its execution. In the first case, certain subgoals are resolved using one approach while others employ the alternative approach. In the second, if the problem does not succumb to one approach, the alternative one is attempted.

Whenever possible, the search must be selective in its choices. However, very often it is difficult to evaluate the alternatives, and determine which is optimal. There are various means of dealing with uncertainty about which choice is "best", and obvious consequences to their use. The simplest methods consist of *backtracking* and *postponing decisions*. When these are inappropriate, then *probabilisitic* methods can be employed.

Backtracking is among the easiest ways of dealing with uncertainty. It consists of attempting one alternative, and if that one doesn't succeed, restoring all changes and attempting another. Although its simplicity makes it attractive, it is not always possible to apply. It can only be used if the attempt does not result in any irrevocable changes being made to the problem data. *Postponing decisions* until all the necessary information is available is an alternative choice. However, it also has its limitations, and requires that delays be tolerable, and do not become indefinite.

Various techniques exist for incorporating probabilities into the decision process. Certain types of probabilistic knowledge can be obtained from the expert and expressed in heuristics. Other methods assign probabilities to the different possibilities, then propagate these values down the various paths. *Belief systems* [5] extend the notion of probability by calculating the "plausibility" of their choices. This expresses the value of both the supporting evidence, and the non-refuting evidence. Many other variations exist. Although all are useful, a common difficulty is the problem of assigning some quantitative value to a qualitative attribute.

Many real-world problems cannot be resolved using any single approach, but require the use of several approaches, possibly applied simultaneously. Such problems are said to employ "*multiple lines of reasoning*". When these are employed in different areas of applicability, interaction between the areas can be limited by careful decomposition of the problem. A *divide and conquer* strategy, or a *hierarchical decomposition* are two possible strategies for partitioning the problem. When different lines of reasoning are applied to the same problem area, each constitutes an alternative strategy that could benefit from the information obtained by the use of the others. In all cases, a certain amount of interaction is necessary to integrate the results obtained. Establishing what form this interaction could take is one of the issues examined in chapter 2.

1.6 Characterisation of Problems

The nature of the problem plays a major role in determining what form the expert system will assume. The type of uncertainty present in the problem, and the way it is handled are two critical factors in this choice. Both the type of problem, and its specific characteristics will affect the choice of knowledge representation and reasoning methodology. The nature of the desired solution will have an additional impact. All these factors will be important in determining the system's problem solving structure, and its capabilities and limitations.

The problems that expert systems are designed to solve can be divided into two categories: *classification problems*, and *construction problems* [8]. Diagnosis, analysis, and interpretation, are typical classification problems, while design, prediction, and planning are generally considered construction problems. Each category employs a different approach for representing and transforming the problem. With classification problems, all possible solutions are specified at the beginning. Reasoning about a given problem case is principally concerned with mapping input data into the solution set. In construction problems, each problem case produces a unique solution, constructed from a unique combination of the basic conceptual objects employed. With these problems, the number of solutions involved makes it impossible to specify all solutions a priori. Certain problems will involve tasks from both categories.

There are four general categories for classifying systems according to the characteristics of their searches: *monotonic*, *partially commutative*, *commutative*, and *non-commutative* [2]. In a monotonic system, the application of one rule does not prevent the application of another that could have been applied when the first was chosen. In a partially commutative system, if the application of a particular sequence of rules produces some state, then any "meaningful" permutation of this sequence will produce the same result. A commutative system combines the properties of both, whereas a non-commutative system has the properties of neither.

Each of the above categories has implications on how the search strategy must deal with uncertainty. In a monotonic system, backtracking is possible. In a partially commutative system, the order in which the solution is constructed is not critical. A commutative system will combine the advantages of both, benefiting from both the ability to backtrack, and a non-critical processing order. The same will not be true of a non-commutative system, which must commit itself to a single solution path once its choice is made. As a consequence, non-commutative systems are obliged to evaluate their options much more carefully before making any decisions.

The size of the problem space, the ease with which it can be represented, and its factorability will affect the problem's decomposition and the choice of search strategy. The strategy's complexity will largely depend on the amount of interaction introduced by the decomposition, the characteristics of the initial input data, and of the intermediate solutions. All of these will contribute to the uncertainty the system must handle. Interaction will affect the number of alternatives available. Unreliable data increases processing complexity, and time varying data introduces processing constraints. The stage at which partial solutions are evaluated, and the efficacy of evaluation will influence the amount of uncertainty present at that stage in the solution. Another factor to consider is whether the uncertainty remains isolated, or is propagated throughout the system. Each will add its flavour to the search requirements.

The nature of the solution, and the form in which it is expected will have an equally important impact on the choice of strategy. Certain problems will have only one solution, while others may have several. If several solutions are possible, is any solution acceptable, should the best one be found, or are all solutions necessary? With some problems, the existence of at least one solution is guaranteed, while other problems may not have a guaranteed solution. This will have a significant impact on knowledge requirements, as it is critical to have some timely way of determining when to abandon futile searches. It may be necessary to structure the reasoning strategy so that it is understandable to the users of the system. All these factors must be considered when a strategy is selected.

1.7 Knowledge Engineering

Knowledge engineering is the process of acquiring the knowledge necessary to solve problems in a given domain, and creating an expert system with it. The knowledge engineer must not only capture the relevant knowledge, but must also represent it using a structure that optimises the system's problem solving abilities. Although general methodologies exist, there is no prescribed method for accomplishing this. Each problem will have a combination of characteristics that make it unique, and each system will be a unique response to that problem's requirements as perceived by the knowledge engineer. The success of any system will largely depend on the engineer's perspicacity and skill.

The expert system development cycle is generally described as an iterative process which passes through several refinement cycles. The process involves acquiring the knowledge, developing a suitable representation for it, structuring the search strategies, and observing how the resulting system performs. Feedback from the performance tests will indicate what improvements are necessary, leading to further refinements of the system. This iterative process is repeated as often as necessary until either the system objectives are attained, or the system is abandoned.

In many systems the knowledge acquisition phase is considered one of the most difficult and time consuming aspects of development. This can be attributed to the amount of time required to thoroughly understand the problem. Familiarisation with the problem domain is necessary before the problem can be analysed. Its analysis requires careful consideration of how subproblems interact, and what knowledge is relevant. Even when the knowledge engineer is familiar with the domain, has an idea of the relevant knowledge, and has an expert at his disposition, the "expert" knowledge he wishes to obtain is generally difficult to extract.

The knowledge representation phase involves similar difficulties. There is no prescribed method for mapping abstract concepts into symbols. The proposed representation formalisms must be adapted to the specific requirements of the problem. The actual representation must have a structure appropriate for implementing the chosen strategies with the available inference mechanisms. In addition to this, it may be necessary to represent what knowledge the system does not have, and otherwise close its definitions. This is not always obvious, and thus adds to the difficulty.

Developing the search strategies must proceed together with the other phases, as all have an impact on each other. The choice of strategy is influenced by the characteristics of the problem. At the same time, the available knowledge determines what strategies can be implemented, and their effectiveness. Determining what strategy will be the most suitable for a given problem is not simple, even when the problem's characteristics are relatively well known, and its representation has been established. The initial choice is often based on intuition, and requires substantial refinement before the expected performance is attained. Nor can this performance be guaranteed unless all possible problem occurrences have been considered.

Knowledge engineering can be viewed as the art of reducing some open body of knowledge into a consistent symbol system that models a subset of that knowledge in a way that permits its processing on a computer. It requires skill, perspicacity, intuition, and time on the part of the developer, with no guarantee that the amount of effort put in will produce a comparable result. Few, if any, systems attain the level of reasoning complexity that researchers would like to claim these systems are capable of achieving. The reason for this is that although it is easy to collect facts, understanding how the mind combines these facts to derive solutions in a timely way is an open issue. For this reason, designing expert systems remains an art, as opposed to a methodological science. The system's intelligence resides in the associations that the developer builds into the knowledge base rather than in the facts themselves. It is these associations that drive the solution process, and determine the system's problem solving intelligence.

1.8 Performance Evaluation

Evaluation can be defined as the process of verifying and validating the system's problem solving abilities. Verification consists of demonstrating the system's consistency, completeness, and correctness, while validation involves establishing that the system attains the objectives which it was designed for. As attaining the problem solving ability of a human expert is impossible with current technology, the goal is reduced to one of finding acceptable solutions within some narrowly defined problem domain. This requires defining what is considered acceptable, and the limits of the domain. Defining the limits of the domain of applicability requires determining the range of problems for which acceptable solutions can be found.

Evaluating the performance of an expert system is a critical aspect of its development. Yet, it is often one of the most rushed phases during which many crucial factors are easily overlooked. The difficulty associated with evaluating an expert system arises because the expert system is not only a program, but also a model that can only approximate its real-world equivalent [9]. Thus its performance will necessarily differ from that of its real-world counterpart. Determining what these differences are is an important aspect of the evaluation process, necessary to establish the system's abilities and limitations. However, these differences are often difficult to establish due to the nature of the systems, and the subjective evaluations involved.

One of the most difficult aspects of evaluation is seperating the bugs in the program from the errors in the model. The difficulty arises because there is no way of guaranteeing that the system contains no bugs, yet proving the correctness of the model relies entirely on the system. Even though the model may appear to work correctly, there is no way of establishing how it will work under unforeseen circumstances. We can only assume that what holds for the observed examples can be generalised into a statement about the model itself. The chosen examples must therefore be representative of all the problems the system may encounter. Yet the number of possible examples is generally large, and it is the unforeseen examples which are precisely those omitted from the set of representative problems used to judge the system.

Chapter 2
Cooperative Problem Solving

The expert system model is a powerful tool for solving many problems that contain uncertainty. Its principal limitations reside in the resource constraints of a single machine, and the developer's analytic skills. *Cooperative problem solving* examines how the model can be extended to take advantage of the benefits that distribution offers. Here, the problem solving expertise is distributed among a set of intelligent agents that operate autonomously, but cooperate with each other to construct a single solution. The agents' concurrent activity reduces processing time and allows the simultaneous pursuit of multiple lines of reasoning. However, along with these advantages, distribution introduces additional complexity, and raises the major issue of how control should be implemented.

The terms "control", "coordination", and "cooperation" all refer to directing the activity of a group. Each term carries connotations on the nature of the direction present. Control simply refers to directives, and does not make any allusion to their collective meaning, nor to the quality of the direction they provide. Coordination has collective significance, and implies a non-conflicting combination of activity. Cooperation conveys additional significance in that all members work for the benefit of the group. The following definitions are used:

Def^n *Control*: Control refers to the exchange of implicit or explicit directives that govern what goals each member of the group pursues.

Def^n *Coordination*: Coordination refers to the combined activity of the group wherein the goals of the individual members <u>do not adversely affect</u> the goals of the other members of the group.

Def^n *Cooperation*: Cooperation refers to the combined activity of the group, wherein all members' goals are <u>coordinated</u>, and <u>all members pursue a common goal</u>.

A system in which control is exerted is not necessarily coordinated, nor do coordinated agents necessarily cooperate. On the other hand, a group must have some form of control in order to coordinate the activity of its individual members, and cooperating agents must also coordinate their combined activity if their activity is to be cooperative. Communication is an essential factor in all three cases, in order to unite the disparate agents into a group.

Within a DPS system, agents make control decisions which determine what local actions or interactions the agents will undertake. These control decisions must be coordinated so that the agents work together as a coherent team, and do not choose activities which conflict with each other's goals. The central issue is how global coherence can be maintained with a minimum amount of communication and without sacrificing the individual agents' autonomy and concurrency. The metaphor "*cooperation*" expresses how autonomous agents interact using "intelligent" communication policies that allow each to influence the others' behaviour, and thus converge onto a globally consistent solution. This type of behaviour can be equated to that found in human organisations.

2.1 Reasoning Paradigms for Complex Problems

The problems considered by cooperating agents are those which are intrinsically difficult due to their size, their complexity, and their real-time response requirements. Their difficulty resides in the complexity of their information requirements [10], and the "*bounded rationality*" of the agents that process the information [11]. Bounded rationality is defined as the limitation on an individual's capacity to process information in order to arrive at "rational" decisions. Information complexity arises because of the amount of information required to solve the problem, its diverse origins, and the interdependencies within it that make it difficult to partition into smaller more manageable units. Yet, partitioning is necessary, as an agent's bounded rationality places a limit on the amount of information that one agent can assimilate, and the amount of control that it can exert within a given time period. People have evolved various ways of handling such problems on both the individual and social level. Understanding these methods is useful before attempting to model them.

Theories about human cognitive processes generally agree that people employ two basic paradigms when reasoning: the *sequential* (or logical) paradigm, and the *parallel* (or gestalt) paradigm [5]. Both paradigms rely on decomposing the problem into more tractable subproblems to reduce complexity, but they differ substantially in the way the resulting subproblems are solved and reintegrated. The sequential approach decomposes problems into relatively independent subproblems which involve only a limited subset of the available data at any moment. The subproblems are then solved independently, and their partial solutions are integrated into a final result. The principles involved in the parallel paradigm are much more difficult to grasp. Somehow, the significance of all the available data is simultaneously considered and integrated. And this occurs inspite of the very real bounds on the amount of information that a single agent can assimilate at one time.

The sequential and parallel paradigms pervade all human activity, whether it occurs at the individual level in fundamental activities such as vision, motion, and creative thinking, or on a larger scale, in social organisations that address problems beyond the scope of a single person. Applying the sequential approach is relatively straight-forward. Given that subproblems involve little interaction, they can be solved either sequentially or concurrently, depending on their temporal relations. Problems that fall into the second category cannot be solved so simply. The solution to any subproblem is dependent on all the others. This interdependency dictates that processing must proceed in parallel and with cross reference if any solution is to be found. Accomplishing this effectively requires some mechanism that focuses on the relevant information, and provides a global overview with which subproblem processing is guided. One approach to this problem is that of considering the system as an "*organisation of intelligent agents*" [12].

An organisation is the structure that identifies the flow of communication, the locus of decisions, and the sphere of influence of the different task centres involved in solving a given problem. Its purpose is the division of tasks to respect the constraints of an individual's *bounded rationality*. The organisation's structure differs according to the complexity and uncertainty present in the problem, and has an equally important impact on the organisation's effectiveness. Fox has identified six basic forms of organisation, each of which has distinct advantages and disadvantages that make it appropriate for certain environments while totally unsuitable for others [11]. These organisational forms are described in Figure 2.1.1.

(1) A ***single member organisation*** is the least complex of all, with no communication requirements, and a simple control structure that is easy to realise. However, its restricted resources place a limit on the problem complexity that it can handle.

(2) A ***team organisation*** overcomes this limitation by providing more resources. Each member is assigned a subtask in the problem. Members work on their individual tasks, but share a common goal. Each member can readily access the information and results produced by the others. Coordination is achieved through some form of *consensus* among the members. The organisation reaches its limits when the problem's complexity makes collective decision making more time consuming than beneficial.

(3) A ***simple hierarchy*** is an alternative way of providing additional resources. Here, a single decision maker handles decisions for the group, assigning specific tasks to each member. He must therefore have all the information necessary to make valid decisions. Although suitable for many problems, centralised decisions create a bottleneck when uncertainty is encountered. This form is inadequate when the problem's complexity makes a single decision maker insufficient.

(4) A ***multilevel hierarchy*** constitutes the next stage of problem solving ability. Authority here is distributed within a tree structure. Members within the same level are only responsible for part of the information, decision making, and control. Authority is shared within a level, while it is exerted over lower levels, and must yield to directives received from above. Individuals have well defined tasks, responsibilities, and communication requirements. Inspite of this, control is complex and bureaucratic overloading may occur, with top levels swamped under high uncertainty.

(5) A ***decentralised organisation*** overcomes this limitation, but introduces other disadvantages. Here, top level member are organisations themselves, responsible for all resources and decisions within their "division". Top levels are more concerned with *long term strategies*, while lower levels deal with the more detailed aspects of daily operation. The problem here is that integration is difficult as no direct mechanism is available to coordinate activity and promote cooperation among the divisions.

(6) A ***market organisation*** employs an alternative coordination scheme. The market consists of a set of autonomous organisations, with each responsible for its own resources and decisions. Each organisation has the choice of performing tasks locally, or contracting them out, with cost the major factor in the decision. The inherent difficulty with contracting mechanisms is how to establish the true market value of services, a critical factor when deciding how tasks should be handled. Constructing a global overview of the market's state may be time consuming.

Figure 2.1.1: Six basic forms of organisation

There is room for much diversity within these six frameworks. Although they describe the locus of responsibility, they do not prescribe how decisions are actually made. The decision procedure used, and the source of information for the decision, have a critical impact on the behaviour of an organisation. Decisions are generally made by either *consensus* or *decree*. The former involves group participation, while the latter relies on a single person to make decisions for the group. A comparison of the advantages and disadvantages is provided in table 2.1.1. The decision process in most organisations rarely occupies either extreme. The source of information may add another skew to it. If the supplier filters the information on which the decisions are based, he has a major impact on these decisions even though their maker may not be aware of this. The combination of decision procedure and source of information make each organisation unique.

Table 2.1.1

The advantages and disadvantages of group decisions. [13]

Advantages:

Groups can accumulate more knowledge and facts.

Groups have a broader perspective and consider more alternative solutions.

Individuals who participate in the decision are more satisfied and more likely to support it.

Group decision processes serve an important communication function, as well as a useful political function.

Disadvantages:

Groups often work more slowly than individuals.

Group decisions involve considerable compromise which may lead to less than optimal decisions.

Groups are often dominated by one individual or small clique thereby negating many of the virtues of group processes.

Overreliance on group decisions may inhibit management's ability to react quickly and decisively when necessary.

Another important characteristic of organisations is that they evolve. Complex organisations do not suddenly appear. They either evolve from a simpler form, or result from a split with a parent organisation, which already has a fully developed structure. An organisation's evolution allows it to handle problems of increasing complexity. At the same time, it increases the complexity of the coordination task. As the organisation evolves, it must adapt its control structure to its changing needs, often passing from one form to another. Thus, organisations are dynamic solutions that evolve in response to changes in the nature of the problem addressed, and their environment.

Organisations are solutions to problems whose complexity exceeds the scope of the individual. Certain researchers believe that an analogous model can be applied to the individual's cognitive processes [12]. The organisational forms in current use have historical roots, and continue to evolve as survival needs change [14]. The formal structures described by organisational theory are not complete models of the organisational phenomenon. It is a generally recognised fact that all the formal structures are necessarily complemented by informal structures in order to function [15,16]. Organisations exhibit similar characteristics to other reasoning processes in that although their manifestation is visible, how they actually function is largely unobservable.

2.2 Distributed Problem Solving

Distributed artificial intelligence (DAI) is concerned with studying how a group of intelligent agents can combine their resources to produce a problem solving potential greater than the sum of the individual agents' intelligence. Thus, it is directly concerned with *gestalt* reasoning. *Distributed problem solving* (DPS) examines the issues involved in getting a group of semi-autonomous agents to interact cooperatively while solving problems in a distributed computing environment [17]. The individual agents are sophisticated problem solvers constructed using standard AI techniques, and they cooperate because of the bounded rationality of any single agent. However, the way this is realised is constrained by the limited bandwidth available for communication. The crucial issue is how these agents can coordinate their activity without introducing a high overhead that would effectively cancel any of the benefits that could be obtained from multiple agents.

The distributed environment in which these agents operate is characterised by the following three criteria [18]:

(a) The distributed agents communicate with each other only by passing messages; (agents are autonomous).

(b) Agents do not share any common physical clock; (agents operate concurrently and at their own speed).

(c) The channels connecting the agents are reliable FIFO channels, but they introduce arbitrary delay in delivering messages; (nondeterminism).

This type of environment offers definite advantages in terms of parallelism and fault tolerance. However it also introduces the additional complexity of coordinating global activity under increased uncertainty due to nondeterminism. The bounded rationality of the agents and the limited communications bandwidth dictate that coordination must be accomplished in a decentralised fashion with a minimum amount of communication. Given that the agents must cooperate together at the same time that they operate asynchronously, they must communicate in a sophisticated way. A consequence of this is that the agents in DPS systems must be capable of *introspective* reasoning about their actions and reactions, in addition to having their basic problem solving intelligence.

DPS systems seem eminently suitable for AI applications which are inherently distributed. A geographic distribution based on the intrinsic properties of the problem produces one partitioning, while a decomposition based on the functional aspects of the system produces another. Further partitioning along hierarchical and temporal lines is possible. Air traffic surveillance, cooperating robots, and understanding of spoken natural language are examples of application domains [17]. Problem partitioning is equally beneficial for problems whose complexity exceeds the understanding of a single individual. Furthermore, DPS systems offer a way of solving problems that involve *gestalt* reasoning.

An ideal problem decomposition produces functionally independent, but logically related subproblems that must concur to pro'uce a global solution. The difficulty of coordinating the global problem solving process arises from the fact that the natural distribution of data within the problems does not coincide with that required to make effective control decisions. Furthermore, the overhead associated with collecting the necessary information from the distributed sources would degrade the system's performance to unacceptable levels. The coordination mechanism within these systems must exploit the functional independence and the logical interdependence between subproblem partitions to constrain processing and converge on solutions in spite of incomplete knowledge and uncertainty.

The agents in a DPS system cannot be totally autonomous. Their individual activity is related by the interdependencies that link them together into a single problem solving system. Agents are given the autonomy to react "opportunistically" to changes they "perceive" in the problem state. However, what they perceive is determined by the policies that govern their interaction with other agents. By varying the type of permissible interactions among the agents, the system's behaviour can be modified to provide individual agents with varying degrees of autonomy and influence on global behaviour.

The relaxation of tight synchronisation requirements in a loosely coupled environment implies that agents must be aware of what is expected from them, and of what is occurring elsewhere, particularly when no communication is forthcoming. This introduces the need for agents to have a *planning function* that integrates their activity with what they perceive of the global system. The amount of autonomy an agent can assume without compromising system performance will depend upon the validity of its local planning decisions. The correctness of these decisions will depend upon the agent's planning knowledge and local view of global activity. This is both a knowledge engineering and communications problem, in which the communications and resource requirements must be balanced against the timeliness and accuracy necessary to produce acceptable solutions within the context of some specific problem.

The type of response possible with any given system configuration will be highly problem dependent. The granularity, frequency, and expected validity of the individual agents' decisions will determine the amount and type of interaction necessary. This, in turn, will affect the agents' accuracy, sensitivity, and autonomy. Similarly, the way control is implemented will determine how serialised the agents' activity is, whether it has a global focus, and how this focus can be modified to take advantage of local opportunities most effectively. All are unresolved questions that are difficult to answer in a general context.

2.3 Control and Coordination

The characteristics of a distributed environment dictate that global system control must be implemented in a decentralised and asynchronous fashion in order to maintain an acceptable level of overall performance [19]. The characteristics of DPS problems dictate that individual agents must be aware of what is occurring elsewhere in the problem solving network, and adapt their activity accordingly if their individual activity is to remain globally coherent. As stated earlier, the terms "control" and "coordination" both refer to directing the activity of a group. "Control" simply refers to the directives by which the activity of individual members is guided. "Coordination" implies a harmonious combination of activity, whereby all agents pursue non-conflicting goals. The term "coordination" is more appropriate than "control" for describing the way in which a group of DPS agents would direct their combined activity.

Within a DPS system, communication is the crucial factor that determines how agents coordinate their activity. The degree of autonomy, and the extent of any agent's influence will be determined by: the nature of the information communicated; the number of agents affected; how the exchange is initiated; and the type of response its receipt elicits. The way in which individual agents arrive at local decisions that are globally optimal is a question of choice. Research in this area has come up with a variety of schemes that result in distinct types of behaviour appropriate for different problem environments.

Communication among the agents serves a dual purpose. It circulates problem specific information between the different processing centres, and conveys the information by which the agents' activity is coordinated. The circulation of information corresponds to a "production model", whereby any system transforms its raw materials into a finished product. The need to convey information concerning coordination arises because of the nature of AI systems, where a great deal of intermediate uncertainty may occur before a solution is found. Deft handling of this uncertainty is essential, without which the system would flounder in an undirected search. A certain amount of information concerning coordination is implicit within the problem data. However, its clarity rapidly decreases as problem complexity and uncertainty grow. In this case, the agents must explicitly communicate information related to coordination if they are to retain a common focus and cooperate rather than simply complement each other's processing.

The agents communicate to resolve inconsistencies and arrive at compatible decisions. The information communicated is highly problem dependent and must necessarily be concise. At the same time, conciseness may lead to the loss of detail, which must be compensated for. Another important consideration is the timeliness with which information is communicated. This also leads to a tradeoff in which the information's accuracy is affected. Meeting the requirements of conciseness and timeliness will contribute to the uncertainty that the agents must deal with.

The actual information communicated consists of *data* and *directives*. There is no hard distinction between these two, as their nature will depend upon how the information is represented, and how the agents react to it when it is received. Partial solutions, hypotheses, alternative hypotheses, constraints, goals, unresolved goals, and tasks are all obviously problem oriented, and can be considered data. However, if the system's operation tends more towards a production model, they can be considered directives, which trigger the desired action by their appearance. The information related to coordination will only differ in the aspect of the problem it addresses. Whether it is considered data or directives will depend upon how the individual agents arrive at decisions.

There are various ways of focusing the system's attention on desirable activities. Along with the basic *data* and *goal driven* strategies described in section 1.4, either a *knowledge driven* or *island driven* approach can be employed. The knowledge driven approach focuses on the sources of knowledge in the problem solving process, while the island driven approach places the focus on specific hypotheses among the available alternatives. With the data driven approach described earlier the agents principally react, while with a goal driven approach they provoke. A knowledge driven approach concentrates on determining which agent has the relevent knowledge, while an island driven approach attempts to identify and expand promising hypotheses or "solution islands". Certain problems that involve multiple sources of uncertainty may require the simultaneous use of different approaches.

The communication policies employed directly affect system behaviour. Agents may exchange information voluntarily, on request, or through a mixed initiative. Their response to communication can range from the reflexive to introspective, depending upon the amount of coordination intelligence possessed by an agent. This will largely depend upon where and how the decisions are made, and who is involved. Decisions can be made by consensus or decree, with all the advantages and disadvantages stated earlier (section 2.1). The choice will depend upon where the necessary information is located, and the cost of transmitting it compared to that of the consensus process and its possible benefits.

A highly reflexive system will function with few decisions, requiring little or no coordination knowledge. As the problem's complexity increases, decisions become more critical, and therefore require more introspective reasoning on the part of the agent. Under certain circumstances the agent may need to vary the way it responds, in which case considerable knowledge may be necessary.

Highly reflexive systems generally function in a tightly coupled manner. Hearsay-II is a typical example [10]. They employ a centralised data space in which individual agents collect their information and post their results. The system relies on a close association between "*pattern*" and "*action*" to drive the agents, while the shared data space provides the global overview with which activity is focused. Although this approach is suitable for certain problems, the agents' limited intelligence about coordination restricts the use of this model, nor can the way in which the agents interact be considered truely "cooperative"[18].

Researchers have developed a variety of schemes for coordinating activity in the different scenarios that occur in DPS systems. The schemes differ in the locus of decisions, in the agents' responsibilities towards each other, in the timeliness of the exchange, and in the communication overhead. All the schemes rely on an exchange of information that results in some form of either implicit or explicit decree or consensus among the agents. The agents' decisions involve varying degrees of introspection according to how critical the decision is, its complexity, and the amount and type of uncertainty present. Durfee et al. have identified six trends in coordination schemes [20], outlined in figure 2.3.1. The fact that there is no DPS system in daily usage today demonstrates that these ideas are not yet fully developed.

Negotiation: Using dialogue among nodes to resolve inconsistent views and to reach agreement on how the nodes should work together to cooperate effectively.

Functionally Accurate Cooperation: Overcoming inconsistency by exchanging tentative results to resolve errors and converge on problem solutions.

Organisational Structuring: Using common knowledge about general problem solving roles and communication patterns to reduce the nodes' uncertainty about how they should cooperate.

Multiagent Planning: Sharing information to build a plan for how agents should work together, then distributing this plan throughout problem solving.

Sophisticated Local Control: Integrating reasoning about other agents' actions and beliefs with reasoning about local problem solving so that coordination decisions are part of local decisions rather than a separate layer above local problem solving.

Theoretical Frameworks: Using mathematical models of agents, their beliefs, and their reasoning to understand the theoretical capabilities of cooperative DPS networks.

Figure 2.3.1: Trends in distributed computing

Negotiation corresponds to a team or market structure, depending upon the subject negotiated. It generally employs an explicit consensus process, with all the associated benefits and disadvantages. The agents require the knowledge necessary to establish the relative merit of the different proposals being negotiated, and mechanisms by which they can influence the choice. It differs from the other forms in that dissatisfied agents are free to disassociate themselves from the group's decision.

Functionally Accurate Cooperation (FA/C) relies on the exchange of intermediate solutions between affected members to corelate partial solutions. The intermediate results are not necessarily accurate or consistent, but their timeliness focuses activity and prunes searches at an early stage. It corresponds to a form of consensus between agents, based on the information they relay to each other rather than on a formal negotiation process.

Organisations pervade all possible schemes, as any system with more than one agent will have some implicit or explicit relation defined between its agents. When the relation is explicit, agents may employ knowledge about this relation to improve their interaction with other system members. Organisations of agents have many of the features discussed earlier. The implications of using organisational structuring in DPS systems are discussed more fully later.

Multiagent planning arises from the need to be aware of the other agents' intentions, to respond flexibly, and to not expend extensive time in negotiating. With this model, agents first negotiate a plan of mutually compatible activities, which is then distributed, and followed by the individual agents. The plan is established either by consensus or by decree. With the latter, a chosen agent is sent all the relevant information and formulates the plan. This model accomodates problems in which team membership and roles change dynamically. It differs from negotiation in that agents are bound to the group's decisions.

With *sophisticated local control*, agents are capable of reasoning about their role in relation to that of the other agents, and of integrating the information they receive in order to construct a local view that allows them to coordinate their activity with the other agents. With this form, the coordination problem is considered an integral part of the problem, and reasoning is as concerned with coordination as with the problem itself.

Theoretical frameworks view cooperation and its implications in an abstract manner, constructing models based on logic or game theory. These models examine what knowledge is available to the agents, and what knowledge must be communicated to them in order to resolve conflicts. The models examine under what conditions cooperation is possible, and the different ways in which the necessary information could be propagated.

The above paragraphs provide a brief overview of the various directions that have been or are being pursued. There are further subdivisions within these categories, often used to describe the individual twists of particular systems. What all these schemes have in common is the basic objective of promoting effective communication between the agents. If the system is to function both effectively and efficiently, the effects of *disruption*, *distraction*, and *local idleness* must be considered [21]. Their description follows.

Disruption occurs when an agent is interrupted from some useful activity to fulfill another function. Although the second activity may in itself be useful, swapping between the activities will simply increase processing time.

Distraction occurs when an agent is flooded with information that distracts the agent from the relevant information it is processing, and results in additional work with no positive contribution. Even if the only activity associated with receiving this information consists of establishing the information's irrelevance, this will still require processing by the agent.

Local idleness concerns how an agent should occupy itself when it has no immediate work to perform. If the agent simply continues producing alternative hypotheses, assigning tasks or calling meetings, it will increase the amount of disruption and distraction occurring in the system, degrading overall performance.

The appropriateness of a coordination scheme for agents in a DPS system will depend upon how well the chosen strategy fits the characteristics of the problem solving process. A large number of factors are involved in the scheme, many of which can have undesirable side effects if not carefully considered. The problem, and the knowledge engineer's skill in analysing the problem, in choosing a scheme, and in integrating the scheme into the problem solving process will affect the scheme's efficacy. The choice of scheme is often guided by the knowledge engineer's preconceptions, and personal experience with group dynamics.

2.4 Cooperation and Communication

Cooperation is a form of group interaction whereby individual members work together towards a common goal, and sublimate their personal interests in favour of those of the group. When agents cooperate, they choose individual goals that have the greatest benefit for the group as a whole. The members of the group must have certain qualities for their behaviour to be cooperative; they must (1) share a common objective; (2) be capable of independent choice; and (3) be rational. The common objective provides the agents with the basis by which they judge the relative merit of their individual goals. The ability to choose implies that the agents are capable of sublimating their individual priorities when this is necessary for the benefit of the group. Their rationality allows the agents to relate their individual goals to the common objective, and make reasoned decisions about the "best" action to take. Making an optimal decision requires that the agents be aware of the choices available to the group as a whole. In a distributed environment, communication is the only means by which the agents can establish what choices are available.

The agents in a DPS system cooperate for a variety of reasons. The objective may be to decrease total processing time by deriving partial solutions in parallel, to increase the scope of the search by considering more alternatives, or to arrive at more timely solutions by exchanging constraints and focusing each other's attention. In certain cases, the objective will be to avoid redundant activity, while in others, redundant activity will be encouraged in order to increase confidence in proposed hypotheses, or provide fault tolerance. A primary objective in DPS systems is to decrease communication requirements by providing agents with the ability to reason about system activity and make local decisions that are globally optimal. It is here that the full implications of cooperation are involved.

Cooperation has different meanings when it is applied to people, in game theory, or to agents in a computer system. The first examines the philosophical issue of how "rational" beings can cooperate when they have divergent interests. The second attempts to develop a formal framework that defines the states of knowledge necessary for cooperation, and the resulting communication needs of the participants. The third is principally concerned with how the concept of cooperation can be used to reduce the amount of communication necessary to coordinate the activity of a set of distributed agents. Here, the relation between knowledge, action, and communication can be exploited. Communication assumes another facet whereby not only is the message important, but the act itself conveys information. This can be used to construct "knowledge based protocols" with which the system's state of knowledge can be changed.

In all three cases, cooperation requires that the agents be aware of each others' existence, of the fact that their own, individual activity has an impact on the activity of the others, and that the others' actions have an impact on their own. All agents are considered *"rational"*, with a "rational being" broadly defined as one who attempts to optimise its "pay-off" when deciding what course of action to take. Globally optimal decisions are not possible among agents who are purely rational, logical, and self-interested. This is illustrated in figure 2.4.1 which describes the classic *"Prisoners' Dilemma"* [22]. In addition to being rational, agents must be aware of a collective goal, and have assumptions about the other agents' behaviour and intentions.

If this is true, then by knowing what others know, and knowing what their actions imply, it is possible to derive knowledge about overall system activity with less transmission of explicit information.

Prisoner's Dilemma

"You and your accomplice have been charged with having committed a crime and are now prisoners, sitting in separate cells, unable to communicate, and awaiting trial. The prosecutor offers a deal to persuade you to confess: "There is enough circumstantial evidence to convict both of you, so even if you both remain silent you will both be convicted and locked up for a year. But if <u>you</u> admit your guilt and help convict your silent partner, you will go free and he will be locked up for ten years. The reverse happens, of course, if he confesses and you remain silent; and if you both confess, then unfortunately, you will both get nine years."

		The other prisoner	
		doesn't confess	does confess
You	don't confess	1 year	10 years
	do confess	0 years	9 years

(1) Either the other prisoner will confess or he will not.
(2) If he does confess, then confessing is better for you than not confessing.
(3) If he does not confess, then (again) confessing is better for you than not confessing.

Therefore, in both cases confessing is better for you than not confessing. If the other prisoner doesn't confess and you do, then you don't get a prison sentence, while if he also confesses, you only get a nine year sentence.

Whether communication is possible or not, if both prisoners are rational, logical, and purely self-interested, they will choose to confess -- which is locally optimal, rather than not confess -- which would be the globally optimal choice.

Figure 2.4.1: The Prisoner's Dilemma

The example of figure 2.4.1 succintly reveals why rationality, logic, and self-interest are inadequate for inducing cooperative behaviour when individual preferences diverge. As presented, the choices are causally independent, and individual rationality is insufficient for resolving the dilemna. Ginsberg [23] argues that if agents are to cooperate, their choices must be guided by "*common rationality*". Here, each agent knows the rationale of those it interacts with, and employs a "*collective decision procedure*". The agents' decisions are not considered independent, and cooperation is possible. Furthermore, under these conditions, agents need only communicate their situation rather than their intentions, thus reducing communication requirements.

Ginsberg examines the situation where a single, common goal is replaced by several local ones whose successful completion is contingent on how the others are completed. Rather than considering the agents' actions, he considers their decision procedures. A decision procedure is the process by which an agent chooses a *rational* course of action. These procedures must be such that all agents would have the same order of preferences when faced by the same choices.

The individual procedures are gathered together into a *collective decision procedure* whereby the agents make a joint decision. These decision procedures are referred to as "*unbiased*" if any shuffling round of the others' decision procedures does not affect the choice a given agent would make. In this case, the decisions are assumed independent, and the agents behave "*uniformly*". It is then possible to establish collective pay-off functions, and the likelihood of any given decision occurring.

The possible pay-offs are then determined and maximised, and all "*irrational*" decision procedures are eliminated; only those that optimise the pay-off are retained. To be "*rational*", an optimal pay-off is not necessary in all cases, but the pay-off must never be suboptimal. The remaining decision procedures will be "*globally rational*", and result in uniform, *unbiased* decisions, referred to as "*uniform rationality*" by Ginsberg.

If the agents' decisions are not independent, which occurs in the "prisoners' dilemma", the agents can arrive at globally optimal decisions by having knowledge about each others' strategies. This introduces the notion of "*common rationality*", where agents that interact are equipped with matching decision procedures. This resolves the prisoners' dilemma nicely. Each prisoner knows that the other will use the same decision procedure, and will make the globally optimal choice, which in this case will be to not confess, and both prisoners will get one year. Thus *common rationality* makes cooperation possible.

However, *common rationality* fails when the optimal choice requires contrary decisions in a symmetrical decision matrix. This is known as an "*ambiguous*" situation, and is illustrated in figure 2.4.2. Here, both agents are faced with an identical choice. However, the optimal decision requires that one agent choose 'A' while the other choose 'B' in order to maximise the global pay-off. Common rationality, with its matching decision procedures, does not allow for this. Therefore, it can handle only situations that are not ambiguous. However, in a totally disambiguated decision space, it would produce a unique, *globally rational decision*. If this is so, agents would not even have to be aware of the interdependence of their choices in order to cooperate. They would only have to agree upon a globally disambiguated pay-off function, after which no further communication would be necessary.

		Agent 2	
		A	B
Agent 1	A	0	1
	B	1	0

To obtain an optimal pay-off in this situation, agent 1 must choose 'A' and agent 2 must choose 'B', or vice-versa. However, both agent 1 and agent 2 face exactly the same situation, and have exactly the same decision procedure. Thus their decisions will be identical, and both must choose either 'A' or 'B' together, which is a suboptimal choice.

Figure 2.4.2: An ambiguous situation

Although Ginsberg provides a formal proof that cooperation is possible when agents have *common rationality* and work with *disambiguated* pay-off functions, applying his theory to real life problems is not straightforward. Determining *globally rational collective decision procedures* that are *disambiguated* for all possible cases is a highly problematic venture. It is easier to state in theory than to accomplish in practice, particularly when the decision procedures are heuristic in nature. An alternative way of looking at the problem of cooperation is to examine what knowledge the agents have, what knowledge they require, and how they can obtain it "intelligently".

In a distributed system, the purpose of communication is to augment available knowledge by giving the agent information that it will eventually use to choose its actions. An agent may derive additional knowledge by observing the behaviour of other agents in the system. By knowing certain facts about the other agents, the rationale behind their actions, and the communication policy employed by the observed agents, an agent can augment its own knowledge and make better decisions. This approach for conveying information among a group of agents is known as a *"knowledge based protocol"* [24]. Here, the act of communication or its absence, can in itself convey information that will change the state of knowledge in a system. The success of a group's collective action may very well depend on the state of knowledge that the group as a whole is capable of achieving.

Knowledge here is defined as some fact which is known to be true. The different states of knowledge that an individual or a group can achieve are given below, with each a subset of the succeeding one.

1. An individual has ***implicit knowledge*** of some fact if, given the relevant information it could derive that fact.
2. Someone in the group knows the fact.
3. Everyone in the group knows the fact.
4. Everyone in the group knows that everyone knows the fact. This can be repeated according to the number of people that know that everyone knows ... some fact.
5. The fact is ***common knowledge***. Everyone knows it, and everyone in the group is aware of this.

Figure 2.4.3: Different states of knowledge.

In a system where a certain amount of common knowledge is available, where assumptions can be made about how others would act, and where it is possible to establish what others know and possibly don't know, this knowledge can be successfully employed to change everyone's state of knowledge. The "Cheating Husbands" example [25] of figure 2.4.4 illustrates this. However, achieving common knowledge makes two basic assumptions about the communications medium and the environment: (1) all messages are guaranteed to arrive; (2) all members are capable of acting simultaneously. These conditions cannot always be guaranteed. However, not all members may require common knowledge, and it is possible to define time windows in which actions are considered simultaneous.

The authors go on to examine the "Cheating Husbands" story through various scenarios which modify the story's outcome considerably [25]. Where applicable, this type of protocol could reduce communication requirements significantly. However, it requires an in depth understanding of the nature of the problem, of the participating agents, and of the communications medium. The participants must have the necessary *"self-awareness"* to be able to reason about their interactions in order to derive the maximum amount of knowledge from them. The problem of managing such a system may prove to be as ill-defined as that which the system addresses in many domains. The sophisticated type of control necessary will require a formal structuring of the system that can be provided by organisations.

Cheating Husbands

The queens of Mamjorca have always opposed and actively fought the male infidelity problem. One day the queen resolves to deal with it definitely. She summons all women, both married and unmarried, and makes a declaration. Using common knowledge, the declared facts, and a known protocol, the women are able to eliminate all unfaithful husbands with almost no communication concerning the subject.

Common knowledge:

All women are perfect reasoners.
All women are always obedient to the queen.
All women hear all shots that are fired.
All queens are always truthful.

Facts declared by the queen to all women:

One or more of you have unfaithful husbands,
Before being told, you do not know about your own husband's fidelity.
However, each of you knows which of the other husbands are unfaithful.

Constraints imposed by the queen:

You shall not discuss this with anyone.

Action that the women will take:

You will shoot your husband at midnight of the day you discover his infidelity.

Having heard this, the women went home, and watched and listened. Thirty-nine silent nights went by, and on the fortieth, shots were heard.

How many unfaithful husbands were shot?
How many unfaithful husbands were there?
How did the cheated wives learn of their husbands' infidelity?

The solution to this is simple. The wives know that there is at least one unfaithful husband, and that they know about all unfaithful husbands apart from their own. If there is only one unfaithful husband, his wife will not see any. She will immediately conclude that it is her own husband that is unfaithful, and shoot him at midnight. If there are two unfaithful husbands, each cheated wife will see one, who is the other's husband. She will listen for shots that night. If no shots are fired, she will conclude that the other cheated wife also sees some unfaithful husband, in this case her own. Both wives will shoot their husbands the following midnight. This is true for any number of unfaithful husbands. The cheated wives will discover their own husbands' infidelity after 'k' silent nights, with 'k' being the number of unfaithful husbands they themselves see.

Figure 2.4.4: Cheating Husbands

2.5 Organisations of Intelligent Agents

The idea of viewing a DPS system as an organisation of agents is attractive because it coincides with the organisational model used by people for resolving complex problems. An organisational model provides a convenient way to describe the agents' problem solving abilities, and to define the communication policies which the agents incorporate into their local decisions. This type of *common knowledge* is necessary if agents are to reason about their individual activity in relation to that of the other agents. However, the fact that it is common knowledge, shared by all agents, makes organisations costly to form. As a consequence, organisations must remain relatively static once they are established. This may conflict with the goal of providing agents with the ability to react flexibly and take local initiatives -- which are essential properties of an "*opportunistic*" approach to problem solving. The organisations must be structured in order to provide a framework that does not inhibit the desired degree of "*opportunism*".

The organisations described in section 2.1 constitute the basic forms that can be adopted. Within these organisations, three basic roles prevail: that of "*peer*", "*subordinate*", and "*director*" [26]. Any of the proposed organisational forms can be realised using these roles, as illustrated in figure 2.5.1. Agents can assume multiple roles, depending upon the number of agents with which they interact, and their relation to them. More complex organisational forms, such as *multilevel hierarchies*, and *decentralised* or *market organisations*, are constructed in this way. Responsibility for decisions is implicit within the assignment of roles, while an agent's influence on others is determined by the method used for making decisions. The resulting organisations will behave very similarly to their human counterparts, displaying many of the same advantages and handicaps.

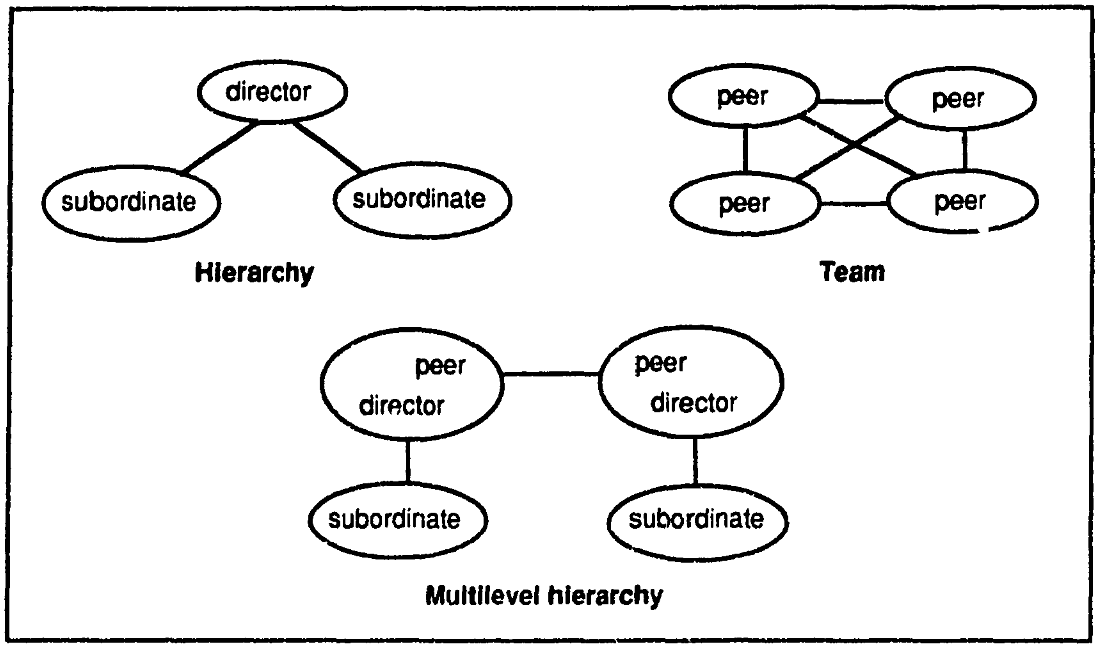


Figure 2.5.1: Organisational roles within basic organisations.

The modes of decision making remain the same -- decree and consensus, with an implicit correspondence between role and mode: directors "*decree*" while peers "*negotiate*", and subordinates "*comply*". This relatively strict definition produces various different types of interaction, depending on the nature of the agents' responsibilities and communication policies. The combination is instrumental in determining to what degree an individual agent is independent, subservient, or relies on the group.

The agents' responsibilities and communication policy establish (1) what information is communicated, (2) who takes the initiative, and (3) what the expected response is. The different possibilities for each category are outlined below. The complexity involved is highly variable, according to the amount of reasoning associated with it. This may be considerable if a knowledge based protocol is used.

(1) The information communicated includes everything that the agents may wish to convey to each other.

(2) The agents (a) may volunteer information, (b) they may provide it upon request, or (c) they may systematically produce it. The agents may send information to specific individuals, or broadcast it to the group or a subgroup.

(3) The agents may respond (a) immediately, (b) at the first opportunity, (c) eventually, or (d) never, depending upon their responsibilities and priorities. The agents may employ a mixed approach, or change it as the solution evolves.

If communications involves knowledge, the agents must be aware of each other's roles, and of the role communication plays in the system. They must understand what they can expect from the others, and what the others expect from them. Agents may produce solutions, propose hypotheses, pass constraints, define goals, or signal that radical events have occurred. The agents may expect immediate responses, an eventual response, or the possibility that no answer is sent. They may wish to order their messages, to respond differently under different conditions, or to modify their relation with others. All of this requires an expanded definition of the agents' organisational roles and expectations.

A large number of factors are involved in determining an appropriate organisation for a given problem. The bounded rationality [11] of an agent places a limit on any agent's capabilities. The bottleneck effect [19] limits the capabilities of a director, while consensus may introduce significant delays. If agents are given too much independence, their work may diverge, while too much subservience or compromise will result in less opportunism. The cost of communication and the need for asynchronism must also be considered. The timely communication of relevant information will have an impact on problem solving quality. At the same time, thought must be given to the consequences of disruption, distractior and local idleness.

As stated earlier in section 2.1, the forms of decision making available to an organisation consist of *consensus* and *decree*. The choice between decisions by consensus or decree will largely depend upon which is the most advantageous. Groups have a broader perspective, consider more alternatives, and are more likely to produce decisions that satisfy each member's view. Furthermore, each member's activities and goals become visible to the group. However, groups also work more slowly, may involve considerable compromise, may be dominated by certain members, and may inhibit the individual's ability to react quickly when opportunities present themselves. On the other hand, although individuals are able to react more quickly and opportunistic-ally, their efforts may be misdirected because they lack the broader perspective available to the group, and the more detailed aspects of specific situations visible to others. A balance is often necessary between reliance on the group, and an individual's independence, or subservience to others.

The specific form an organistion of DPS agents assumes is highly problem dependent. Both the requirements of the problem and the designer's perception of them are deciding factors in the organisation's structure. This raises an interesting question. Given that an expert system is generally developed in an incremental fashion, will its organisational structure evolve in the same way? It is quite possible that as the DPS system evolves from a skeletal prototype to its full stature, the organisation's needs will evolve similarly. Unless the problem's needs are fully anticipated from the beginning, it is quite possible that a chosen organisational form will become inappropriate as greater complexity is added to the problem solving process.

There are many open questions related to using organisations. Is it possible to establish criteria by which the "goodness" of an organisation can be judged? What is the best way of developing an organisation for a given problem? Are there any "best" organisations for certain types of problems? And perhaps most critical of all, can organisations provide a cost-effective way of coordinating activity, or will their overhead prove too heavy a burden?

2.6 Evaluation of a DPS System

Evaluation of a DPS system involves all the difficulties associated with evaluating an expert system, magnified by the problem of evaluating a coordination strategy in a distributed environment. Coordinating a conventional distributed system is an ill-defined problem in itself [19]. Monitoring distributed systems is a research area of its own. Observing how a given coordination strategy operates will introduce delays that will change the way in which it performs. This will make its verification and validation extremely difficult. Yet it is essential if we wish to draw even tentative conclusions about the performance of different coordination schemes.

Evaluation of a DPS system is not only concerned with validating the individual agent's competence, but must also ensure that the organisational structure used permits the agents to exert their competence effectively. A critical aspect is the number of factors that affect the system's operation. The agents' roles, their intelligence, and the communication policies they employ are all involved, making it difficult to assign credit or blame to any specific aspect. Good communication policies combined with "dumb decisions", or good decisions combined with bad communication policies will both have a negative impact on overall performance. Similarly, a good organisation applied to an unsuitable problem may perform very poorly for non-obvious reasons. Evaluating organisations and pinpointing their weaknesses is in itself a difficult problem.

Evaluating the relative merit of different organisation schemes is difficult because they all perform differently according to the problem's complexity. The different organisational forms described earlier are all designed to handle different types of complexity (size, interdependence, and uncertainty), in different types of environments (fixed or rapidly changing). In certain problems timeliness is not critical, while in others it will be the decisive factor in determining the solution's acceptability. These differences make it difficult to draw comparisons between the possible organisations at large. Some form of categorisation will be needed to establish the suitability of the different organisations for the different types of problems.

Chapter 3
A Comparative Study of Two Testbeds for DPS

The primary objective in developing a DPS testbed is to provide an environment in which it is possible to explore how different organisations of agents perform. The nature of the problems addressed, the number of factors involved in their solution, and the possible sources of unexpected interactions in the overall system make experimental work an essential aspect of DPS research. Testbeds are necessary to observe, validate, modify, expand, and most importantly, comprehend the implications of the models that have been proposed. They are particularly important if we wish to judge the relative merit of these models, and establish what kind of problems they are suitable for. All the coordination schemes described in the previous chapter evolved in conjunction with specific testbeds designed to solve specific problems. The DVMT testbed [17,21, and 29-33] is among those which have contributed significantly to DPS research.

We have carefully studied and assessed the virtues and shortcomings of the various testbeds described in DAI literature. The characteristics of the problem and the facilities available to the user both have a significant impact on the range of DPS issues for which the testbed is suitable. We propose a testbed that is based on a game called *Distributed Blackbox* (DBB). We compare DBB to the well known DVMT testbed, and bring out the advantages of using DBB. Knowledge engineering, verification, and performance evaluation of the underlying expert systems involve less labour in DBB than in DVMT, while most of the richness of the problems to be solved is still retained. The nature of DBB provides a scenario in which a large number of issues related to coordination and cooperation can be directly observed.

3.1 Desirable Characteristics of the Testbed

The chosen testbed must reconcile our research objectives with our resource constraints in order to be viable. The testbed environment will establish what issues can be studied at what cost. A low cost and the need for incremental development are crucial factors in determining whether a testbed is feasible. The nature of the testbed problem will affect both the knowledge engineering task, and the range of experiments that can be run on the testbed. Care in its selection can significantly reduce the amount of effort involved in the testbed's development and subsequent usage. The problem must cater to a wide range of DPS issues and not require excessive analysis to be useful. This will promote rapid understanding of its operating principles, and the underlying issues it addresses.

Researchers normally employ one of the following approaches to support their propositions: (1) Model the problem under study using abstract mathematical structures and theoretically study certain selected aspects; (2) Simulate the behaviour of the system at an appropriate level of abstraction and study the system based on the simulation results; or (3) Construct an experimental prototype of the system and study it in a controlled environment. These approaches are not mutually exclusive, and each has its own merits and limitations.

Mathematical structures often reveal the attainable limits of the model, but generally work in ideal universes that are difficult to realise. Simulations reveal what kind of behaviour can be expected under the different conditions that arise during the model's "normal" operation. However, constructing the simulation may involve considerable work, and verifying its operation must validate the simplifying assumptions that it makes. Prototypes are useful because they provide a more detailed and realistic view of how the model behaves when the characteristics of a problem or class of problems are considered.

An experimental prototype requires a test environment which normally has several conflicting requirements: the test environment should provide rich insight into the system but not be too complex; the data required by it should be easy to obtain but not be too trivial; operating the prototype with the test data should reveal "more than" obvious behaviour of the system, but it should not be too expensive; implementing the prototype should not require an undue amount of resources (manpower, equipment, tools, time, and space) but at the same should reveal those aspects of the system which cannot be studied through simulation or abstract analysis.

Choosing a sample problem that the prototype can use to examine DPS issues is difficult. Any problem that will encompass the issue of cooperation among multiple agents must possess sufficient complexity to justify the need for cooperation. The complexity of the problem should be easily controlled, allowing agents to be observed in many different situations. The problem domain must be rich in possible decompositions, allowing agents to participate in a variety of "meaningful" organisations. The problem must be amenable to the use of different solution strategies that entail different coordination schemes and communication policies. In addition to this, the prototype system must allow for easy modification of the agents' roles without requiring extensive modification to each agent's knowledge base and heuristics. Finally, the basic problem must be simple for users to learn, as their primary interest is not the problem itself, but the study of cooperation among the agents.

The difficulty encountered when choosing an appropriate testbed is the immense effort involved in developing any system that displays the required degree of complexity. This complexity will be reflected in the nature of the searches and lines of reasoning used to solve the problem. The way these are structured will directly affect what information is communicated, how it is exchanged, and the degree of opportunism possible during the solution process. The different situations that can occur in the overall system are a direct consequence of all these factors. The observable situations will determine what DPS issues the testbed is suitable for, and the scope of the study. Ideally, the testbed should provide a platform in which a range of issues can be examined.

The testbed's research objectives add to its design requirements. In addition to having the basic intelligence necessary to solve the problem, the testbed must have facilities that allow the user to observe its operation and measure its performance. These are critical to both validate the system's problem solving abilities, and evaluate how well a given organisation of agents operates in comparison to others. The amount of time it takes the user to familiarise himself with the system's operation, and the ease with which he can run experiments, analyse results, and modify the system, will all have a major impact on the project's progress.

3.2 Advantages and Limitations of Existing Testbeds

Although the desirable qualities of a testbed are easy to enumerate, finding a problem that satisfies them is not as simple. Few problems exhibit the desired levels of simplicity, and complexity in coordination requirements. Furthermore, It is often difficult to establish what issues the testbed is suitable for until a significant amount of effort has been invested in the problem's analysis. The difficulty also resides in acquiring the relevant, domain specific expertise. And even when suitable problems are found, they often contain more complexity than necessary, needlessly adding to the design task. Many of the prototype systems attempted thus far in DAI have been too complex, restricting their portability to other environments. Some of the problems that have been attempted are outlined below in figure 3.2.1.

Hearsay-II: A natural language speech understanding system that responds to spoken commands in a restricted domain [10].

Crysalis: Determines the structure of proteins given their amino-acid sequence and x-ray diffraction data [10].

OPM: Models human cognotive processes in an errand planning scenario where errands have different priorities and require goning to different places in some "town". The plan must optimise displacement and time in a "travelling salesman" type problem [10].

Scene Understanding: Identifies and labels objects in an aerial photograph of a suburban area [10].

Contract Net: Air or ocean surveillance with negotiation among agents using contracting mechanisms in a market form of organisation [27].

DVMT: Land traffic surveillance using a functionally accurate cooperative (FA/C) approach to coordination [21].

Negotiation In Air Traffic Control Systems: Modifying flight plans among planes entering a crash situation. The concerned planes form a team, and negotiate a crash avoidance plan [28].

"Reds and Blues": A scenario in which four "blue" agents attempt to encircle a signal emitting "red" agent on an infinite, two-dimensional grid. The agents have set moves, and all agents move in a cycle. The red agents are structured into various organisations for performance comparisons [26].

Figure 3.2.1: Prototype problems attempted in DAI

As can be seen, the possible application domains are quite diverse, and include many current research areas in AI. The systems mentioned above introduced many of the ideas that characterise current DPS approaches. The difficulties that they encountered constitute an important source of inspiration for subsequent systems. The first four systems decompose knowledge into knowledge sources, and examine the resulting problem with successively more complex coordination mechanisms. The relatively tight coupling they employed made it difficult to extend their ideas into a distributed environment, introducing the need for more sophisticated local control, and the concept of cooperation. The Contract Net, DVMT, and Air Traffic Control are the testbeds in which *negotiation*, *FA/C*, and *multiagent planning* were developed as coordination schemes, while "Reds and Blues" examined the formali …s of organisations.

What is noticeable about these problems is that many involve considerable domain specific expertise in areas which are considered distinct application fields in AI (natural language understanding and vision). Some of them involve realtime monitoring. The need for domain specific expertise has the obvious drawback of increasing the complexity of knowledge requirements, while monitoring has the disadvantage of requiring realtime signals, or an effective simulation of them. Modeling the data will itself become a complex problem if we wish to ensure the data's integrity, and provide a rich variety of situations. This still leaves a wide range of possible problems from which one can choose.

The disadvantages of choosing a real life problem are obvious. Its implementation would involve a considerable amount of knowledge engineering, understanding its operation would require domain specific expertise, and once completed, experimenting with alternative designs would be both expensive and difficult. Thus, we are left with the option of either designing a problem to satisfy our requirements, or simplifying a real problem. There are certain difficulties associated with both options. Inventing a generic problem that is complex, nondeterministic, and internally consistent is a major creative effort. The alternative of simplifying a real problem carries some inherent drawbacks. For one, the original problem must be understood in order to obtain an accurate simplification of it. For another, there is always the danger of oversimplifying the problem, or of contriving it to fit our assumptions about how it should behave.

Instead of going for either alternative, we have combined the two approaches by adapting a game to support multiple cooperating players. Games have certain properties that make them interesting candidates for study. They provide a structured task in which it is easy to measure success and failure. They generally involve a highly stylised representation of some global situation in which more than one element is involved. Although the elements' interaction may be complex, their individual behaviour is determined by a small set of well-defined rules. Knowledge about a game is much easier to acquire than the expertise of a real world expert. However, the chosen game must be neither too simple as in "Reds and Blues", nor too complex as in chess. The only remaining difficulty is that of finding a game that involves cooperation rather than competition. Competitive games constitute a distinct class of AI problem in which we are not currently interested. A cooperative game is obtained by decomposing a one player puzzle called "*Blackbox*" into a game that can be solved by several players: "*Distributed Blackbox*".

Although a large number of prototypes do exist, few if any, have ever been extended to examine the full range of issues that their operation involves. Many face unresolved obstacles to making this transition, and some of them have reached the limits of their utility as testbed environments. The choice of testbed must reside on the issues for which it is intended, and the ease with which it can be developed and employed. The Distributed Vehicle Monitoring Testbed (DVMT) is a well documented testbed that has provided much insight into the issues involved in DPS systems. Distributed Blackbox (DBB) is the alternative that we advocate here. Although DBB does not display the same type and degree of complexity as DVMT, we believe that DBB will permit us to study many interesting research issues within a "shorter" time period.

3.3 DVMT: The Distributed Vehicle Monitoring Testbed

The *Distributed Vehicle Monitoring Testbed* (DVMT) simulates a network of problem solving agents so that approaches for DPS can be developed and evaluated. It addresses the problem of monitoring a two-dimensional geographic area to identify all passing vehicles and track their displacements. DVMT is composed of a set of semi-autonomous agents, and employs a *functionally accurate cooperative* (FA/C) approach for coordination. The agents work together to solve the global problem by individually solving interacting subproblems and integrating their partial solutions to arrive at an overall solution.

DVMT collects data via a set of distributed sensors that capture acoustic information about the type and velocity of vehicles passing within their range. The problem has a natural spatial distribution, and involves a considerable amount of data that must be abstracted to produce a dynamic map of area traffic. Vehicles are recognised based on the frequency classes of the sensed data. A two-dimensional square grid represents the monitored area. A hypothesis as to where vehicles are located is represented at several levels of abstraction. Sensor signals are aggregated into signal groups, which are combined into vehicle types. The displacement of a recognised vehicle at discrete time intervals produces a vehicle track, which has some spatial relationship to the other vehicles that have been detected. Hypotheses about vehicles are interdependent in both space and time, as there is an overlap in the sensors' range as the vehicles move into physically adjacent areas. (Figure 3.3.1).

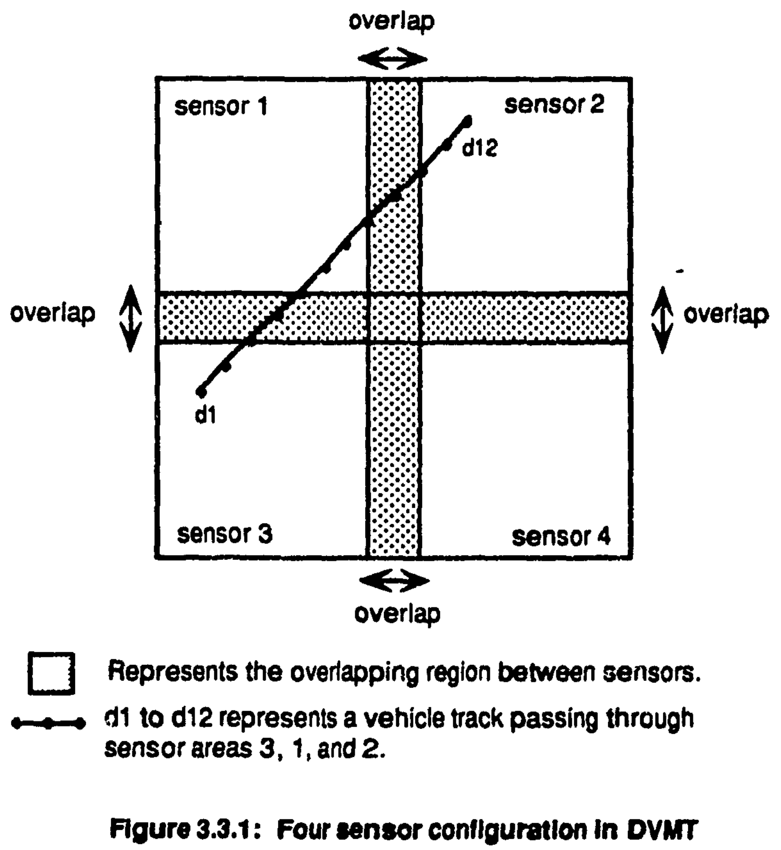

overlap
sensor 1
sensor 2
d12
overlap
overlap
d1
sensor 3
sensor 4
overlap
Represents the overlapping region between sensors.
d1 to d12 represents a vehicle track passing through sensor areas 3, 1, and 2.

Figure 3.3.1: Four sensor configuration in DVMT

In DVMT the problem becomes more complex due to the following facts: communication between the agents is subject to the random loss of entire messages; sensors can fail to detect a signal; sensors may detect a non-existent signal; a vehicle may be incorrectly identified or located; sensors, agents, or communication channels may fail without warning. The problem of locating vehicles is *commutative*, as the map of vehicle tracks is incrementally constructed and revised. The coordination problem is *non-commutative*, as the solution's goodness is measured in terms of its timeliness, and is detrimentally affected by the choice of inappropriate activities which irreversibly consume limited resources and time.

The global solution is derived from the partial hypotheses constructed by each agent. Uncertainty and errors in the input data require that agents correlate their partial hypotheses in order to reinforce and confirm intermediate results, or to eliminate incompatible alternatives. Agents exchange data, or control information called *goals*. The information exchanged describes hypotheses, or the intent to create hypotheses, and serves to resolve uncertainty and coordinate activity. Agents must perform a correct ordering of local activity to obtain an acceptably accurate and timely solution. Note that if perfect input data were available, the agents are designed so that they are capable of deriving their individual solutions independently.

The hierarchical nature of the hypotheses combined with the temporal and spatial aspects of the problem provide a wide range of problem decompositions appropriate for a variety of organisational structures among the agents. In addition to varying the accuracy of the agents' individual intelligence, their functional abilities can be easily modified to experiment with different problem solving strategies. The problem's complexity is controlled by varying the density of vehicles, the similarity between their patterns, and the error and uncertainty in the input data. The complexity of the problem will affect the system's computational and communications load, and the amount of uncertainty the system must handle.

For the sake of prototyping, the problem is simplified in various ways. Both the network and the agents are simulated, and input data that includes belief values is used to simulate the signal analysis that would otherwise be performed by the sensors. Vehicles are represented by a small number of frequency classes. The agents' problem solving intelligence is statistically simulated. The agents' ability to detect local consistency and inconsistency among hypotheses is based on the belief values assigned to the hypotheses. Agents generate plausible hypotheses; an oracle with access to the actual solution modifies their belief values in order to reflect the level of intelligence of the agents. Thus it is possible to vary the agents' problem solving intelligence without modifying their basic knowledge.

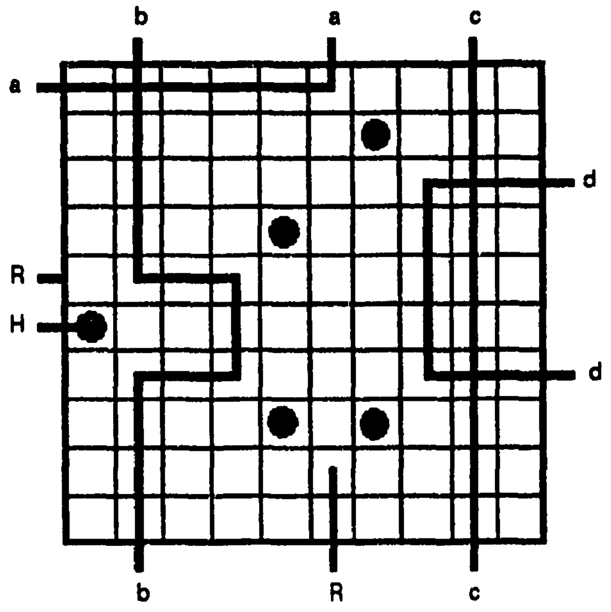
The DVMT testbed environment provides a number of facilities for observing the system's behaviour, manipulating intermediate operations, and measuring performance. The evolving solution can be compared to the correct one at any stage of processing. The potential effect of message transmission can be examined during execution. Performance is measured according to the accuracy of information, the time required to derive a solution, the amount of interagent communication involved, and the network's robustness in the face of communication errors. In the experimental prototype, front-end subsystems have been added to reduce the effort required to create the desired input data, specify the system's architecture, and analyse results.

The issues examined by the researchers of DVMT include how the communication policies and coordination schemes affect the system's performance. *Disruption*, *distraction*, and *local idleness* emerged as three critical issues. The authors' results reveal that the performance of different organisations of agents varies significantly according to the amount of inaccuracy and uncertainty in the input data and communications channels that must be dealt with.

The designers of DVMT report that they would have liked to explore larger configurations composed of 10 to 20 agents. However, the time required to set up and run these experiments was prohibitively long, and only a few test cases were run. This was due to the complexity involved in specifying the system's structure, and creating the corresponding input data with the desired degree of uncertainty and error. The manpower expended in this project is estimated at 15 to 20 man years.

3.4 DBB: The Distributed Blackbox Testbed

Blackbox is a game that consists of a square, two-dimensional grid in which a number of balls are hidden. At the beginning of the game, the player is informed of the number of hidden balls. The objective of the game is to find the locations of the hidden balls by firing beams into the grid along its sides, and observing how the beam trajectories are affected by the hidden balls. A beam, when it encounters a ball, is *reflected* (turned 180^0), *deflected* (turned 90^0), or *absorbed* by the ball when it encounters a ball head on. (See figure 3.4.1). A player plays the game interactively with a computer. He fires a beam, and the computer reveals where the beam emerges from the grid, if it is not absorbed. The player hypothesises about the possible locations of balls from the information he obtains from the fired shots, and then chooses the next beam to fire. The player's objective is to determine the location of all hidden balls with a minimum number of shots.



a - c: Deflected beams.
R: Reflected beams.
H: Absorbed beams.

Figure 3.4.1: Example of beam behaviour in the game of Blackbox

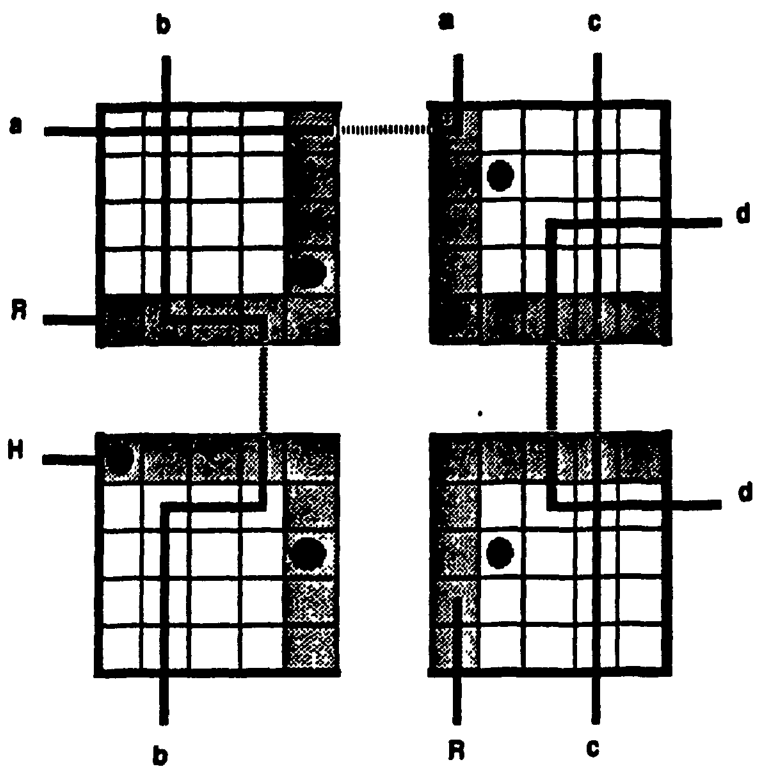
One way of measuring the player's expertise is to count the number of shots fired in order to find all the hidden balls for a given game configuration. The player's score provides a weighed sum of the number of shots taken. Beams that are absorbed or reflected are valued at one point, while beams that exit from the grid are assigned two points. This corresponds to the number of edges which the beam provides information about. Other performance measurements can be derived based on the amount of time it takes the player to complete a game, and the number of errors in his final hypotheses about board configurations. These measures can be averaged for a series of games to give an average measure of the player's problem solving performance. In the computerised version of the game, the human player is replaced by an expert system or agent. Unlike human players, these agents will play consistently if they repeat the same game. Under these conditions, changes to the solution strategy will be easy to evaluate by comparing the observed measures for a series of representative games.

Although the problem has a finite solution space, its size (2.4×10^9 possible board configurations for a 10×10 grid containing six hidden balls) makes the search involved non-trivial. The search's complexity varies according to the grid size, the number of balls, and their relative locations on the board. The basic solution process is commutative. The solution is developed incrementally by applying knowledge about the relation between beam behaviour and grid contents to construct a board configuration that produces the observed beam behaviour. The objective of determining the correct configuration with a minimum number of shots introduces a non-commutative aspect to the problem, as the utility of any shot depends entirely on the current state of the solution.

Different strategies must be employed as the game progresses from start to finish in order to make the search tractable. At the beginning of the game, the amount of uncertainty associated with any beam makes a "*least-commitment*" approach the most suitable. With this, tentative hypotheses are formulated, but no commitment is made until their validity is proven. As the solution evolves, the amount of uncertainty decreases, making a more exhaustive search feasible. At this stage of the game, an attempt is made to integrate the tentative hypotheses from several beams in order to define the contents of the region they pass through.

Distributed Blackbox (DBB) is a modified version of this game. In DBB, the game grid is divided into four equal quadrants, with each quadrant assigned to a different player. Thus the quadrant of each agent consists of two external edges, and two internal edges which border the quadrant of another agent. The internal edge constitutes an overlap region, since the contents of this region will affect trajectories in either quadrant. Agents may only fire beams into the grid along their external edges. All information about their internal edges must either be derived, or obtained from the agent's neighbour. (Figure 3.4.2).

The formulation of hypotheses representing ball locations and beam trajectories requires that agents exchange a variety of information for the following reasons. A single beam may pass through several quadrants in the course of its trajectory. Explanation of beam behaviour involves multiple partial hypotheses distributed among the agents and interrelated by the beam's possible transit points into adjacent quadrants. Agents are also interrelated by the overlapping region between their quadrants. The number of balls that remain to be located provides additional strategic information. The need to share all this information when deriving non-trivial hypotheses, obliges the agents to communicate and cooperate with each other. Communication implies that the agents may have an inconsistent view of this information.



▬ External boundary along which agents can fire.

▨ Shaded area represents overlapping region shared by adjacent agents.

Figure 3.4.2: Distributed Blackbox (DBB)

The order in which shots are fired affects the aggregation of knowledge and progress of the game. The order in which unresolved trajectories are analysed, and the strategies that are applied, have a similar effect on the solution's progress. Agents fire shots to acquire additional information about their individual quadrants. At the same time, these shots must be planned at the global level to minimise the total number of shots taken, and optimise the information obtained from each shot for all agents. A decision at this level is non-commutative. The organisational structure employed will have a direct impact on the ordering of shots and progress of the solution.

As knowledge about the quadrants and the firing of shots is distributed, there is no compelling reason to centralise the planning process. This inherent distribution provides opportunity for studying various models of cooperative behaviour, and of observing how the agents cope with uncertainty. The ability of an organisation to maintain a global focus for agent activity will depend upon what information is exchanged by the agents, and how often. The agents' consistency and symmetry, and the ease with which they can be scaled, will facilitate the task of comparing the performance of different organisations. Any variations in the observed measures can be directly related to the communication policies and coordination schemes employed.

3.5 A Critical Comparision of DBB and DVMT

The problem dependent nature of DPS systems makes it difficult to integrate the results obtained by different research groups. As a consequence, the benefit that could be obtained from others' experience is generally not exploited to its full potential. Because these systems address non-deterministic AI problems, there is no absolute measure for evaluating their performance. Each system has its own criteria for determining the validity of initial premises, the expected accuracy of its final solution, and an acceptable response time. Although it is easy to compare the performance of two systems that address an identical problem, comparing the performance of systems that address completely different problems entails a subjective evaluation of the relative complexity of their individual domains.

The problems addressed by both the DVMT and DBB system involves situation assessment and planning. They both employ a geographic decomposition of the problem space, with a further functional decomposition that introduces alternative levels of abstraction in the hypotheses. The basic nature of both problems is commutative, whereas the control aspect is not. Cooperation is necessary at all levels of hypothesis formulation in order to deal with uncertainty in the problem. As a consequence of this, there is a large amount of bi-directional interaction among the agents and between the different levels used to represent a hypothesis.

Planning within DVMT is principally concerned with ordering activity to achieve timely solutions. Agents are subject to their sensor data, and do not control the collection of information. Planning knowledge within the system is related to the problem's distribution and the type of uncertainty it must handle. Timeliness will be set by the sensing rate, which will also determine the amount of information available to the agents at any time. Dealing with uncertainty in this context implies balancing the sensing interval with the corresponding signal processing requirements to obtain some desired degree of accuracy.

Planning in DBB must deal with uncertainty due to incomplete knowledge about the problem state when ordering the activity of the agents. Each activity involves a choice of strategy that is applied according to what stage the game has reached. The agents in DBB have the choice between firing shots, and analysing unresolved beams. In the first case, they must determine which shot to fire. In the second, they must choose an unresolved beam to analyse. In both cases, they must deal with incomplete knowledge about the board. This incompleteness is a property of the problem itself, as opposed to noise in the input data. It is resolved by the choice of activity made by the agents.

The agents complete their knowledge about the board by firing shots to acquire additional data. The agents are actively involved in determining when they require more information, and in seeking it by choosing what shots are fired. In this context, handling uncertainty involves determining what is known, what isn't known, and the best way of acquiring the missing information. This requires greater introspective reasoning by the agents in DBB than in DVMT about how the global solution is evolving. Timeliness here is not a constraint imposed by the input rate, but rather a factor that influences the sequence in which shots are fired, the order in which information is aggregated, and the total number of shots taken.

Communication noise, error, and failure have a different effect on the reliability of hypotheses in DBB and DVMT. In DVMT, the monitored vehicles are independent from each other; the only interrelationship between vehicles is that they cannot occupy the same location at the same time. Communication noise and error concerning the vehicles' location can be resolved by interpolation and prediction, supported by the arriving sensor data and established hypotheses. If an agent fails, it can recover at any subsequent time by simply resuming the sensing of data, whose degree of uncertainty is not affected by the failure. Furthermore, the degree of uncertainty in the other agents' input data remains invariant, whether an agent has failed or not. The same is true for communication noise and error.

In DBB, beam trajectories are interdependent, as they must coexist on the grid. Establishing a trajectory hypothesis relies on the hypotheses that have already been established.

Any errors introduced by communication noise and error will be propagated throughout the evolving solution. Detecting and correcting communication errors is more difficult in DBB. The failure of an agent implies that knowledge about the region it represents will remain incomplete, and the surviving agents will be obliged to search for alternative plans that do not require cooperation with the failed agent to complete their own knowledge. They will accomplish this by choosing and firing an alternative sequence of shots. Thus, although failure reduces the choice of activity available in both systems, in DBB the agents devise alternative plans to complete their hypotheses, whereas in DVMT they must simply accept a higher degree of uncertainty.

In the case of both DBB and DVMT, the need for cooperation between the agents exists. In DVMT, this occurs between two agents whenever some vehicle exists in the overlap between their regions. The agents have no influence over this, as it is dictated by the input data. In DBB, the nature of multi-quadrant beams requires that all agents involved interact in determining their trajectory, and agents can influence with whom they interact by their choice of beam. Where cooperation in DVMT is limited to determing the optimal order in which to interpret sensor signals, cooperation in DBB encompasses more than establishing the transit location of beam trajectories. The agents also cooperate to define the grid contents in the areas they share, to determine what search strategies are appropriate for the current stage of the game, and to choose shots that minimise the score and optimise the acquisition of knowledge. This range of choice provides a much larger arena for examining cooperation in DBB.

A highly significant difference between the two systems is the manpower required for prototype development. This difference is largely due to the fact that DVMT is a simulated prototype whereas DBB is a game. The knowledge engineering involved in constructing a DBB agent is much simpler. Rather than searching for external expertise, users can acquire it rapidly by playing the game themselves. Familiarisation with the system's operation is easy, as intermediate results are immediately visible in a form that is intuitively simple to grasp. Game configurations (the locations of balls) can be randomly generated, providing a large set of problem cases. In DVMT, using the system requires understanding of the underlying signal analysis task. The set-up of experiments is a tedious and error-prone task due to the need to specify input data that simulates sensor output with the desired degree of uncertainty [22].

Another area in which the two systems differ substantially is in the approach we have chosen for providing DBB with the flexibility it requires to assume different organisational structures. Within DBB, the agents' organisational roles will be explicitly specified. Rather than opting for a fully parameterised testbed environment such as DVMT, we have decided to facilitate modifications at the programming level. This decision was based on the fact that although DVMT had a large number of parameters that could be varied, the interrelation between these parameters made the specification of different organisations a highly complex task [22]. Instead, modification of the program structures of the agents in DBB is planned using an object oriented approach. In this environment, respecifying the structure of an organisation consists of specifying the interrelation and type of interaction between the agents and related "objects".

3.6 Research Issues for which DBB is Suitable

The nature of the problem in DBB gives us a scenario in which we can study a variety of DPS issues related to coordination, cooperation and organisations. Possible problem decompositions produce different communication requirements that can be satisfied in several ways. These give us situations in which agents can assume different roles that are realised through different organisational structures. This will give us the possibility of studying various coordination schemes and communication policies when applied to the same problem, which in this case is the same hidden ball configuration. Furthermore, the nature of the problem allows us to compare the performance of our system to that of human players. This ability to observe people playing provides us with additional information about the game, a rapid proving ground for our ideas, and a source of insight into what cooperation implies.

Although we have chosen a geographic decomposition for our initial design, the problem also supports a functional decomposition in which the choice of shots to fire, the choice of beams to analyse, and the actual analysis can be distributed. A second way of viewing the problem is to have individual agents assigned solution islands to develop and expand over the global board. With this, the most knowledgeable agent in the context of some specific subproblem assumes responsibility for solving that subproblem. A third approach would be to have agents modify the size of the quadrant for which each is responsible according to the amount of knowledge each has. Alternatively, agents could assume organisational roles according to what they know about the solution, or their individual work loads. These are but a few examples of the different ways in which the problem can be tackled. Within the geographic decomposition that we have chosen, there are several issues to deal with:

(1) When should an agent focus on its own quadrant instead of firing shots or analysing trajectories to help reduce the uncertainty faced by another agent?

(2) Deciding which shot to fire next. Should the decision be centralised, or decentralised, and in that case how should it be realised?

(3) What information should be shared among the agents? Should an agent's confirmed or tentative hypotheses be available to the others? What communication policy will be used for the exchange? Will the communication policy achieve a balance between *information overload* and the uncertainty faced by an agent due to the lack of information?

The organisation in which the agents participate will define how these issues are handled. Two obvious organisational forms for decisions concerning shots are a simple hierarchy and a team (figure 3.6.1). The hierarchy will centralise all pertinent information whereas the team employs some consensus process whereby each member arrives at a compatible decision -- a situation in which cooperation, and knowledge based protocols can be explored. The information exchanged in the course of trajectory analysis will have an equally significant impact on the solution. Various communication policies can be applied at this level. These policies will vary the amount of disruption, distraction, and local idleness that occur. The organisations can remain static for the duration of a game, or change according to how the beam trajectories lie on the board. It is also possible to allow agents to form two, three, or four member teams associated with analysing individual beam trajectories in a multiagent fashion.

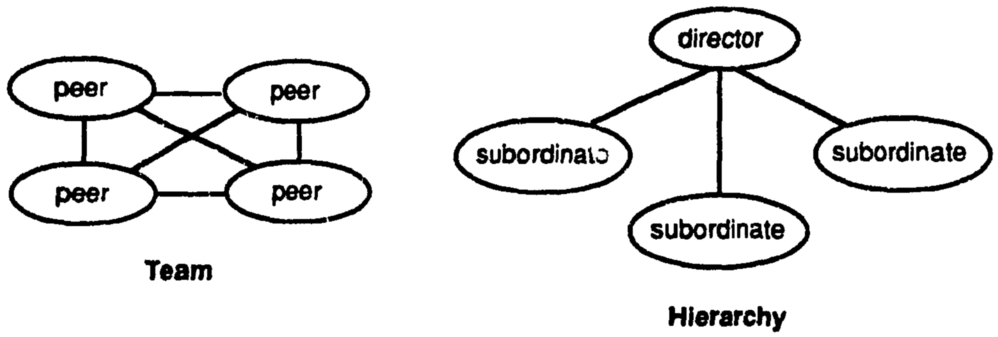


Figure 3.6.1: Possible organisations for DBB

The availability of a DBB "playground" for people is an added bonus to our work [34]. In this version of the game, four human players playing on seperate workstations replace the agents. Experiments have already been run with this version, using three different teams of players [35]. Although there are several ways in which a group of humans differ from a group of expert systems, these experiments have provided us with valuable insight into the nature of DBB, and may provide useful results for evaluating the performance of the expert system version of the game that we propose to construct.

Chapter 4
The Blackbox Expert System

Understanding the nature of the underlying problem is necessary for distributed problem solving. In this chapter, the design of an expert system for the Blackbox game is presented. The design employs a modular decomposition of the reasoning process to allow for the system's eventual extension into a multicomputer environment. The design includes facilities which make it easy to monitor and measure the system's problem solving performance for a large number of games. These measurements serve to evaluate the expert system's performance by comparing it with that of human players or other references.

The single expert version of Blackbox was developed on a Macintosh SE-30 in BNR Prolog. It is a 400k system that took approximately five months to develop. This time was more or less equally divided among the knowledge acquisition, implementation and evaluation phases of development. Although a certain amount of knowledge was acquired by watching others play, the major part was acquired by analysing the game rules and observing myself play. The system's implementation involved considerable intermediate evaluation and refinement of the knowledge. The system's final evaluation is based on measurements obtained from over 150 games. The observed measurements reveal the nontrivial nature of the game, and the limitations of a single agent.

4.1 An Overview of the Problem and Our Objectives

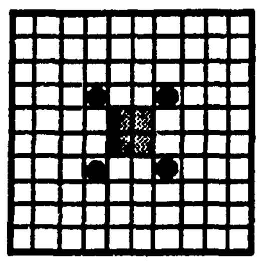
Our primary objective in constructing a single expert version of Blackbox is to obtain an expert system whose problem solving ability is thoroughly understood. By this we mean that when the expert system makes certain decisions during the solution process, we know how these decisions affect the solution, and why they were made. This requires that the expert system be observed and measured in a large number of different game situations. The ease with which game situations can be observed has a significant impact on the amount of time spent debugging, evaluating, and refining the system. Blackbox includes a facility that runs and measures a series of games for the user. This facility has permitted us to collect many days' worth of measurements at next to no cost in time to the user.

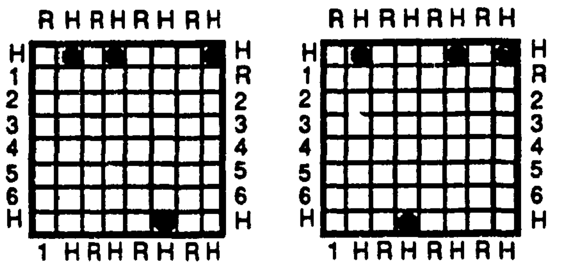
Our performance objectives for the expert system are to develop a system capable of finding "acceptable" solutions for "reasonably" difficult games. Our definition of "acceptable" and "reasonable" are based on the results obtained by people playing the same or similar games. With an *acceptable solution*, the score, the number of errors, and the time required to solve the problem are comparable to that of people. A *reasonably difficult game* is a game in which the solution is not found by random chance, but requires some thinking on the part of the player. If the player's problem solving skill is largely dependent on luck, his performance will be directly related to the game boards he plays rather than his problem solving ability. The game must contain a minimum amount of complexity to ensure that resolving uncertainty involves deliberate reasoning by the expert.

In Blackbox, the complexity of the problem is determined by (1) the size of the grid, (2) the number of balls, and (3) the location of these balls relative to each other on the grid. A game which consists of one hidden ball is not very complex. To find the ball, the player simply fires beams until a deflection occurs, at which point the ball is located. The number of shots fired will depend upon the location of the ball and the method the player uses to choose shots. The same method will produce widely differing results depending upon where the ball is located. A comparison of these results will reveal little about the expert's reasoning. The games we are interested in observing must involve uncertainty that is resolved by deliberate reasoning rather than random chance. The uncertainty we examine occurs in games that consist of five, six, or seven balls hidden in random locations on a 10 x 10 grid.

There are three sources of uncertainty in the problem: (1) the expert does not know the contents of the grid apart from the total number of hidden balls, (2) the expert has no sure way of predicting the behaviour of a beam passing through an unknown area on the grid until the beam is fired, and (3) the trajectory of a fired beam has several alternative routes that it could follow to justify its observed behaviour. The problem's complexity will determine the amount of uncertainty the system must resolve. Although the number of hidden balls and grid size are variable, the expert system is designed to operate optimally when solving six ball games on a 10 x 10 grid. This configuration was chosen because almost all problem instances involve the desired level o uncertainty from all three sources. This would not be guaranteed with a smaller number of balls or grid size, while larger games would involve more uncertainty of the same type.

Only one solution is considered correct in any game. This is the solution that corresponds to the actual game board. However, it is <u>not always possible</u> for the expert to find this solution. Certain ball configurations will prevent beams from passing through certain areas of the grid, while some games can be explained by alternative boards. This is illustrated in figure 4.1.1. The final solution in this type of game can only be found by guessing. Our expert guesses at the solution when he encounters these situations. The expert's ability to guess permits the expert to play randomly generated games without requiring the user to analyse each game beforehand to ensure that it can be solved.


(a) Inaccessible area



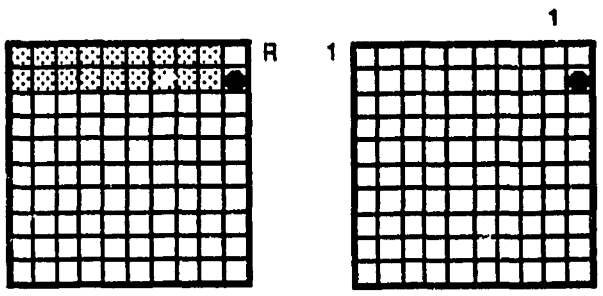
(b) Alternative game boards with identical shots

In figure (a), no shot can provide any information about the shaded area. If some ball is hidden within this area, its location must be guessed. In figure (b), all fired shots exhibit the same behaviour even though the two boards are different. In this case finding a solution requires guessing which board is the correct one.

Figure 4.1.1: Games that require guessing

The expert reduces his uncertainty about the grid by choosing and firing shots. The beam's behaviour allows the expert to define the contents of the grid along the beam's trajectory. If there is only one trajectory that the beam could follow, no uncertainty is involved. However, if several alternative trajectories are possible, the expert must establish which trajectory is the correct one. He may fire an additional beam to obtain more information about the grid. The expert's objective of minimising his score requires that he choose shots which will provide the most information. Determining which shot is optimal requires that the expert perform a heuristic assessment of the current state of the game. This involves evaluating available shots in relation to what is already known about the board and the beams which have been fired.

The expert determines the structure of the board by combining the information he obtains from fired beams. A different series of shots will produce alternative information describing the same board. One solution is considered better than the other if (1) its score, (2) the amount of time it takes to derive the solution, or (3) the number of errors it contains is lower. Changing the order in which a series of beams are fired will not affect the total amount of information obtained from the beams. However, it will affect the amount of information that an individual beam provides. In certain cases, firing one beam before another will make the second less profitable. This is illustrated in figure 4.1.2. In such cases, the expert will alter the series of shots he fires.



(a) Fire "R" first (b) Fire "1" first

Figure (a) and (b) illustrate two shots on the same board. In figure (a), the firing of "R" only provides information about the two squares immediately ahead of the firing point. This shot does not provide any information about the shaded area in the figure. In figure (b), shot "1" is fired before shot "R". Shot "1" provides information that defines the contents of the two top rows. In this case, it is no longer necessary to fire shot "R", as its behaviour is already defined.

Figure 4.1.2: Firing order affecting a beam's contribution

A large part of the knowledge required to solve Blackbox is procedural in nature. It is based on the rules which define how a beam's trajectory is affected by the contents of the area through which the beam passes. Procedural knowledge alone is insufficient for finding acceptable solutions. The player must also decide when exhaustive searches are feasible, when more information is necessary, and which shot to fire. When humans play the game, this choice is based on a visual assessment of the game board, and is guided by intuition. In the case of the expert system, the visual assessment is implemented by a systematic examination of the board which is neither as rapid nor as selective. The expert system uses a simple heuristic that does not express the full flavour of a human player's intuition.

4.2 Overview of the Blackbox System

The Blackbox System consists of a *session component* which controls a *game component* that contains the actual problem solving expertise. The session component presents a menu to the user through which the user enters specifications about the game session and the performance measurements to collect. The game component constitutes the expert system that solves Blackbox. An overview of the system's structure is given in figure 4.2.1. A detailed description follows. One can observe the expert's operation by watching the game evolve on a screen display.

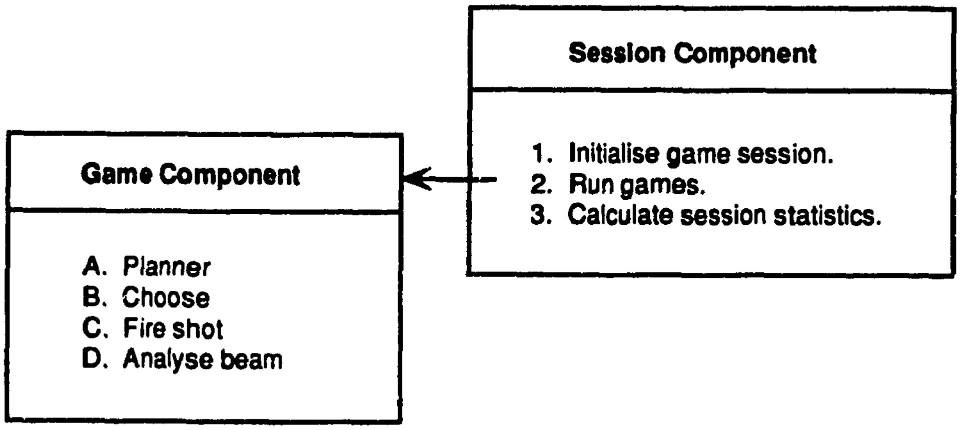
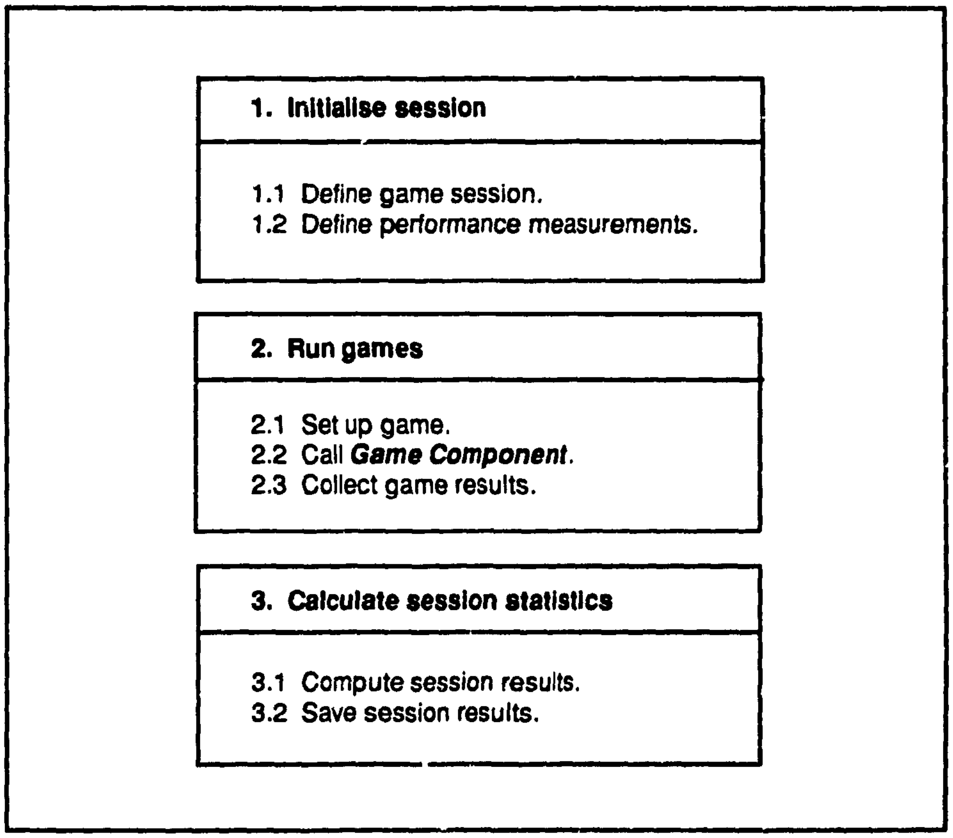


Figure 4.2.1: Overview of the system's structure.

Session component: The session component provides the interface through which the user defines how the game session will progress, and what performance measurements will be collected. Once the specifications are entered, the session component controls execution of the game session. Its functions consist of (1) setting up each individual game, (2) calling the game component to solve the game, and (3) saving all the game's measurements. At the end of the session, the session component calculates the session averages, and terminates. The session component is summarised in figure 4.2.2.



1. Initialise session
1.1 Define game session.
1.2 Define performance measurements.
2. Run games
2.1 Set up game.
2.2 Call *Game Component*.
2.3 Collect game results.
3. Calculate session statistics
3.1 Compute session results.
3.2 Save session results.

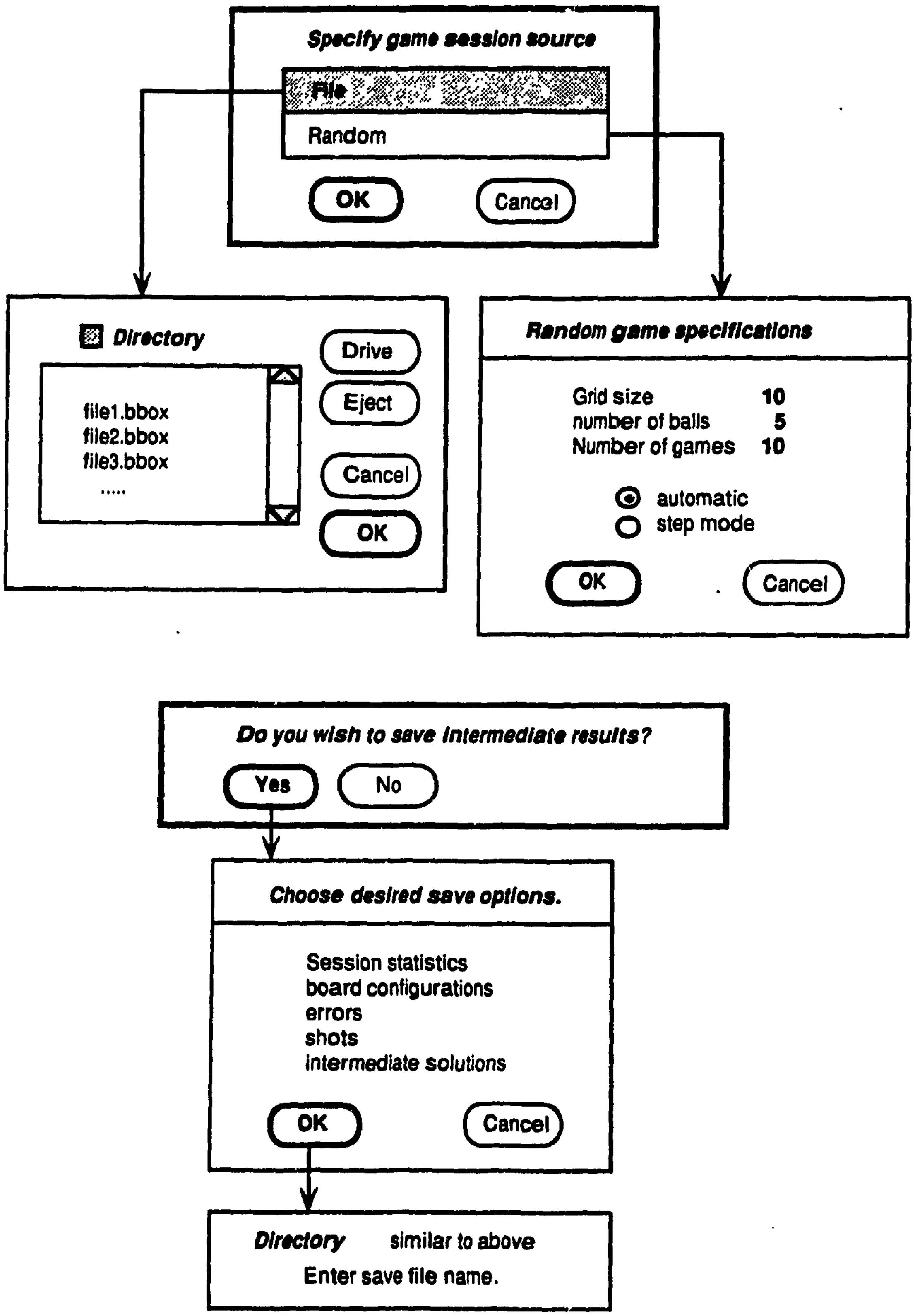
Figure 4.2.2: Session component

The user has the following options when entering specifications for a session. The user can choose to run sessions in either *automatic mode* or *step mode*. When the session is run in automatic mode, the system will play a specified number of games and then terminate with no intervention on the part of the user. When run in step mode, the system will pause before each game. The user can then (1) view the game, (2) proceed to the next game, (3) let the game be played, or (4) quit the session. The user also specifies whether game boards are (a) read from a file, or (b) randomly generated. If read from a file, the system provides the user a listing of available files. If game boards are randomly generated, the user specifies the number of hidden balls, and the size of the grid. The user specification menus are given in figure 4.2.3.

The user identifies which performance measurements to collect, and a file name in which the data will be saved through a menu interface. By default, the save option collects the following for each game:

(1) the game board
(2) the number of shots
(3) the score
(4) the time in minutes
(5) the number of errors in the solution
(6) the order in which shots were fired.

At the end of the session, the averages for 2, 3, 4, and 5 are calculated. As it stands, the system does not collect any other intermediate data. This does not seem necessary as the reasoning process that lead to a particular solution can be reconstructed from the information saved. Reconstructing the solution seems preferable to saving large amounts of more detailed data that would rarely be used. However, if anyone does wish to save more data, the necessary hooks are available in the system.



Grid size	10
number of balls	5
Number of games	10

[✓ automatic] step mode

Figure 4.2.3: Session specification menus

Game component: The *game component* contains the actual problem solving expertise. The expertise is functionally decomposed into modules. An overview of the game component is provided in figure 4.2.4.

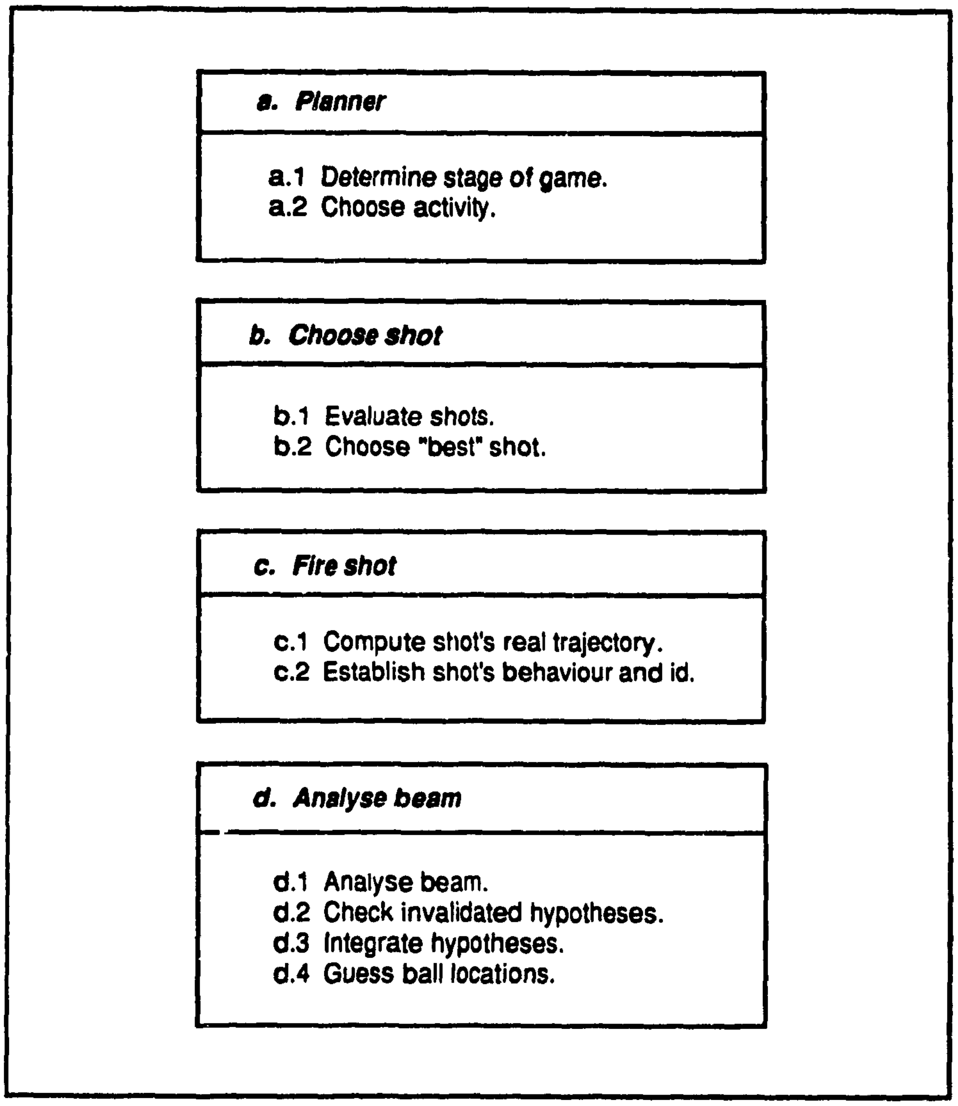
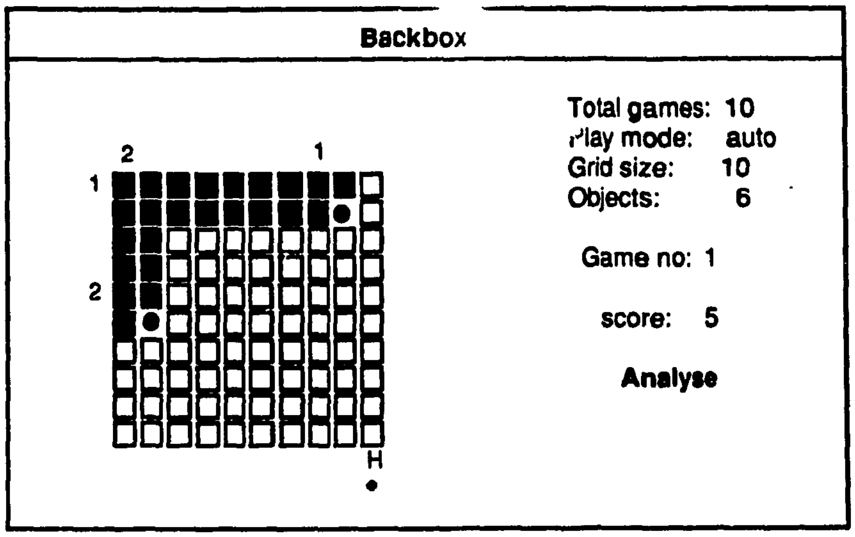

a. Planner
a.1 Determine stage of game.
a.2 Choose activity.
b. Choose shot
b.1 Evaluate shots.
b.2 Choose "best" shot.
c. Fire shot
c.1 Compute shot's real trajectory.
c.2 Establish shot's behaviour and id.
d. Analyse beam
d.1 Analyse beam.
d.2 Check invalidated hypotheses.
d.3 Integrate hypotheses.
d.4 Guess ball locations.

Figure 4.2.4: Overview of the game component

The highest level module (a) controls the progress of the game. It evaluates the current state of the problem, and calls one of the other modules accordingly. The lower level modules can be called in any order. The lower level modules (b) evaluate and choose the "best" shot to fire, (c) compute the trajectory of the fired shot and establish the behaviour of the resulting beam, and (d) analyse the unsolved beams to determine the hidden contents of the board. The invoked module performs its reasoning, and then returns control to the control module. Each lower level module is further decomposed to handle the different strategies required at different stages in the game. These strategies are discussed in detail in the following sections.

The user observes the game's progress on the screen. The screen displays (1) which game in the series is being played, (2) the number of hidden balls, (3) the current score, (4) the state of the grid, (5) what shots have been fired, their behaviour, and their identification, (6) the expert system's current activity, and (7) the end of the beam on which it is focused if performing analysis. The display is modified each time a shot is fired, a change is made to the board's contents, or the system changes its activity. The screen display is illustrated in figure 4.2.5.

The state of the board and the identity of the beam being analysed allow the user to reconstruct the reasoning process the system is performing. The user can determine whether the action is successful by the changes that appear on the screen. The user's ability to reason about the problem at the same time that the expert system is running allows the user to compare his decisions to those of the expert and question any differences. This visual interaction with the system makes the system's reasoning easy to trace. This facility for on-line comparisons is preferable to analysing large quantities of data after execution.



Total games: 10
Play mode: auto
Grid size: 10
Objects: 6

Game no: 1

score: 5

The screen display during progress of a given game provides; **session information:** the total number of games in the session, the play mode, the grid size, and the number of hidden objects; **game information:** the game number in the session, and the current score; and the **grid.**

Black squares ■ indicate empty squares, circles ● indicate balls, and blank squares □ indicate that the contents is unknown. Each end of an exiting shot is marked with an integer, reflecting shots are marked "R", and absorbed shots are marked "H". The dot • indicates which end of a beam is currently being analysed.

Figure 4.2.5: Screen display

4.3 The Blackbox Solution Process

Reasoning in Blackbox is performed by the game component (figure 4.2.4). The system attempts to obtain the maximum amount of information from available sources before firing another shot. This is reflected in the priorities that the control module applies when determining what action to perform next. The modules for choosing shots and analysing beams perform either a cursory or exhaustive search depending upon the state of the solution. The module that fires shots simply calculates a shot's behaviour on a given board according to the game rules. The firing module is not considered further as it does not involve any reasoning.

The solution process consists of deciding where to shoot, firing the shot, and then analysing the point at which the beam enters the board. Analysis establishes a hypothesis for a beam by applying the rules that relate how a beam's trajectory is affected by the contents of the board. The board's contents are derived by determining where the beam could pass, and where it cannot pass in order to produce the observed behaviour. Changes made to the contents of the board during analysis are checked to see if they affect any hypotheses about beam trajectories. Hypotheses that are invalidated by the changes are reanalysed. Reanalysis of hypotheses may produce further changes, invalidating more hypotheses. These are reanalysed until no more information can be gained. The solution process is repeated until a solution is found.

The control module examines the current problem solving state and determines which reasoning module to activate. It has the choice of continuing analysis of currently available information, or of firing another shot. If it chooses to continue analysis, the system must determine which beam hypothesis or hypotheses to examine. If firing is chosen, the system must determine where to fire the shot. Choosing an "optimal" shot *will* or *will not* involve evaluating the potential contribution of "available shots", depending upon how many changes have occurred since the last evaluation. In certain cases, available shots cannot provide any additional information, while in others the available information appears sufficient for finding the solution. In either case the system attempts an exhaustive analysis during which it integrates what it knows about all unsolved beams. The system "guesses" at ball locations when no other strategy is applicable. Figure 4.3.1 describes the available control choices.

Control choices	
Fire	*Analyse*
1. Evaluate available shots.	1. Analyse beam hypothesis.
2. Choose shot.	2. Check invalidated hypotheses.
3. Shoot.	3. Integrate hypotheses.
	4. Guess ball locations.

Figure 4.3.1: Control choices in the solution process

The problem solving states in the game are distinguished by a number of conditions, labeled C1 to C7. These conditions indicate which reasoning module can be applied, and what stage the game has reached. The conditions and their relation to control choices are given in figure 4.3.2.

Conditions

C1.	All balls have been located.
C2.	The number of balls equals the number of unknown squares.
C3.	Only one ball remains to be found.
C4.	Board changes have occured.
C5.	Invalidated hypotheses are present.
C6.	No more information can be derived from current sources.
C7.	Remaining shots can provide no additional information.

	Conditions						
Control Choice	*C1*	*C2*	*C3*	*C4*	*C5*	*C6*	*C7*
TERMINATE	X	X					
FIRE	...	...	...	...	...	X	
1. Evaluate	...	...	X	X			
2. Choose							
3. Shoot							
ANALYSE							
1. Analyse	...	...	...	...	X		
2. Invalidate	...	...	...	X			
3. Integrate	...	...	X	...	...	...	X
4. Guess	...	...	...	...	...	..	X

Figure 4.3.2: Conditions and affected control choices

The occurrence of C1 signals that the game is over. C2 indicates that the game is over once the remaining unknown squares are filled with balls. C3 is the condition indicating that the game is in its final stage, and an exhaustive search can be attempted. The occurrence of C4 and C5 indicates that additional information might be obtained by analysing hypotheses about unsolved beams. In the case of C4, the hypotheses should be checked to determine if any have been invalidated by the changes, while C5 indicates that invalidated hypotheses remain to be analysed. C6 results from the absence of conditions C4 and C5, and indicates that another shot should be fired in the absence of C1 and C2. C7 indicates that all remaining shots are useless. When combined with C6 and the absence of C1 and C2, it indicates that finding the solution involves either an exhaustive search or guessing.

Shot evaluation: The principal difference between the two strategies available for evaluating the potential contribution of shots is the depth of their search. Both rely on a systematic examination of each available shot. The value assigned to the shot is based on the shot's predicted behaviour and the number of unknown squares on its hypothetical trajectory. The first strategy is cursory in that it only considers a single trajectory for each shot. The second strategy is exhaustive. It examines each alternative trajectory that the beam could follow, and bases its value on a count of the unknown squares and of the different types of behaviour that the beam could exhibit. This eliminates many useless shots whose behaviour would not provide any information at the end of the game. The utility of applying it any earlier is overwhelmingly offset by its time complexity.

The process associated with evaluating shots consists of (1) establishing the point at which the shot actually enters a region on the grid whose contents is unknown, **(2)** hypothesising a trajectory from that point, **(3)** counting the unknown squares along that trajectory, and **(4)** determining the behaviour which would result if the shot followed that trajectory. The point at which a given shot enters an unknown region is maintained for each shot and used to eliminate shots whose behaviour becomes totally defined without firing. The hypothesised trajectory assumes no balls are located in any of the unknown squares. The count distinguishes between squares which are situated on the trajectory, and those which are adjacent to it. If the hypothetical beam exits, its exit location is noted.

Beam analysis: A similar choice between either a cursory or exhaustive search occurs in analysis. The strategy may be applied to a single beam, or in resolving several beams together. The inappropriateness of applying an exhaustive search at the beginning of the game is obvious. When little or nothing is known about the grid, an exhaustive search on a single beam may take over four hours[1]. Thus, an attempt is made to avoid exhaustive searches in the initial stages of the game. As more knowledge is gained about the board, the number of alternative trajectories that any single beam could follow diminishes. A more exhaustive search that attempts to integrate alternative trajectories from several unsolved beams becomes feasible.

The basic reasoning process applied when analysing single beams consists of the following five steps:

1. When a shot is fired, an initial hypothesis is established for it.

---

[1]This was noted during one debugging session. It is neither a maximum nor a limit.

2. This hypothesis and the board are then analysed in order to make changes to the board's content and the hypothesis.

3. All other beam hypotheses are then examined in order to establish which hypotheses have been invalidated by the changes.

4. The beams with invalidated hypotheses are assigned priorities for analysis.

5. The beam with the highest priority is reanalysed. If its hypothesis can't be revalidated, (2) is repeated. If the hypothesis is revalidated, then (5) is repeated for the remaining beams.

Whenever changes are made to the board, beam hypotheses are checked (3). Invalidated hypotheses are reanalysed (4 and 5) until no more information can be gained.

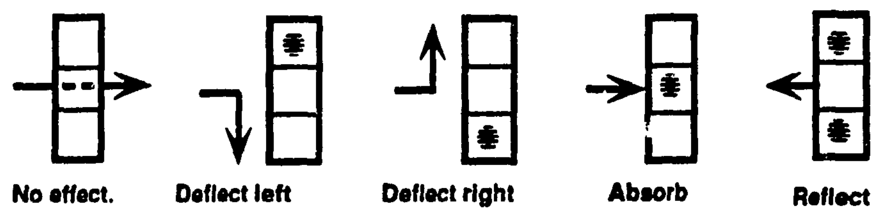
The analysis by which several beams are integrated employs the same basic reasoning as that applied to single beams. It considers several alternatives for each beam and compares them to those available for other beams in order to eliminate all alternatives which cannot coexist. In certain cases the elimination of alternatives establishes the contents of the grid. When this occurs, the basic reasoning process described in the previous paragraph is applied. The stage at which this exhaustive strategy is appropriate varies from game to game. It is employed when possible shots cannot provide any additional information, and it is deliberately attempted when only one hidden ball remains to be located.

4.4 Knowledge Representation in Blackbox

The way in which knowledge is represented in an expert system is related to the information requirements of the reasoning strategies. In our case, the domain specific knowledge is represented in a procedural manner to keep the representation concise. The solution is represented by a set of *conceptual objects* that also provide a description of the current problem solving state. The reasoning process manipulates these objects to transform them from an initial problem state to a final solution state. The "significant" states are those which affect control decisions (see figure 4.3.2). Control decisions relate to analysing beams or firing shots, and all associated subactivities.

The domain specific knowledge required to solve Blackbox is contained within the game rules, which describe how a beam moves over the board. These rules describe how the trajectory of a beam is affected by the contents of a 3 x 1 area adjacent to the location through which the beam passes. This area is named the "*vicinity*" of a location along a beam's trajectory. As stated earlier (section 3.4) the contents of the beam's vicinity can "*absorb*", "*reflect*", or "*deflect*" the beam in either of two directions. The effects of the vicinity's contents on the beam's trajectory are illustrated in figure 4.4.1. These are the basic rules by which the behaviour of any beam can be explained.

The beam's orientation on the board in relation to the vicinity determines how the beam's trajectory is affected by the vicinity's contents. A beam is considered travelling inwards from its entry point towards the centre of the grid. The relevant vicinity is the area perpendicular to the beam's direction and immediately ahead of the beam's currently known location. If the contents of the vicinity are known, then the beam's next location is known. If the vicinity's contents are unknown, then three alternative locations are possible, corresponding to *deflect left*, *deflect right*, and *no influence*, as illustrated in figure 4.4.1. The beam's current direction is identified by the unique direction which identifies the beam's preceding location rather than its next location. The labels "up", "down", "left", and "right" on the board identify the current direction in relation to the preceding location. This is illustrated in figure 4.4.2.



The above are the basic ball configurations that affect a beam's trajectory. An empty vicinity has **no effect** on a beam's trajectory. **Deflect left** or **right** indicates the direction in which the beam is deflected. The beam is **absorbed** when it encounters a ball head-on. When a beam is **reflected,** it reemerges at the same point that it entered the grid. A reflection at the edge is a deflection that prevents the shot from entering the grid.

Figure 4.4.1: Ball configurations that affect beam trajectories.

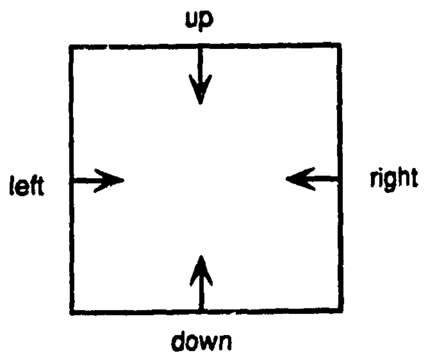


Figure 4.4.2: Direction of a beam on the board

The beam's next location is determined by combining its current location and direction with the deflection produced by the contents of its vicinity. A beam may be deflected several times before it reaches the target configuration that produces the behaviour exhibited by the beam. Each of these deflections results in a change of direction for the trajectory of that beam. A beam's trajectory is uniformly represented by decomposing the trajectory into a sequence of *partial trajectories* corresponding to each deflection that occurs in the trajectory. Each partial trajectory is described by:

(1) An entry location and the direction of the beam.

(2) An area through which the beam moves in one direction.

(3) The location of a ball which changes the beam's direction.

The next direction and location resulting from the beam's deflection correspond to the entry location of the next partial trajectory. If the beam is absorbed or reflected, the last partial trajectory will contain the required ball configuration. If the beam exits from the grid, the last location in the area will correspond to the beam's exit location. Thus the trajectory of any beam can be represented by a sequence of partial trajectories. The number of partial trajectories will vary according to the number of deflections that occur in the trajectory. This is illustrated in figure 4.4.3.

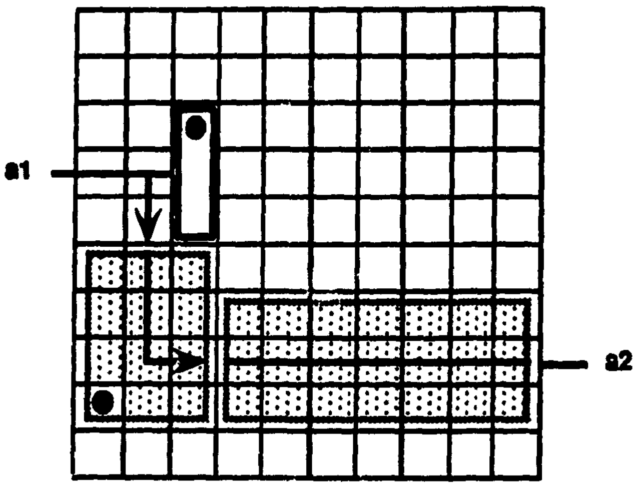


Figure (a): Representation of a beam

The representation of a beam entering at **a1** and exiting at **a2**. The beam's immediate **vicinity** is outlined in black. The beam's **entry point** is the square preceding it. Each beam is represented by a sequence of partial trajectories that describe the beam's route over the board. The illustrated beam consists of two partial trajectories resulting from two deflections.

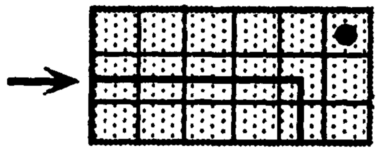


Figure (b): Representation of a partial trajectory

A partial trajectory is represented as: an **entry point** (➔), an **empty region** (shaded area ☐), and a **ball location** (●). The last partial trajectory may contain *no ball location, a "hit" ball*, or two balls corresponding to *a reflection*, depending upon the beam's behaviour.

Figure 4.4.3: Representation of a beam's trajectory.

The state of the solution process, or the problem state, is represented by a quintuplet of objects as follows:

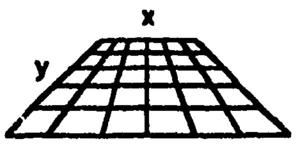
S: < G, N, P, E, U >

where:

1. **G:** the grid.
2. **N:** the number of hidden balls that remain to be located.
3. **P:** possible shots.
4. **E:** evaluated shots.
5. **U:** unsolved beams.

The five objects are interrelated by the aspects of the solution they describe and manipulate. The state of each object is relevant in determining the current problem state and in deciding which action to perform next. A detailed description of each object follows.

The grid <G> is represented as an ordered sequence of (X,Y) locations and their contents. The possible contents of a given location are *unknown*, *empty*, or *ball*. At the beginning of the game all locations are initialised to unknown. Whenever the grid's contents are modified, the location and its change are marked. These markings are used in invalidating trajectory hypotheses. The markings are maintained between each invalidation check. The type and number of changes determine whether or not the potential value of shots should be reevaluated. The type and number of changes are maintained from one evaluation until the next. When the number of unknown squares is equal to the number of hidden balls, or N = 0 (all balls have been located), the game is over. The information contained within the grid is illustrated in figure 4.4.4.



grid:	(X,Y) location and content.
changes:	(X,Y) location and change.
type:	Number of changes to empty, and changes to ball.

The contents of the grid are *unknown* ▨, *empty* □, and *ball* ●.

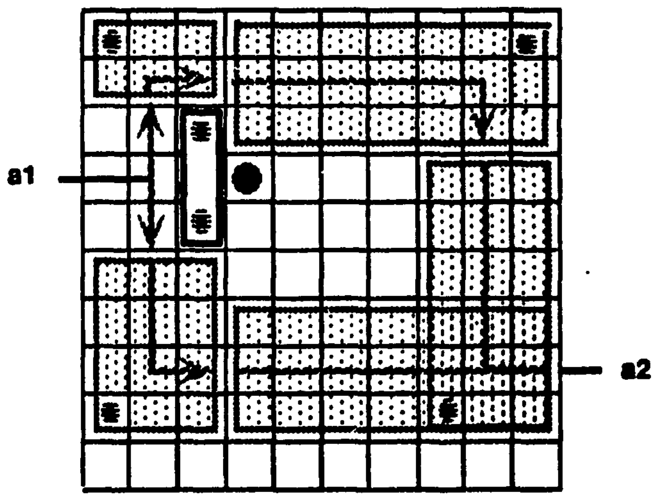
The number of unknown squares can be determined for any specified row, column, or area on the grid.

Figure 4.4.4: Representation of the grid

The *number of hidden balls* <N> is initialised at the beginning of each game. Each time a ball is located on the grid, its insertion is made through the "ball object", which decrements 'N' accordingly. The object that describes hidden balls is slightly more complex than a simple counter. It ensures that more than one ball is not inserted into the same location, and distinguishes between *actual* and *available* balls.

Actual balls provides the total number of balls that remain to be located in the game. *Available balls* describes the number of balls that can be used in the construction of a particular trajectory hypothesis. The number of actual balls and available balls will differ according to what is known about the trajectories and the board. The explanation follows.

At certain points during the solution process it is possible to identify "*ball vicinities*" on the grid. A *ball vicinity* is a 3 x 1 vicinity on the grid in which at least one ball exists, but the exact location of the ball is not known. A ball vicinity is established during analysis of some unsolved beam. This implies that everywhere outside that vicinity there is one less ball available for constructing trajectory hypotheses. Whenever the analysis component hypothesises about a ball location, it accesses the *ball object* to determine if a ball is available. Whenever the hypothesised location is not within a vicinity, there is one less ball than that which would be available if the location were in the vicinity. Figure 4.4.5 illustrates a situation in which a ball vicinity is defined.



The beam entering at a1 must be deflected either *left* or *right* by the contents of its vicinity (outlined in black). If the beam is not deflected, and continues straight on, then the beam would be absorbed by the ball known to be located immediately ahead. As being absorbed contradicts with the beam's known behaviour, a ***ball vicinity*** is established. This ball vicinity must contain one ball in either of the locations indicated with ■ in the vicinity.

Figure 4.4.5: Situation in which a ball vicinity is defined

Ball vicinities may overlap. In this case, one ball could account for the overlapping vicinities. The ball object establishes the number of balls available when all vicinities are taken into consideration. A description of the ball object is provided in figure 4.4.6.

Actual balls: Total number of balls that remain to be located.
Available balls: Actual balls *less* all balls within inaccessible ball vicinities.
Ball vicinities: A ***vicinity*** on the grid which must contain at least one ball.

A ***ball vicinity*** is represented by $[(X_1,Y_1), (X_2,Y_2), (X_3,Y_3)]$, with at least two, and possibly three of the locations marked as potentially containing balls.

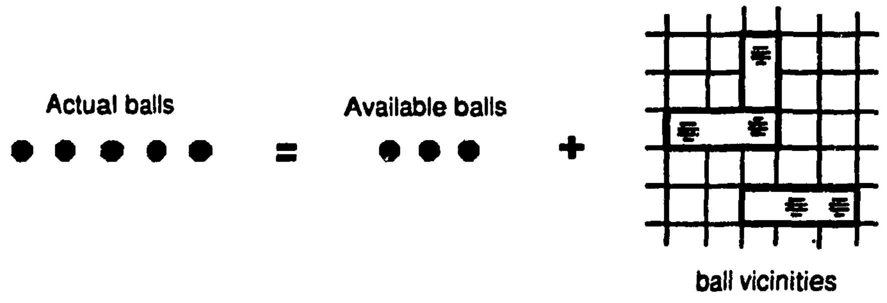
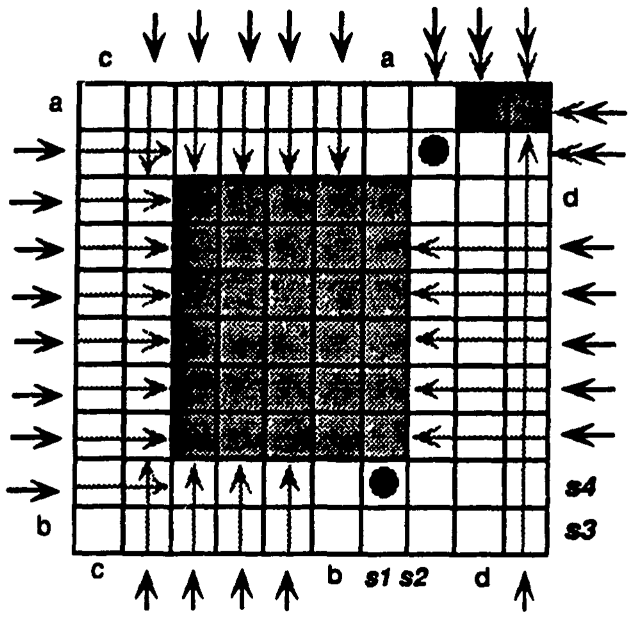


Figure 4.4.6: Ball object

Possible shots <P>, *evaluated shots* <E>, and *unsolved beams* <U> are different representations of the same object as it is transformed during the game. A possible shot represents a potential trajectory that could provide information about the grid. An evaluated shot places a value on the information which that potential trajectory could contribute. An unsolved beam is the result of firing a possible shot. The shot fired is that with the highest value among the evaluated shots. Once fired, the shot is eliminated from possible and evaluated shots. Once resolved, the beam is eliminated from unsolved beams as no additional information can be gained from it.

Possible shots <P> are initialised at the beginning of each game by creating a record for each firing location on the grid. Each shot is identified by its entry location outside the grid, and its current direction and location, which describe the point at which that shot encounters an unknown region on the grid. At first, the shot's current location is identical to that of its entry location. As the game progresses and the grid's contents become defined, its current point will change according to what becomes known. In certain cases, the shot's entire trajectory will become defined. Shots that become defined are eliminated from the set of possible shots. Possible shots are illustrated in figure 4.4.7. The structure of possible shots is given in figure 4.4.8.

Evaluated shots <E> are created by accessing each possible shot, and assigning a value to its potential contribution. The evaluation object maintains a set of evaluated shots, which are recreated whenever evaluation is considered necessary. This object is accessed to obtain the shot with the maximum value. Evaluation is based on a count of unknown squares and the shot's predicted behaviour. If the shot's hypothetical trajectory leads that shot out of the grid, its exit location is noted, and the possible shot that corresponds to that exit location is not reevaluated. Thus, although each evaluated shot corresponds to a possible shot, there is not a one to one correspondence between them. All evaluated shots that correspond to a fired shot are eliminated when the shot is fired. The structure of evaluated shots is illustrated in figure 4.4.9.



The **entry locations** of *possible shots* are marked with ⇾ on the board.
The **current location** and **direction** of *possible shots* is marked by ⇢.

The figure illustrates *possible shots* after beams a, b, c, and d have been analysed. As can be seen, the *current location* of a *possible shot* advances as the contents of the grid become defined. Possible shots ***s1***, ***s2***, and ***s3*** can provide no information, and are therefore eliminated. The remaining shots will provide varying amounts of information, depending upon their *current location* and *direction*, and what is known about the contents of the grid beyond that point.

Figure 4.4.7: Possible shots on the board

Possible shots: [(Xe,Ye) (Xc,Yc) Direction]

(Xe,Ye)	Coordinates of *entry location* at the edge of the grid.
(Xc,Yc)	Coordinates of *current location.*
Direction	*Current direction* specified as [*up, down, left, right*].

The object initialises all *possible shots* at the beginning of the game, and maintains them throughout the course of the game. Whenever the shots are evaluated, their *current location* and *direction* are updated. Shots that are fired, or which become defined are eliminated from the set. An empty set implies that no more shots are available.

Figure 4.4.8: Representation of possible shots

Evaluated shots: [(Xe,Ye) (Xo,Yo) Hits Deflects Value Behaviour]

(Xe,Ye)	*Entry location* identifies the shot.
(Xo,Yo)	Identifies hypothetical exit location of the shot.
Hits	Number of *unknown squares* **on** the shot's hypothetical trajectory.
Deflects	Number of *unknown squares* ***adjacent to*** the hypothetical trajectory.
Value	Value assigned by the shot evaluation function.
Behaviour	Behaviour that the shot would exhibit on its hypothetical trajectory.

The object maintains a set of *evaluated shots* derived from the set of *possible shots*. Whenever shots are fired, the corresponding *evaluated shots* are eliminated from the set. *Evaluated shots* are accessed to determine the shot with the greatest potential value.

Figure 4.4.9: Representation of evaluated shots

A description of an *unsolved beam* <U> is created when a shot is fired. Figure 4.4.10 describes an unsolved beam. It consists of a unique identifier, the beam's behaviour, its current location and direction, and a description of the hypotheses associated with that beam. An *unsolved beam* description is associated with every beam, and with each end of a beam if that beam exits from the grid.

The hypothesis description contained within the representation of an unsolved beam indicates what alternative trajectory hypotheses are possible from that location. The maximum is three trajectory hypotheses, corresponding to the three deflections that could occur in that location's vicinity. (see figure 4.4.1). The presence of a trajectory hypothesis indicates the existence of a sequence of partial trajectories with which a trajectory hypothesis is represented. Figure 4.4.11 shows the description of a partial trajectory hypothesis.

Unsolved beam: [Id End (Xc,Yc) Direction Behaviour Hypothesis]

Id	Unique identifier assigned the beam when it is fired.
End	Identifies each end of an *exiting* beam.
(Xc,Yc)	Coordinates of the beam's *current location.*
Direction	The beam's *current direction.*
Behaviour	The beam's observed behaviour [*exit, hit, reflect].*
Hypothesis	Indicates the presence of *trajectory hypotheses* associated with the beam. The hypothesis representation is described below.

Hypothesis: [0/1 0/1 0/1]

The position and value within the *hypothesis* description indicate if a trajectory hypothesis is associated with the location in question. The positions correspond to *left, centre,* and *right* in the vicinity of that beam's *current location.* Each identifies a trajectory hypothesis represented by a sequence of *partial trajectory hypotheses,* whose representation is given in figure 4.4.11.

Figure 4.4.10: Representation of an unsolved beam

Partial trajectory hypothesis

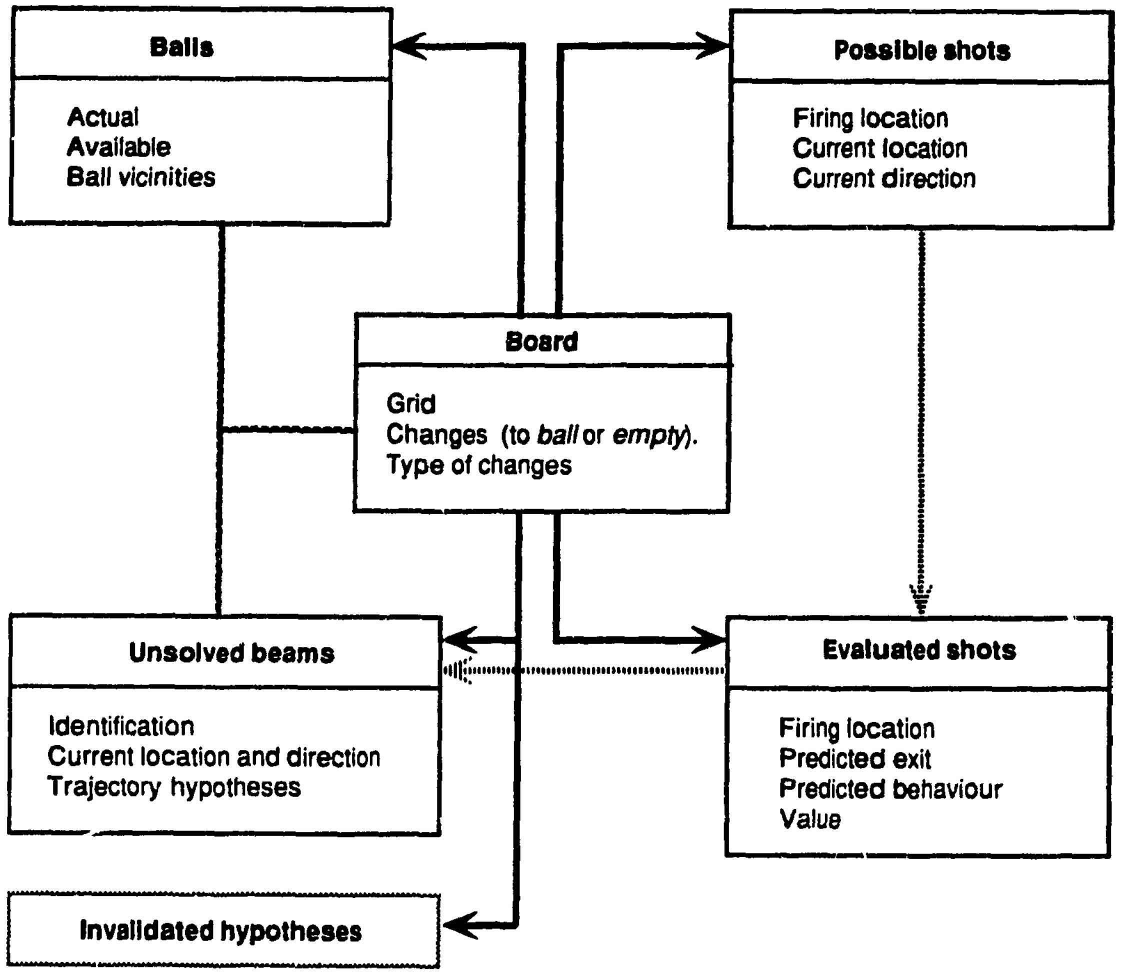
[Id End Branch (Xc,Yc) Direction (Xb,Yb) (Xu,Yu,Xd,Yd)]

Id	Beam identifier.
End	Identifies which end of the beam the hypothesis concerns.
Branch	Identifies the hypothesis *(left, centre,* or *right).*
(Xc,Yc)	Coordinates of the hypothesis' *entry point.*
Direction	The hypothesis' *current direction.*
(Xb,Yb)	Coordinates of a hypothetical ball causing a deflection.
(Xu,Yu,Xd,Yd)	Coordinates of the rectangular grid area through which the beam passes.

This representation corresponds to a *partial trajectory hypothesis,* as illustrated in figure 4.4.3. The contents of **(Xb,Yb)** in the last *partial trajectory hypothesis* in the sequence will differ when the beam is *reflected,* or *exits* without a deflection.

Figure 4.4.11: Representation of a partial trajectory hypothesis

There are obvious interrelationship between the five objects described. The number of hidden balls must correspond to the number on the grid, while the number of balls available depends upon which location is being considered. Possible shots correspond to firing locations on the board, while their current location is determined by what is known about the grid. Evaluated shots describe the number of unknown squares and the predicted behaviour of a possible shot if its trajectory is extended into the unknown region of the grid. Unsolved beams describe hypothetical trajectories for beams whose exact trajectory is unknown. The grid's content and the number of balls available will determine whether the trajectories are valid. Changes to the grid may invalidate trajectory hypotheses, and the value of evaluated shots. An overview of these interrelationships is provided in figure 4.4.12 on the following page.



The above illustrates the five objects used to represent the state of the Blackbox solution. *Possible shots* are *evaluated*, and then fired to produce *unsolved beams*. This relation is represented by ⇣.

The trajectories of *unsolved beams* are constrained by the state of the *board* and the number of *balls*. These constraints are used to eliminate impossible hypotheses, and change the contents of the *board*. This relationship is represented by —.

The state of the *board* determines what *balls* and what *possible shots* are present, the value of *evaluated shots*, the trajectory hypotheses that *unsolved beams* could follow, and the trajectory hypotheses *invalidated* by any changes to the state of the *board*. (Note that *invalidated hypothese* are simply invalid trajectory hypotheses for *unsolved beams*.) This relationship is represented by →.

Figure 4.4.12: Overview of the Interrelationship among the objects

4.5 Reasoning Strategies Within Blackbox

The reasoning strategies employed in Blackbox are based on certain assumptions about the game: (1) The solution space is nontrivial. (2) The location of balls is random. (3) Information about the behaviour of a fired beam is always reliable. These assumptions have an impact on the nature of the reasoning strategies. The searches and strategies used for shot evaluation and beam analysis are structured in consequence:

a. Exhaustive searches are intractable. They are therefore avoided until either highly constrained, or inevitable.

b. Balls have an equal chance of being hidden in any unknown square on the board. Thus the same strategy can be applied when analysing any size region containing any number of balls.

c. Once a beam has been fired, there is at least one game board that will produce the observed behaviour. This ensures that eventually some solution will be found, although it may have to be guessed.

Shot evaluation: can be either cursory or exhaustive. A cursory evaluation considers only one hypothetical trajectory for a shot, while the exhaustive version considers every possible alternative. In most cases, a cursory evaluation is considered sufficient because (a) there is no accurate way of determining the value of any shot before it is fired, and (b) finding a solution generally requires a relatively even distribution of beams over the board. An exhaustive evaluation would not be more accurate than a cursory evaluation when little or nothing is known. This is taken into account in the heuristics used to evaluate possible shots, and the values assigned to them.

The cursory form of evaluation assumes the simplest trajectory for a possible shot and counts the number of unknown squares on it. The simplest trajectory assumes that all unknown squares are empty. This is a valid assumption since balls have an equal probability of being situated anywhere on the grid. Choosing shots according to the number of squares they may provide information about ensures that shots will be aimed at areas that contain the largest number of unknown squares.

At the beginning of the game when the grid is relatively unknown almost all evaluated shots will exit from the grid. The shot associated with the exit location is not reevaluated. This effectively reduces the total number of possible shots processed. With the entire grid unknown, only half the shots are evaluated. The number of shots evaluated varies as the specific game progresses. Shots are not reevaluated until a significant number of changes have been made to the grid. The loss in "accuracy" due to non-evaluation is considered insignificant as evaluation itself is inaccurate. This results in a substantial savings in time.

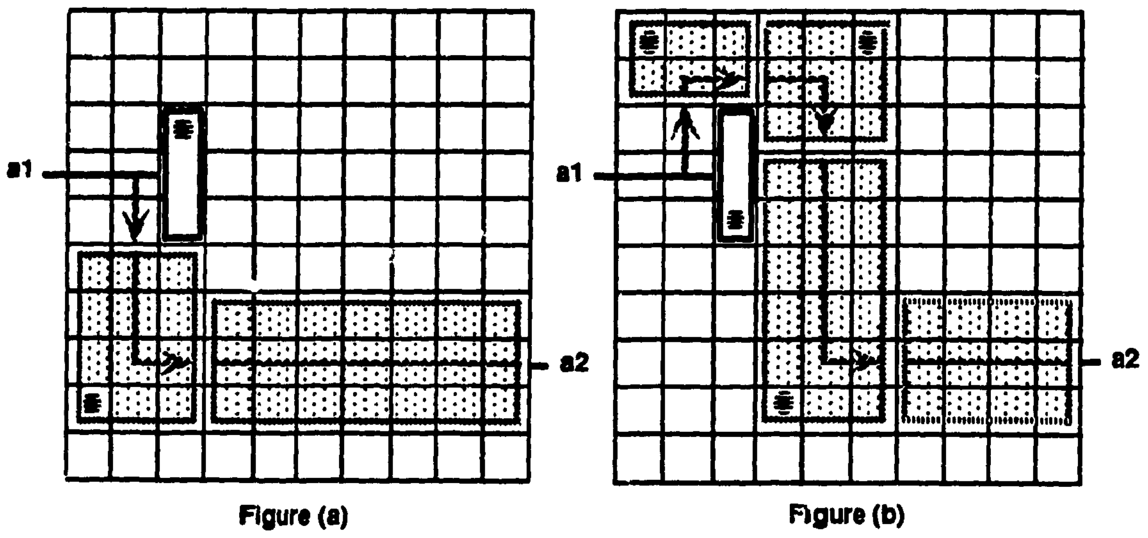
The value of a shot is based on the number of possible hits and deflections that could occur in the course of its trajectory. Shots which are more likely to be deflected are favoured. Shots with a limited number of possible deflections are assigned a higher value. Shots that are potentially absorbed receive the lowest value since they provide the least information. The heuristic used to assign values to shots promotes (1) expanding the known regions on the grid, (2) ensuring an equal distribution of beams through unknown regions, and (3) avoiding shots that provide little information about the grid.

A cursory evaluation is suitable in the initial and intermediate stages of a game. At the end of the game when only one ball remains to be located, the grid is relatively well defined and the number of available shots is limited. A more careful evaluation at this stage eliminates useless shots, and favours shots which provide the most information, thus minimising the game score.

Beam analysis: The reasoning applied when analysing a beam relates the beam's behaviour to the vicinity ahead of its current location. The contents of the vicinity are defined according to what is known about the beam, the grid, and the hidden balls. If the beam exited from the grid, then it cannot be reflected or absorbed. On the other hand, a reflected beam must encounter a configuration that produces a reflection, while an absorbed beam must hit a ball somewhere along its trajectory. This knowledge allows us to eliminate certain configurations from those possible in the unknown vicinity ahead of the beam. The elimination process establishes what the contents of the vicinity must be in order to produce the observed behaviour. An intermediate step in the elimination process consists of establishing a *ball vicinity*.

If the contents of the vicinity are unknown, three alternative trajectories are possible from that point: the trajectory may be deflected *left*, *right*, or pass *straight through the centre* of the vicinity (see figure 4.4.1). An alternative trajectory is possible if the observed beam behaviour can be produced by following that trajectory. The beam is analysed by examining each alternative trajectory to determine whether it is valid. The alternative's validity is established by constructing a hypothetical trajectory on the current board with the available balls.

A hypothetical trajectory consists of the sequence of deflections necessary to produce the observed beam. Each deflection corresponds to a ball that exists, or could potentially be located on the grid. This is illustrated in figure 4.5.1. The hypothesis is at all times constrained by the contents of the grid, and the number of balls available along its trajectory. If a trajectory cannot be found, that alternative is eliminated. The solution advances by eliminating alternatives until only one trajectory remains. When this occurs, the contents of the vicinity are defined, and the beam advances to the next location.



Two possible hypotheses for a shot entering at **a1** and exiting at **a2**. The immediate ***vicinity*** is outlined in black. The ***entry point*** is the square preceding it. Each hypothesis consists of a sequence of partial hypotheses that describes a route that could be followed to reach the shot's exit point.

Figure (a) represents a beam hypothesis composed of two partial hypotheses that would require two balls. Figure (b) represents a beam hypothesis composed of four partial hypotheses that would require four balls. A third beam hypothesis that would not be affected by the vicinity is not illustrated.

Figure 4.5.1: Alternative trajectory hypotheses for a beam

The analysis of a trajectory can be cursory or exhaustive. Each vicinity on a given trajectory gives rise to three possibilities associated to *deflect left*, *right*, and *centre*. The number of possible trajectories makes it impossible to consider all trajectories for a given beam unless the search is highly constrained. Instead, the strategy attempts to construct the simplest trajectory for each alternative. In the case of exiting shots, a "*hill-climbing*" strategy is used [2]; a hypothetical trajectory is found by trying to reduce the distance between the beam's known entry and exit locations. Trajectories for beams that are reflected or absorbed are found using a "*depth first*" search [2]; the search attempts to find the necessary configuration on the most direct trajectory.

Although an effort is made to avoid exhaustive searches, there is no guarantee that the search will not have to consider every possible trajectory for an alternative in order to validate or eliminate that alternative. In the case of exiting shots, the number of trajectories considered often depends upon which end of the beam is analysed first. There are six heuristic rules associated with choosing the end, and various strategies for constructing alternative trajectories if the simplest hypothesis fails. The heuristics used are described in detail in the program listing, which is available on request.

When changes are made to the board, the change may invalidate hypotheses about trajectories. Hypotheses are invalidated whenever changes to the grid place a ball or ball vicinity within an area that was hypothesised as empty, set a hypoth-esised ball location empty, or exhaust all the balls avail-able without proposing a complete trajectory. When this occurs, the hypothesis is reconstructed from the point at which it failed. If this does not succeed, the attempt is repeated with preceeding partial hypotheses for that trajectory. If the trajectory hypothesis cannot be reconstructed, the alternative in question is eliminated. The order in which invalidated hypotheses are examined may have an impact on how rapidly the solution advances.

There are seven levels of priority assigned to beams with invalidated hypotheses. The priority reflects the number of alternatives the beam has, and the number of alternatives that have been invalidated. When no valid hypotheses remain, there is a good chance that the vicinity they describe will become defined. The number of invalidated hypotheses gives an indication of the amount of work involved in their reconstruction. The beams most likely to define a vicinity with a minimum amount of work receive the highest priority.

During cursory analysis only a single beam is considered at any moment. Although the search employed to establish alternative trajectories may be exhaustive, it will be limited to establishing a maximum of three alternative trajectories; one for each possibility in the vicinity of the beam's current location. The exhaustive analysis performed at the end of the game is not constrained in this way. It integrates what is known about all unsolved beams, and establishes the set of possible ball locations for all alternative trajectories that explain each beam without invalidating the others. The search will find one or more solutions, ordered from the simplest trajectories to the most convoluted.

An exhaustive search is employed when only one ball remains to be found or guessing is required. The search may establish a single solution that will end the game with no further shots. If it establishes several solutions, these solutions will be employed to eliminate impossible alternatives, and thus locate empty squares on the grid. This will reduce the number of unknown squares and make the subsequent *exhaustive evaluation* both more accurate and rapid. In the case of guessing, the search may not be constrained by a single hidden ball. I have observed the expert system locating three hidden balls within an acceptable time frame when all other sources of information were useless. When guessing, the search stops at the first valid solution.

The strategy used to guide the exhaustive analysis orders the unsolved beams according to the number of alternative trajectories they have. Those with the least number of alternatives are considered first because their invalidation is more likely to result in changes to the grid. The unsolved beams are analysed one by one. A trajectory hypothesis that does not invalidate those already established, is constructed. If no trajectory exists, the current hypothesis is invalidated. Analysis backtracks, and establishes an alternative possibility that satisfies the preceeding unsolved beams. This is repeated for every beam until a valid trajectory is found for each. The time complexity of an exhaustive search can be considerable, even when the process is highly constrained by a low number of balls, a well-defined grid, and a limited number of unsolved shots.

The Blackbox Expert System performs two reasoning activities: **(1)** choosing shots; **(2)** analysing unsolved beams. Each of these activities can be performed using either a cursory or exhaustive search. In addition to this, when analysing beams, the system must determine if analysis is possible, or whether guessing is required. The expert system must choose which activity to perform, and which strategy to apply. The choice depends upon the problem solving state, which is defined by the quintuplet of objects, *S:<G,N,P,E,U>* described in section 4.4. Although certain activities are associated with certain "significant" states (see figure 4.3.2), the system must establish that these states have occurred. Furthermore, the relation between a particular state and the available activities does not necessarily identify a single, optimal choice of activity for the expert system.

4.6 Evaluation of Blackbox

The evaluation of the Blackbox Expert System is based on observations made during its development and measurements obtained once the system was considered complete. As stated earlier, the system's evaluation is based on the results obtained by playing over 150 games. These results and our observations allow us to draw certain conclusions about the nature of the problem and the reasoning strategies used by the expert system. The present version of the Blackbox Expert System finds "good" solutions within an acceptable time frame using deliberate reasoning strategies.

Much of our knowledge about the nature of searches in Blackbox was obtained while debugging the system. One of the first facts to emerge is that if exhaustive searches are applied too early, they can take several hours to complete. This lead to extensive refinement of the knowledge used to determine when and how exhaustive searches are attempted. The knowledge defined within the expert system is a substitute for the visual assessment performed by a human player. Given the difficulty of modeling this type of knowledge, it is neither as accurate nor as complete. In spite of this, the expert system achieves an acceptable level of performance, as demonstrated by the results in figure 4.6.1 on the following page.

grid	balls	games	score	shots	errors	time
7 x 7	5	20	16.25	11.4	0.1	4.19
7 x 7	6	19	16.95	12.25	0.0	5.5
10 x 10	5	20	19.85	12.95	0.05	13.45
• 10 x 10	6	98	22.255	15.4	0.051	18.928
• 10 x 10	7	25	28.36	21.24	0.24	23.32
• 15 x 15	6	7	28	16.571	0.0	104.429

(a) Expert system results *exclude* exceptional games

grid	balls	games	score	shots	errors	time
*10 x 10	6	101	22.386	15.525	0.059	26.218
*10 x 10	7	26	28.538	21.192	0.0231	26.615
*15 x 15	6	8	27.625	16.25	0.0	159.0

(b) Expert system results *include* exceptional games

grid	balls	games	score	shots	errors	time
10 x 10	6	105	22.253	na	0.1	na
10 x 10	7	39	25.487	na	0.38	na

(c) Results obtained by human players

In the tables above, *grid* indicates the grid size, *balls* indicates the number of hidden balls, *games* gives the number of games that were played, while *score*, *shots*, *errors* and *time* (in minutes) are the averages obtained in these games.

Average results obtained by the expert system and by human players. The results in figure (a) exclude exceptional games from the averages marked with •. In figure (b), the results marked with "*" include games which took an exceptional amout of time to solve. The averages in (c) were obtained by human players.

There were three games on the 10 x 10 grid with 6 hidden balls which took an exceptional amount of time to solve: 174, 273, and 346 minutes. One game in the 7 ball series on a 10 x 10 grid took 109 minutes, while one game on the 15 x 15 grid took 542 minutes to complete. Correct solutions were found in all cases.

Figure 4.6.1: Performance measurements

The results displayed in figure 4.6.1 clearly reveal the relation between the problem's complexity and the game configuration. As the number of hidden balls increases from five to seven, there is a corresponding increase in the number of shots and the time required to solve the problem. The differences between individual games are a consequence of the relative ball locations. Figure 4.6.2 gives the maximums (worst case) and minimums (best case) obtained by the expert system with a 6 ball game on a 10 x 10 grid.

	series	games	score	shots	errors	time
a)	S7	10	13	8	0.0	56
b)	S7	6	14	8	0.0	9
c)	S6	7	22	13	0.0	4

(a) Minimums obtained with 6 balls on a 10 x 10 grid

a) The minimum score and shots obtained with a relatively high time.
b) Minimum shots with low time requirements.
c) Minimum time requirements.

	series	games	score	shots	errors	time
a)	S10	2	38	30	0.0	8
b)	S8	8	35	29	0.0	38
c)	S11	5	20	13	0.0	346
d)	...	...	...	...	1.0	...

(b) Maximums obtained with 6 balls on a 10 x 10 grid

a) The maximum score obtained with the maximum number of shots. (Note that if all shots were fired, the score would be 40).

b) The second highest score and number of shots. This is provided for comparison. Note the difference in time requirements.

c) The maximum time required to solve a game.

d) The highest number of errors in a final hypothesis. This indicates the soundness of the heuristics used for guessing ball locations.

Figure 4.6.2: Maximums and minimums obtained by the expert system

The increase in errors with an increase in hidden balls is due to the more common occurrence of ball locations that require guessing. The large leap in time requirements with a 15 x 15 grid indicates the non-linearity due to the increase in possible alternatives on the larger board. These results confirm the relation between complexity and uncertainty we described in section 4.1. As stated, the complexity of the problem is determined by the size of the grid, the number of hidden balls, and the location of these balls relative to each other on the grid. This relation is illustrated in figure 4.6.3.

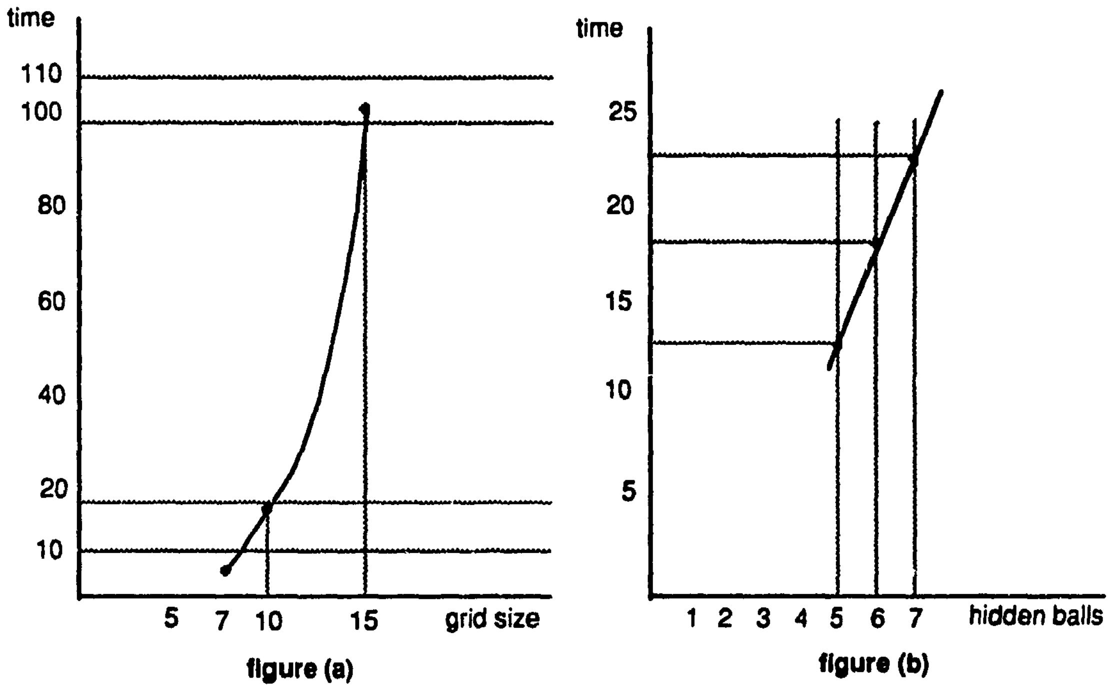


Figure (a) plots *grid size* against the *average time* (in minutes) required to solve a six ball game on a 7x7, 10x10, 15x15 board. As can be seen, the time requirements increase non-linearly as the board becomes larger. The values are taken from figure 4.6.1 (a).

Figure (b) plots the *number of hidden balls* against *average time* on a 10 x 10 grid. In this case, the increase in time appears linear. The values are taken from figure 4.6.1 (a).

Figure 4.6.3: Problem complexity

The definition of "acceptable" solutions provided in section 4.1 states that they should be comparable to that of human players. It is difficult to establish a direct comparison between the two because of (a) the difference in their approach, (b) time measurements are not comparable, and (c) a human's performance is highly variable. The human player is impulsive whereas the expert system is systematic. The expert system does not make errors due to oversight or distraction, but it may spend an excessive amount of time analysing possibilities that a human might discard with one glance. An accurate measurement of time requires a controlled environment to ensure that human players are not otherwise occupied. This is difficult to provide for a large sample. Another consideration is that humans get bored if the game is too complex or the session is too long, with direct consequences on their performance. Figure 4.6.4 compares the results of the expert with those of a person playing identical games. (The human's results are taken from [36]).

player	grid	balls	games	score	shots	errors	time
expert	10 x 10	6	3	19.667	13	0	9.67
human	10 x 10	6	3	21	13.37	0	7.33

This figure compares the results of the expert system and that of a human when playing the same game configurations.

Figure 4.6.4: Comparison between the expert system and a human player

Validation of the expert system is straightforward. The expert system was run on over 200 randomly generated games, and measurements were retained for 151 games. (The measurements for individual games are available on request). All discrepancies between the expected performance and actual measurements were examined to determine their cause, and relate them to the expert system's decisions.

Our validation of the expert system's performance is based on the fact that on average its results are equivalent to those obtained by human players. The expert system found solutions for all the games it attempted. Furthermore, the solutions in which errors occurred involved guessing in all the cases that were examined. We feel that the size of the sample space used for testing provides a good case to ensure that it is representative of the problem instances that can be encountered. The exceptional cases are noteworthy in that they reveal how time consuming an unconstrained search can be when an inappropriate strategy is chosen.

Our understanding of the Blackbox Expert System allows us to make the following conclusions about its operation: If the searches are not well constrained they become intractable. The constraints are determined by reasoning. These constraints are related to:

a. The state of the grid.
b. The actual and available number of balls.
c. The state of possible and evaluated shots.
d. The contents of the vicinity.
e. The number of alternative hypotheses for a beam.
f. The number of unsolved beams.

The expert system controls these factors by making a deliberate decision about which activity it will pursue, and when. Furthermore, it has a choice of reasoning strategies for performing each activity. This choice has an impact on performance. The decision involves a tradeoff between accuracy and the time required to solve the problem. The expert must weight the advantages of a more accurate intermediate solution against the cost of a more exhaustive search. These properties make Blackbox an ideal candidate for the study of DPS issues.

Chapter 5
A Distributed Blackbox Expert System

A design for the Distributed Blackbox Expert System (DBB) is proposed in this chapter. The purpose of the DBB system is to construct a testbed for studying coordination and cooperation in a DPS system. One of the objectives of our research team is to experiment with different organisations of distributed agents, and to compare their problem solving performance. This objective leads to three design requirements for the DBB testbed: (1) A problem solving structure based on the organisational model; (2) An expert system composed of multiple agents to solve the DBB problem; and (3) A testbed environment, which provides facilities for studying different organisations.

Although alternative organisations for DBB are considered, only one organisation is discussed in detail. This consists of a team of agents obtained by partitioning the Blackbox problem geographically. Within its knowledge base, each agent incorporates the Blackbox expertise described in the previous chapter, augmented with the knowledge necessary to solve the problem in a distributed way. This knowledge governs how the agents interact by relating the agents' expertise to their role in the problem solving process. The DBB testbed system provides the facilities for developing, observing, and evaluating the performance of the distributed expert system.

5.1 An Organisational Model for DBB

In this section we propose a general framework for implementing an "organisation" of DPS agents (see section 2.5). The framework provides a structure for representing the knowledge related to coordinating system activity without imposing any particular coordination scheme on the system, apart from that of "*organisational structuring*". An organisation is created by defining an "*organisational role*" for each agent within the system. An agent's role describes the relationship between an agent's problem solving activities, and its interactions with the other agents in the organisation. The agents' combined roles define the relationships that link the set of agents into a problem solving whole. The modular structure used to represent an agent's organisational knowledge makes it easy to experiment with alternative coordination schemes and organisations.

An agent's *organisational role* relates its local problem solving state to that of the organisation. It describes how the agent reacts to the changes that it "perceives" in the state of the organisation, and how the agent changes the state of the organisation as a result of what it perceives. This involves defining: (1) The changes, or conditions in the organisation that are relevant to a particular agent; (2) The agent's expected response when these conditions arise; and (3) The way in which the agents convey the information pertaining to these conditions among each other. An agent's role is represented by its *responsibilities* and its *communication policy*, which are discussed below.

An agent may determine the relevant conditions itself in the course of its reasoning, or it may receive messages from the other agents indicating the existence of these conditions. An agent's reaction to the appearance of these conditions will be defined by the agent's *responsibilities*. These will identify what locally produced conditions are relevant to which other agents in the organisation, and how the agent responds to externally produced conditions that are communicated to it by the other agents. Thus, an agent's responsibilities will relate its individual activities to the flow of information through the organisation, with outgoing messages resulting from locally produced changes, and incoming messages reflecting changes in the global problem solving state produced by the other agents.

The actual exchange of messages among the agents is governed by a *communication policy*. The communication policy defines how an agent derives information from the messages it receives, and how an agent conveys information through the messages it sends. This distinction between "information" and the actual transmission of messages makes it easy to experiment with different protocols for the exchange. The protocol will establish at what point the agent perceives the information conveyed by the messages that it has received, thus controlling disruption and distraction of the agent. The protocol may construe "information" from some sequence of messages, or their absence, if a "knowledge based protocol" is used. The communication policy will also describe what messages are sent when the agent wishes to convey information to other agents.

An agent's responsibilities describe what control decisions it is responsible for, and the type of decision process it employs. Its communication policy describes how the agent interacts with other agents in order to obtain the information that it requires for making its decisions. Their combination defines the specific flavour of the coordination scheme being used. The scheme necessarily involves *organisational structuring* with *sophisticated local control*, of varying sophistication according to the agent's role. Local control decisions are realised through *negotiation*, *multiagent planning*, an *FA/C approach*, or by *decree*. (Figure 2.2.3).

Within an organisation, each agent integrates knowledge about its organisational responsibilities and its communication policy with the information at hand to derive a "view" of the global problem solving state. This view incorporates local state information with the information received from other agents, and any additional information that can be derived from it. The agent employs a local *planning function* to choose an "optimal" activity based on this view. This activity concerns transforming its local problem solving state in accordance with what it "perceives" of the global system, and transforming the global system state by communicating any relevant changes in accordance with its organisational responsibilities.

The requirements described above can be summarised as follows. An agent has a local working memory in which it maintains its local problem solving state. At the same time, the agent maintains a view of the global problem solving state, related to its local state and the information sent to, or received from the other agents. The agent chooses what activity it will perform according to its organisational responsibilities and its view of the global problem solving state. The agent transforms its local problem solving state by exercising its local problem solving abilities. It influences the global problem solving states of the other agents by communicating the information defined by its responsibilties in the organisation. The communication of information leading to transformations of the organisation's global state is regulated by an agent's communication policy. The actual transmission of messages over the network is handled by a communications module. The framework for implementing such an agent is given in figure 5.1.1.

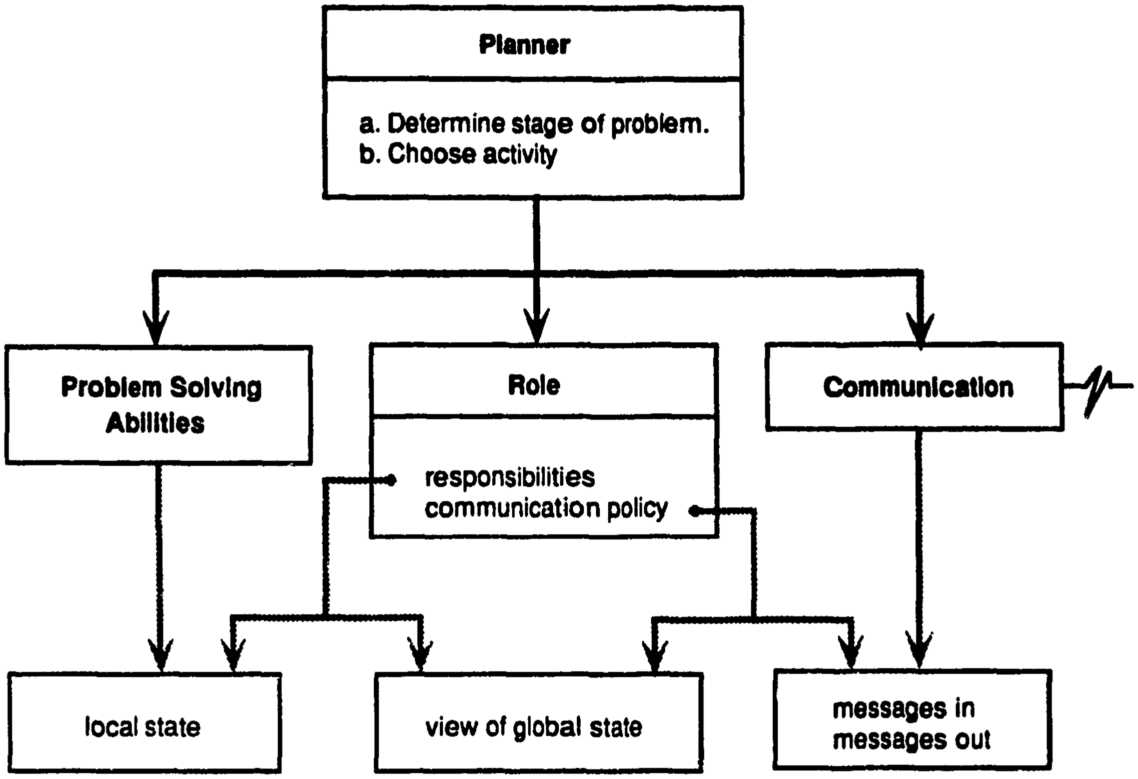


Figure 5.1.1: Structure of an agent in an organisation

5.2 Description of the Multiple Agent Blackbox Problem

As previously stated, DBB is obtained by partitioning the Blackbox grid into four equal quadrants, and assigning one agent to each quadrant (figure 3.4.2). Each agent solves its Blackbox subproblem using the reduced field of view and firing range that results from the partitioning. This reduction limits the information that an agent can acquire on its own within its quadrant. The agents must obtain the information they lack by communicating with each other in order to solve their individual subproblems. The agents' combined partial solutions constitute the total solution to DBB. The goal of finding a correct solution with a minimum score as rapidly as possible implies that the agents must coordinate their actions and interactions effectively.

The decomposition of the problem introduces several factors:

F1) An agent's view of the game grid is restricted to what occurs within its own quadrant.

F2) An agent can only "see" shots that enter or exit the grid along the two external sides of its quadrant.

F3) An agent can only fire shots into the grid from the two external sides of its own quadrant.

F4) Although an agent knows the total number of balls hidden within the global grid, it does not know the number of balls hidden within its own quadrant.

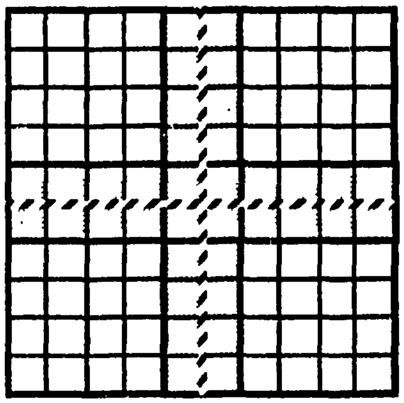
F5) An agent must communicate with the other agents in order to obtain all other information concerning the game.

F6) The agents must be synchronised so that they have a more or less consistent view of the beginning and end of each game.

An agent's view of the game is restricted to what occurs within the quadrant it is assigned. Each agent also views an overlapping region as shown in figure 5.2.1. This overlap is defined because its contents affect trajectories on either side. The overlap results in adjacent agents sharing information about the contents of squares on either side of the internal boundary that separates them. All agents share information concerning the contents of the four squares at the centre of the grid. Some definitions follow.

Def^n: An **internal boundary** delimits and separates the quadrants of adjacent agents.

Def^n: The **overlap** consists of the two columns or rows of grid squares within which the internal boundary lies.



⫽⫽⫽⫽ Internal boundary separating adjacent agents.
□ The overlap region shared by adjacent agents.

Each agent is responsible for the area contained within the internal boundary. In addition, each agent can view and modify the overlap region adjacent to its internal boundaries.

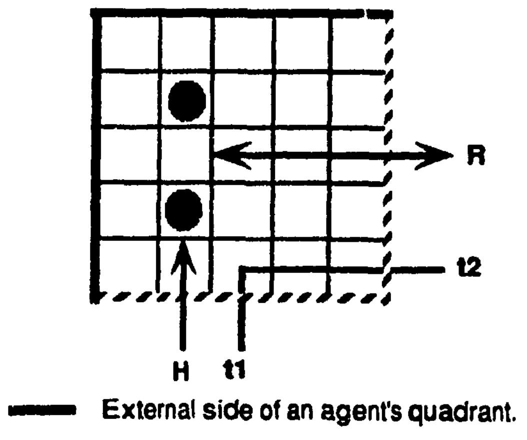
Figure 5.2.1: Overlapping regions shared by agents

An agent can only "see" beams that enter or exit its quadrant through its two external sides. A single beam may pass through several quadrants. Beams that originate in another quadrant, and that are subsequently absorbed, reflected, or transit through an agent's quadrant without exiting along one of its external sides are not immediately visible to the agent. An example is given in figure 5.2.2. An agent only becomes aware of such beams when it is informed of their presence by the other agents. The following definitions are used:

Def^n: An **external side** is the side of a quadrant which is not adjacent to another agent's quadrant.

Def^n: A **multiquadrant beam** is a beam that passes through more than one quadrant in the course of its trajectory.

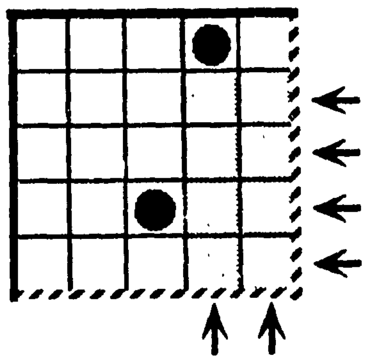
Def^n: A **transparent beam** is one segment of a multi-quadrant beam that lies within one agent's quadrant, and that does not exit from that agent's quadrant through any of its external sides.



H is an absorbed beam, R is a reflected beam, and t1-t2 are the entry and exit points of a beam that transits through the quadrant. All three beams are "*transparent*", as none exit through an external side of the quadrant.

Figure 5.2.2: Transparent beams

An agent may only fire into the grid along the two external sides available to it. Certain board configurations may result in the condition where an agent cannot reach all areas in its quadrant by firing its shots alone. An example is given in figure 5.2.3. Information about such inaccessible areas can only be derived from *transparent beams*.



Information about the shaded area □ can only be derived by having some other agent fire the transparent beams indicated with ➔ .

Figure 5.2.3: Inaccessible area within an agent's quadrant

Although an agent knows the total number of balls hidden within the global grid, it does not know the exact number hidden within its own quadrant. However, it can infer certain things. The number of *actual* and *available* balls within its quadrant is based on the total number of balls in the board, less all balls located outside its quadrant, including those in external "ball vicinities". This is defined below. Thus, if the total is three balls, and two other agents have identified "ball vicinities", then the agent will have one *actual ball* to locate.

Def^n: An **external ball vicinity** is a ball vicinity that is located in some other agent's quadrant from the point of view of an agent.

Def^n: An **internal ball vicinity** is a ball vicinity located within some agent's quadrant from that agent's point of view.

An agent's restricted view of the grid has implications on how the agent chooses shots and analyses beams. For both actions, the agent has less information with which it can reason. This fact increases the amount of uncertainty that it must deal with. When choosing shots, an agent can only evaluate that portion of a shot which lies within its own quadrant. Thus it can only determine the local value of the shot, and must communicate with the other agents in order to establish the global contribution of its shots. The analysis of beams is similarly limited. The agent only hypothesises about the partial trajectory that lies within its quadrant, and the point at which the beam transits into another agent's quadrant. The agents must exchange information concerning transit locations in order to establish a correct hypothesis. How this information is exchanged will depend upon the organisation.

Another practical consideration is synchronising the agents so that they have a consistent view of the beginning and end of a game. This is necessary so that individual agents begin reasoning at approximately the same time. It is also necessary for all the agents to know when the game is over.

The differences F1 to F6 described in the preceding paragraphs, establish what information the agents must exchange to solve the DBB problem. Given an agent's reduced sphere of action, the agents must communicate with each other to choose globally optimal shots, to request inaccessible shots, and to determine what has happened when beams disappear, appear, or transit through their quadrants transparently. The agents must keep each other informed of the number of balls left to locate in the game. Adjacent agents automatically share information about the contents of the overlapping region between them. The agents employ an appropriate protocol to signal the beginning and end of a game. The information requirements are summarised in table 5.2.1.

Table 5.2.1: Information requirements

a. The information necessary to establish ***the global contribution of locally available shots***

b. ***Shot requests*** for inaccessible shots.

c. ***Beams that appear or disappear*** from an agent's quadrant, to inform the other agents of a beam's possibly transparent passage through their quadrant.

d. The ***transit location of beams*** that enter or exit an agent's quadrant along an internal edge, to establish that beam's partial trajectory within the quadrant.

e. The ***contents of grid squares within the overlapping region*** shared by adjacent agents.

f. The ***number of balls*** that remain to be located.

g. Synchronising the ***beginning*** and ***end of a game.***

The decomposition of Blackbox into DBB changes the nature of the game in several ways. The DBB system must accomodate situations that do not occur in the single player version of the game. The way in which these situations are handled could simulate different circumstances. We have a certain amount of leeway in defining how the decomposition of the game affects the problem. The circumstances that are simulated will depend upon how DBB handles the situations outlined below. In all three cases, there is a choice between simplicity and realism, with realism requiring more domain specific knowledge.

P1. How do agents determine the behaviour of a shot?

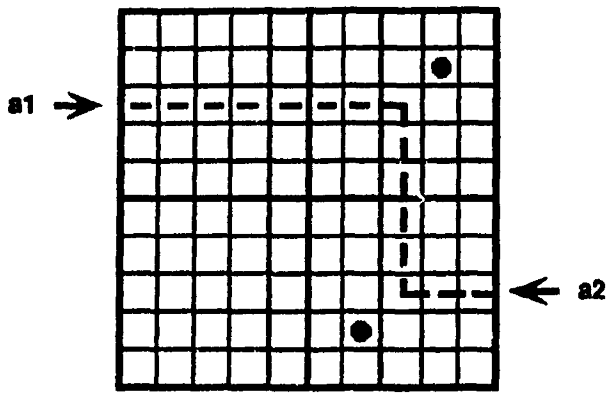
P2. How are multiquadrant shots assigned a unique identifier?

P3. How do agents avoid duplicating shots?

With the single player game, when the agent fires a shot, the agent is immediately informed of the shot's behaviour by the firing module. This constitutes part of the game's definition; the agent knows the shot's behaviour with total certainty because the shot's behaviour is defined as the observable outcome of firing. The same is not true in the distributed game. Although the agent can "see" shots that are reflected or that exit from the grid within its quadrant, it cannot see shots that are absorbed or exit from another quadrant. The agent is only aware that its shot has disappeared. There are two ways of dealing with this partial information: (1) Have the DBB firing module inform the agent of the shot's behaviour the same as in Blackbox; (2) Consider the partial information a consequence of distribution which the agents must resolve. The first requires no additional reasoning, whereas the second involves an exchange of information using an appropriate message protocol, whereby the firing agent establishes whether or not some other agent has "seen" its shot. The first is simple, whereas the second is more realistic.

Another consideration is the way in which agents identify multiquadrant beams in order to relate their individual partial hypotheses to the same beam. Once again we have a choice between simplicity and realism: (1) Use a global identifier for each beam, assigned within the quadrant of the firing agent; (2) Have agents relate outstanding disappearances to sudden appearances through reasoning. The first requires no reasoning whereas the second may involve considerable reasoning, particularly if more than one entry or exit location is outstanding. If this is the case, then the problem assumes an entirely new level of complexity.

The third consideration is the problem of duplicated shots. This problem does not occur in the single player game because an agent is only capable of firing one shot at a time. In DBB, the unpredictable nature of beams makes it impossible to avoid duplicated shots unless firing is serialised. Without serialised firing, there is no way of ensuring that two agents do not simultaneously fire the same exiting shot from its opposite ends, as shown in figure 5.2.4. In fact, the contrary is more likely, and corresponds exactly to the "*ambiguous situations*" described by Ginsberg (figure 2.4.2). The options are (1) Serialised firing, versus (2) Concurrent firing. With the first, all duplicate shots are avoided by using some form of mutual exclusion for the firing privilege. With the second, the agents can fire concurrently, but duplicate shots are possible. The choice can be based on the relative cost of additional shots compared against that of additional processing time.



If the agent in the top left-hand quadrant fires shot "a1" at the same time that the agent in the bottom right-hand quadrant fires shot "a2", then both agents will fire the same shot, thus duplicating it.

Figure 5.2.4: Simultaneous firing of the same shot

A problem decomposition in DPS raises the following issues in DBB concerning how the multiple agents coordinate their actions and interactions. The way these are handled will determine how the overall system maintains a "common focus", and the degree of "opportunism" possible.

S1. How do agents choose what activity each will perform?
S2. How do agents choose globally optimal shots?
S3. How do agents exchange information related to analysing beams?

Determining how agents choose individual activities that are globally "optimal" is a strategic problem. The agents evaluate their local options individually, and then somehow communicate with each other so that only those activities which are "optimal" for the entire group are performed. Accomplishing this involves determining the best way of propagating the necessary information through the system.

The flow of information is a function of the agents' roles within the organisation. The agents' roles are defined by the agents' responsibilities and communication policies, which determine how the information is propagated.
Getting the individual agents to choose globally optimal activities involves choosing an organisation that realises a coordination scheme whereby individual agents arrive at cooperative decisions.

In the system design, wherever possible I have chosen those options which require the least domain specific knowledge while retaining the interesting features of a DPS system. Where domain specific knowledge is involved, it can be added at a later date. My initial design goals for the DPS system may appear ambitious. I believe that it is easier to consider the widest range of possibilities at the design stage, and then reduce the agents' abilities rather than doing the contrary. This has lead to the following choices for P1 to P3, and S1 to S3:

P1. **The firing agent is informed of the shot's behaviour by the firing module immediately after firing.** This is consistent with the rules of the original game, and simplifies knowledge requirements.

P2. **All beams are assigned a unique global identifier in their quadrant of origin.** The alternative requires too much DBB specific knowledge.

P3. **Agents may fire concurrently.** This provides a greater degree of concurrency, but requires establishing the tradeoff between shots and processing time.

S1. **The agents are organised in a team.** The team imposes the least restrictions when analysing the problem, and the most flexibility during development. It is also more easily implemented (agents are identical), and intuitively interesting (many opportunities for studying cooperation).

S2. **The choice of shots is decentralised.** This is consistent with concurrent firing. If it subsequently proves too costly, it can easily be modified by adding a mutual exclusion mechanism.

S3. **An FA/C approach is used for exchanging the information concerning beams.** This approach is appropriate because the solution is found by exchanging tentative hypotheses in order to eliminate conflicting alternatives, and thus converge on the correct solution.

The benefits that can be obtained by partitioning the problem and solving the resulting subproblems in parallel are open to discussion. The problem decomposition changes the nature of the blackbox game, increasing the amount of uncertainty present. Much of the information that a single agent uses in solving the problem is no longer immediately available. The interactions necessary to obtain the relevant information introduce delays that do not occur in the single player version of the game. The DBB expert requires additional knowledge to deal with all this, and thus increases the system's total processing requirements.

On the other hand, problem decomposition and distribution have certain advantages. The smaller grid size and lower number of balls within a quadrant correspond to a much less complex game. The ability to analyse several beams simultaneously may reduce total processing time. The timely exchange of intermediate hypotheses may lead to the earlier elimination of incorrect alternatives. All of these are potential benefits of using a DPS system.

DBB's performance will depend upon how effectively the organisation exploits the DPS potential. Although the additional overhead of an organisation may outweigh the benefits on a 10 x 10 grid, the same is not necessarily true for a larger board. The results obtained by a single expert indicate that the expert's performance degrades in a non-linear fashion as the problem becomes more complex (figure 4.6.3). DBB may be capable of finding "better" solutions for such complex problems. Establishing at what point the DBB organisation becomes a viable alternative to a single agent is an issue of interest, to be explored.

5.3 The Testbed Environment

The environment in which the DBB system will ultimately operate should provide facilities for developing, evaluating, and experimenting with alternative organisations. These facilities would be identical to those required to debug, finetune, and validate an initial DBB prototype. They allow the user to (1) run game sessions; (2) collect performance measurements; (3) monitor intermediate execution; and (4) intervene during execution if so desired. The first two fulfill the same functions as the *session component* described in the preceding chapter. Monitoring of execution is concerned with how the user observes the system's operation. The need for user intervention arises because few if any development environments exist for constructing DPS systems.

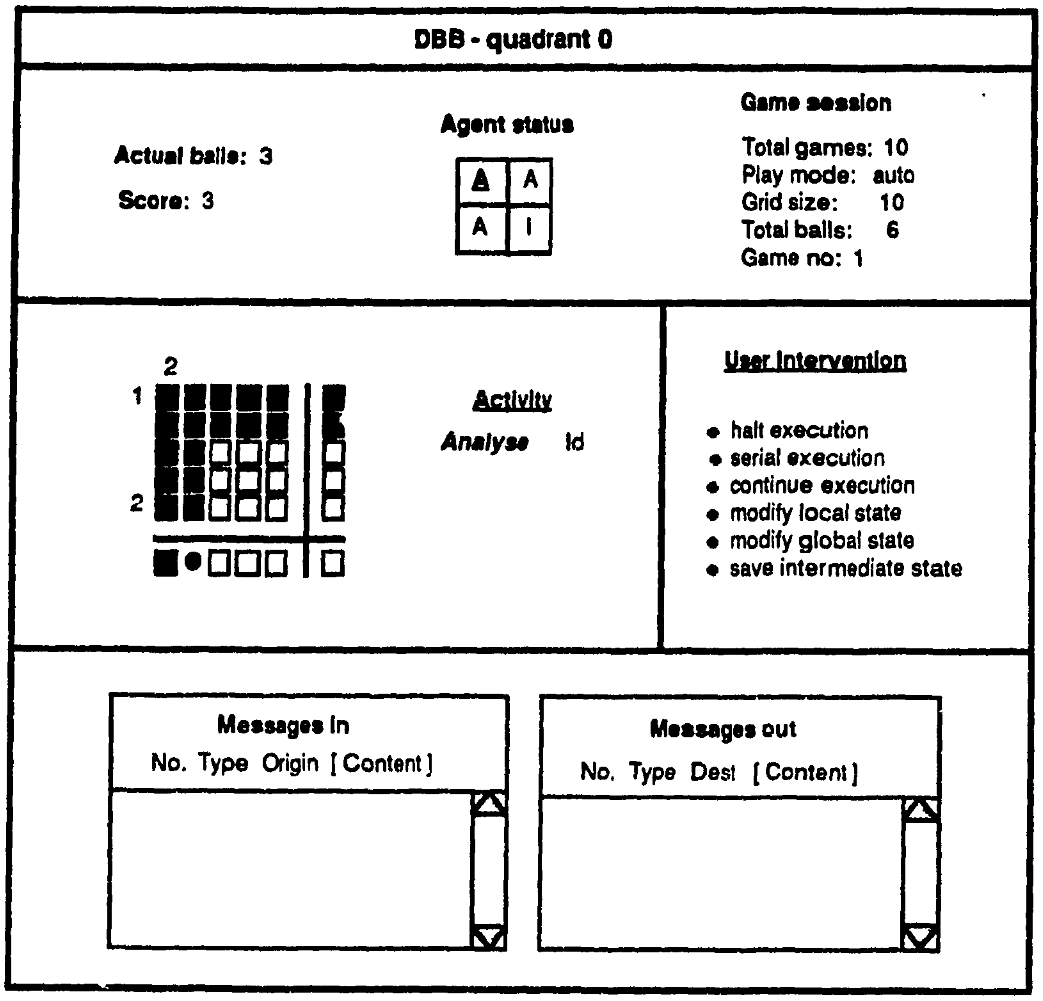
Game sessions: The ability to run game sessions will prove to be as valuable in validating the performance of DBB as it was for Blackbox. The *DBB session component* performs the same functions as its Blackbox equivalent (figure 4.2.2): **(1)** session initialisation; **(2)** control of game execution; and **(3)** calculation of session statistics. In addition to generating random boards and reading board configurations in from a file, the system should allow the user to specify that the system starts at an intermediate stage within a problem. Thus, the user can restore some problem state and reattempt execution. This will reduce debugging time considerably. Its presence implies that the user must have some way of stopping execution, and saving intermediate solutions for all agents. This is discussed under interventions.

The game session is controlled by a single, designated agent, who is called the *session controller*. It sets up each game, sends the board or file information to the other agents, and then broadcasts a "*begin message*". Once an agent receives the begin message, it starts working on its subproblem. Each agent broadcasts a "*termination message*" when it has solved its quadrant. The game ends when all agents have terminated (i.e. each agent has received a termination message from all the other agents). The session controller then polls the agents to collect the game results.

Performance measurements: Along with the performance measurements described in chapter 4, (score, shots, amount of time, and number of errors), the agent's idle time and network traffic must also be measured. Idle time is the amount of time an agent performs no problem solving activity. Network measurements include line utilisation as an average, at its peaks, and in relation to problem solving. The last requires that messages be classified by type, and ordered in relation to the agent's local decisions and activity. Messages are classified as *incoming*, or *outgoing*, and divided according to content. Messages relate to (a) choosing and firing shots; (b) analysing beams; (c) the number of balls; and (d) the contents of overlapping regions. It is possible to have further subdivisions within each of these categories, and measure the utility of messages by recording message access and the subsequent changes to the solution.

Monitoring intermediate execution: The facilities available for monitoring will determine how easily the user can understand the operation of the system. There are two ways of monitoring: (1) observing the screen display on-line, or during execution; and (2) off-line, or post-mortem analysis of data. The first is useful for keeping track of what is happening, but it may be difficult for the user to simultaneously follow the activity of all the agents. The second method is useful for tracing the solution process, but may involve considerable analysis by the user.

During execution, the quadrant of each agent is displayed separately on the screen. The display for each agent consists of its local problem solving state, and the global problem solving state, as "perceived" by the agent according to the messages that it has sent and received. The display is similar to that used for a single expert with certain exceptions due to the restricted field of view and the addition of message traffic. The screen contents are outlined below. The screen display for one agent is illustrated in figure 5.3.1. The user can view all the quadrants together by displaying them on the same screen.



DBB - quadrant 0

Actual balls: 3

Score: 3

Agent status

A	A
A	I

Game session

Total games: 10
Play mode: auto
Grid size: 10
Total balls: 6
Game no: 1

Activity

Analyse Id

User Intervention

- halt execution
- serial execution
- continue execution
- modify local state
- modify global state
- save intermediate state

Messages In

No. Type Origin [Content]

Messages out

No. Type Dest [Content]

The windows displayed on the screen for one agent are from left to right, and top down: the *status window*, the *grid window*, the *user intervention window*, and the *message window*. The explanation of their contents follows.

The ***status window*** gives the number of *actual balls* available to the agent, the agent's *score*, the *agent status* - which indicates which agents are Active and Inactive in which quadrants (when all agents are 'I' then the game is over), and the *game session status* - which informs the user about the session.

The ***grid window*** displays the agent's *quadrant and the overlap region*, and the agent's *current activity*. Activities consist of Plan, Communicate (what), Wait (identifies what), Choose shot, Shoot, and Analyse (beam id).

The ***user intervention window*** gives the user access to the functions displayed by clicking on the associated button and opening a window. Highlighting is used when appropriate.

The ***message window*** displays incoming and outgoing messages in seperate menu windows that allow the user to browse through them. All messages are assigned a number which gives their local order relative to each other.

Figure 5.3.1: Screen display for one agent

1. The grid area displayed is limited to *the agent's quadrant and the overlapping region* that it shares with adjacent agents. The view of the overlap may be inconsistent due to communication delays.

2. The number of balls expresses the number of *balls available to the agent* for constructing hypotheses within its own quadrant. This may vary within each quadrant due to "ball vicinities" and communication delays.

3. *The agent's current local score*. It is counted as one point for each fired beam that passes through one of the agent's external sides.

4. *The agent's current activity*. Possible activities are identical to those of the single expert, (evaluate, analyse, invalidate, integrate, guess - figure 4.3.1) with the addition of: *communicate* and *wait*. These are explained in the following section.

5. The *identifier* and *behaviour of fired shots*.

6. *Incoming and outgoing messages*. Incoming messages are displayed on arrival. They are also displayed when they are accessed by the agent for decision purposes. They must be classified by type, and otherwise ordered so that the user has a coherent view of network activity in relation to local problem solving.

Care must be taken in choosing what data is retained for analysis after execution, as a prodigious amount of raw data could be produced. Indiscriminate data collection would result in increased handling costs, and wasted time on the part of the person attempting to interpret the data. The objective of retaining data is to provide a trace of the operation of the system. This can be realised by tracing the local activity of an agent in relation to the messages it receives and generates, for each agent in the system. The messages are classified according to the categories used for performance measurements. (Namely, incoming or outgoing, and related to choosing shots, analysing beams, the number of balls, or the contents of overlapping regions). Outgoing messages are easy to place in relation to local activity, as they are the result of a local decision. Incoming messages are related to local activity by describing their point of arrival, and the balls located, squares emptied, shots fired, and beams analysed as a consequence of integrating the messages into the agent's local planning process.

No attempt is made in our design to reconstruct an accurate "*snapshot*" of the global problem solving state at one point in time. This is considered irrelevant because the agents operate in an imperfect world. (They do not have accurate information, and the cost of collecting this information is prohibitive.) The objective of the system is to determine how the agents can employ the possibly inaccurate information at hand to make "better" decisions more rapidly. Thus, the problem is traced from the perspective of each agent, rather than that of an omniscient entity with an overview of global system activity.

Intervention during execution: One reason for intervention is to save intermediate problem states for subsequent reexecution. Another is to provide the user with a totally controlled execution environment. Most expert system development shells include builtin debugging facilities that allow the developer to "step through" the reasoning process; modify the value of variables; backtrack; try alternative solution paths; save intermediate states; and otherwise contemplate how the solution is evolving. In DBB, although each agent operates in such an environment, the interactions between agents will disrupt the normal course of events. The interactions can be controlled by using a serialised "*execution privilege*" that would activate one agent at a time. This would allow the developer to "step through" the system in a controlled sequence of actions, interactions, and reactions among the agents, with the ability to save intermediate states along the way.

The ability to control execution and save intermediate results is provided by an interface for *user intervention*. The interface allows the user to halt execution; modify the local problem solving state; modify the global problem solving state visible to the agents (indirectly by modifying the local state, or directly by passing messages); control nessage transmission; modify the content of messages (sent or received); save an intermediate state; continue execution; or continue execution in a serialised fashion.

The facilities necessary to run game sessions, collect performance measurements, monitor intermediate execution, and intervene during execution are modularised in the design of DBB. The structure of a DBB agent is illustrated in figure 5.3.2. This structure provides the environment in which the agents are developed, debugged, and validated.

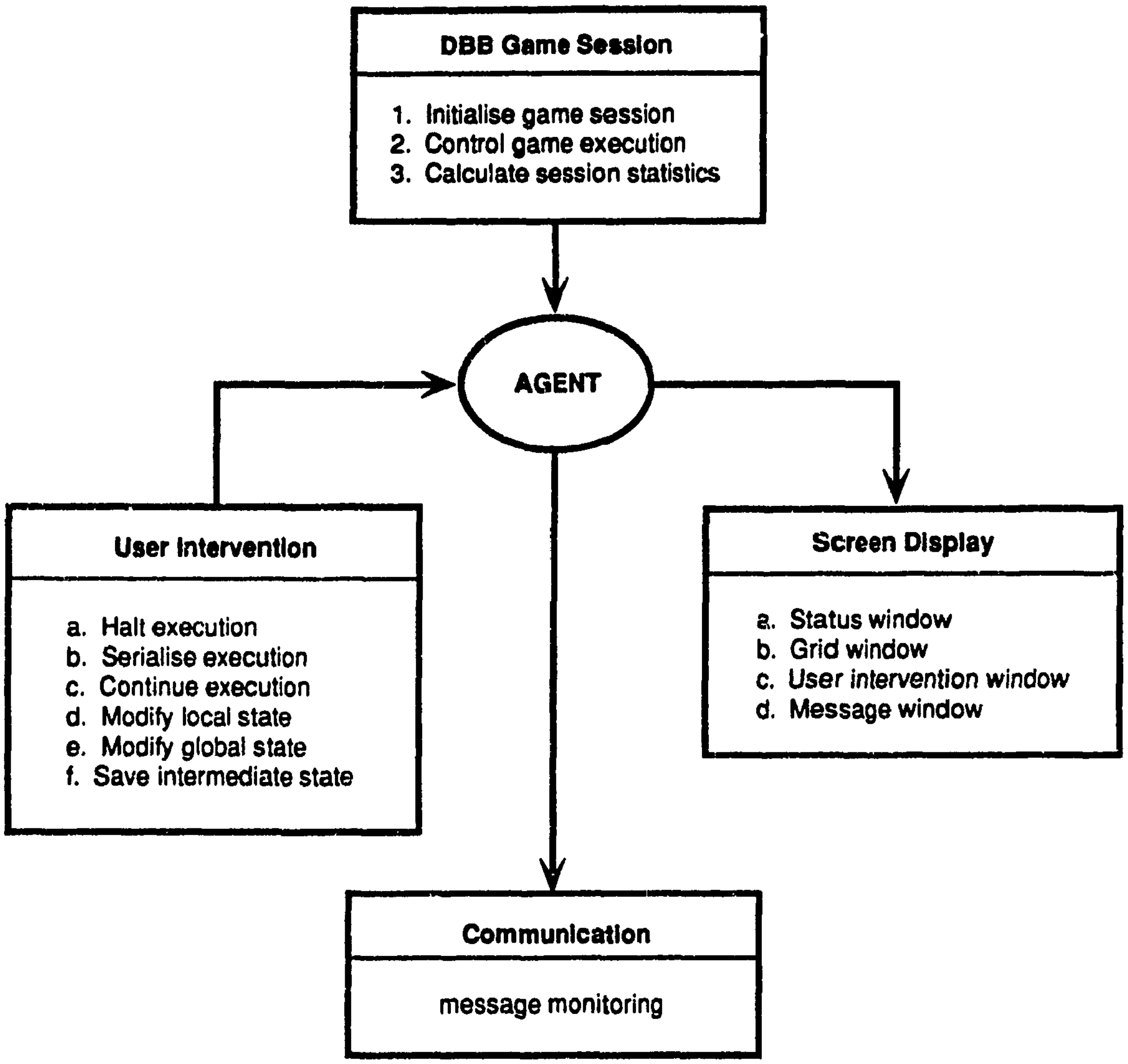


Figure 5.3.2: Agent within the testbed

5.4 The DBB Solution Process

The Blackbox solution process described in chapter 4 must be modified to accomodate the interactions that will occur between the agents in DBB. These interactions are necessary to convey the information listed in figure 5.2.4. Furthermore, the interactions are also largely unpredictable because the agents are dispersed, and they work on different subproblems at varying speeds. Thus, an agent's local processing is regularly disrupted by the arrival of messages from the other agents. The objective of this section is to establish the "optimal flow of messages" in order to minimise the negative consequences of "disruption" and "distraction", and also to find the best uses for "local idleness". This optimisation requires answering the following questions:

Sending Agent

"When should an agent send messages?"

"What messages?"

"To whom?"

Receiving Agent

"When should an agent use the received messages in its reasoning process?"

"What messages?"

"From whom?"

All Agents in General

"What should an agent do if it receives no messages?"

"What should an agent do if it has no goals to pursue?"

These questions must be asked in relation to each aspect of the solution process, viewed from the perspective of both the individual agent and the group as a whole. In the case of an individual DBB agent, the solution process involves the same problem solving activities as the Blackbox expert system, with two additions: *communicate*, and *wait*. Communicate describes how "information" is transformed into "messages", and "messages" into "information". Wait indicates the absence of all other activities. The activities of a DBB agent are summarised in figure 5.4.1, and in the descriptions that follow.

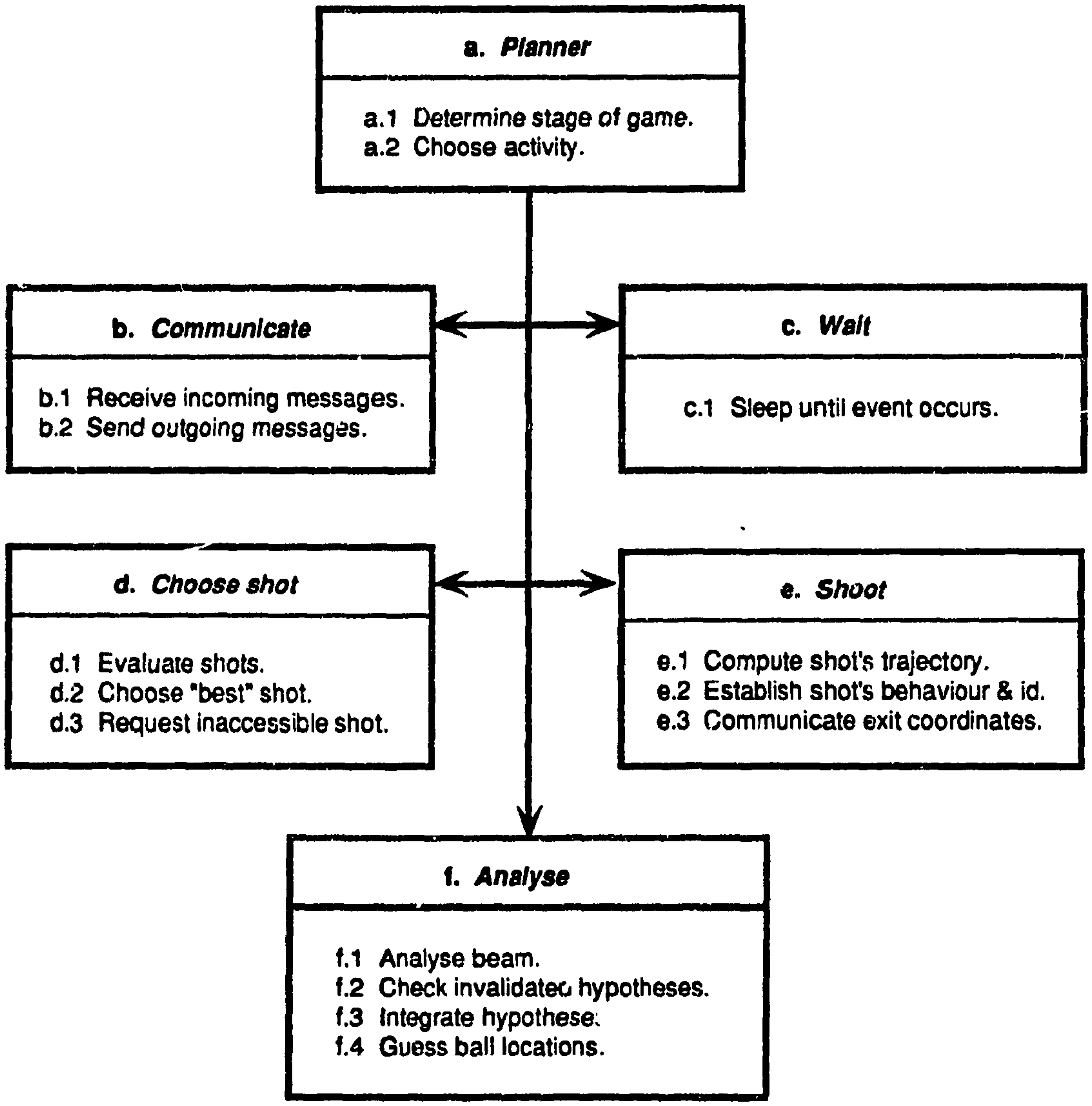

a. Planner
a.1 Determine stage of game.
a.2 Choose activity.
b. Communicate
b.1 Receive incoming messages.
b.2 Send outgoing messages.
c. Wait
c.1 Sleep until event occurs.
d. Choose shot
d.1 Evaluate shots.
d.2 Choose "best" shot.
d.3 Request inaccessible shot.
e. Shoot
e.1 Compute shot's trajectory.
e.2 Establish shot's behaviour & id.
e.3 Communicate exit coordinates.
f. Analyse
f.1 Analyse beam.
f.2 Check invalidated hypotheses.
f.3 Integrate hypotheses.
f.4 Guess ball locations.

Figure 5.4.1: Problem solving activities of a DBB agent

Planner: Planning consists of determining what stage the game has reached, and then choosing an appropriate plan of action or activity. The choice will depend upon what changes the agent has made in its local problem solving state, and what messages have arrived since the last plan was put into action. These constitute the two sources of change in an agent. Determining what stage the game has reached consists of integrating the two to get a possibly inconsistent view of the global problem solving state of the system. The agent then chooses an activity based on this view.

Communicate: Communication concerns the handling of incoming and outgoing messages by the *communications module*. The module provides the network services with which the messages are transmitted. It performs the logical to physical, and physical to logical transformations necessary to relate network traffic to the problem solving process. It creates messages from information, and derives information from messages, based on message content, the temporal relation of messages, and their origin or destination. The module sends or broadcasts outgoing messages, receives incoming messages, and performs all other intermediate storage, transformation, and classification functions. These include identifying message types, ordering messages, and generating interrupt signals on the arrival of specified messages, according to the established communication policy.

Wait: Corresponds to an idle state of the agent, during which the agent is not actively involved in problem solving. An agent waits when it has no goals to pursue, or it is waiting for communication. A waiting agent is awoken by an external event. An agent enters the wait state when:

The agent has solved its quadrant, but the global solution has not yet been found. When in this state, the agent is awoken by **(1)** A request from another agent; **(2)** A change in the global problem solving state (i.e. termination or hypothesis change).

The agent is waiting for a message from another agent to continue. In this case, the agent is awoken by **(1)** the arrival of the expected message; **(2)** the arrival of an overriding message. All other messages are ignored.

Choose shot: Performs all "reasoning" related to evaluating shots and choosing which shot to fire in a decentralised manner. The problem partitioning results in agents having two types of shots to consider: **(1)** Shots that an agent can fire itself; **(2)** Shots that an agent must request from another agent. The evaluation of shots that an agent performs must resolve several strategic questions: When should an agent fire a shot? When should an agent request a shot? How should an agent respond to shot requests that it receives? The knowledge required to deal with this is outlined below:

Determine the local value of shots. This is accomplished using the evaluation strategies described for the Blackbox expert system.

Determine the global value of shots. This covers two aspects: **(1)** inform other agents about the local value of a shot, and **(2)** derive the global value of a local shot from the information received from the other agents.

Choose the "best" shot. A number of additional factors are involved in the choice: **(1)** the type of shot (local, to request, or requested by another agent); **(2)** the hypothetical trajectory of the shot, (local or multiquadrant); **(3)** the likelihood of duplicated shots; and **(4)** whether the agent has any other outstanding shot requests that have not yet been satisfied.

Request an inaccessible shot. When an agent chooses an inaccessible shot, it must communicate its request to the appropriate agent, and then await that agent's response. The agents involved in such an exchange must (1) maintain a record of requested shots, (2) relate these requests to information about beams, and (3) have the ability to cancel outstanding requests if so desired.

Shoot: The DBB firing module has to concern itself with a number of factors. The computation of a fired shot's trajectory is not distributed. The entire trajectory of a shot is computed by the firing module of the quadrant in which the shot is fired. If that shot exits in another quadrant, the firing module is responsible for sending the shot's exit coordinates to the remote quadrant. The firing module also handles the inverse situation of receiving exit coordinates from another quadrant. Its functions are outlined below:

1. Compute the shot's trajectory over the board.
2. Establish the shot's behaviour and global identifier.
3. Communicate exit coordinates to another quadrant.
4. Receive exit coordinates from another quadrant.

Analysis: The DBB agents would employ the same reasoning strategies as the Blackbox expert (analyse, invalidate, integrate, and guess), with the difference that one or both ends of some beam hypotheses may consist of a transit location through an internal boundary. This will affect the analysis of beams and the invalidation of trajectory hypotheses in both their cursory and exhaustive forms. An additional complication arises due to the need to inform an agent about the number of balls available for constructing a particular partial hypothesis.

Analyse: Analysis now has the goal of establishing a beam's transit location when constructing an initial trajectory hypothesis or reconstructing an invalidated hypothesis for a multiquadrant beam. **Integrate** and **guess** are similarly affected, as both employ analyse.

Invalidate: Along with the local reasons for hypothesis invalidation (placing a ball in a square that was hypothesised empty, emptying a square that contained a hypothetical ball, and insufficient balls) there are external reasons: **(1)** a change in the transit location of a hypothetical trajectory; and **(2)** a change in the number of balls available for constructing a hypothesis. The source of the change and its credibility are considered when assigning priorities for reanalysis.

The above constitute the activities available to a DBB agent. Planning is decisive, as it determines which of the other activities the agent will perform. The agent's choice is guided by the previously stated principle: "*obtain the maximum amount of information from available sources before firing another shot*". The agent considers the needs of the other agents when making its decisions by "*providing its neighbours with any information that they can use to improve their solution, and its own solution in consequence*". In all cases, the choice is tempered by the modifiers stipulating that an agent should not attempt an exhaustive search unless **(1)** the search is highly constrained; **(2)** the search is unavoidable; or **(3)** <u>the agent has nothing better to do</u>. The last must be further qualified to ensure that an otherwise "idle" agent does not "disrupt" or "distract" the others. This leads to the following solution strategy described in six steps. The steps are not necessarily executed in sequence.

Step 1 - Analyse: An agent analyses as long as it has possible sources of information. These sources are: **(a)** The entry or exit location of a fired shot; **(b)** Changes to the grid; **(c)** An insufficient number of balls; **(d)** The transit location of a beam hypothesis. The last may be an initial location, an invalid location, or an alternative to a previous location.

Step 2 - Communicate changes: An agent communicates all relevant changes to affected neighbours whenever these occur. The relevant changes concern **(a)** The contents of overlapping regions; **(b)** The number of available balls; **(c)** Transit locations.

Step 3 - Receive changes: On receiving such information, an agent incorporates the changes into its reasoning. The contents of overlapping regions or **(b)**, and the number of available balls or **(c)**, are incorporated immediately. If "analysing", the agent should only consider a transit location or **(d)**, if it is analysing that particular hypothesis at the moment that the message arrives. In that case, it should abandon its search. If otherwise occupied, the agent will "plan", from where it will proceed to check invalidated hypotheses for reanalysis.

Step 4 - Invalidate beam hypotheses: Once an agent has completed its analysis, it checks to see what trajectory hypotheses have been invalidated by the changes produced locally or received from other agents. The invalidated hypotheses are assigned priorities for reanalysis as in Blackbox. Special consideration is given to hypotheses whose transit location is sent by an agent who has subsequently sent messages concerning the choice of shots. This indicates that the agent in question has no more sources of information, in which case it would be expedient to provide it with one.

Step 5 - Choose a shot: Once an agent has exhausted all possible sources of information, it chooses a shot with the intention of firing it. As with the single expert, the agent chooses the highest valued shot from the list of *evaluated shots* that it maintains for its quadrant. The choice involves performing a local evaluation of *possible shots* if local changes warrant it. The agent's decision to perform a global evaluation of multiquadrant shots is guided by similar consideration of changes.

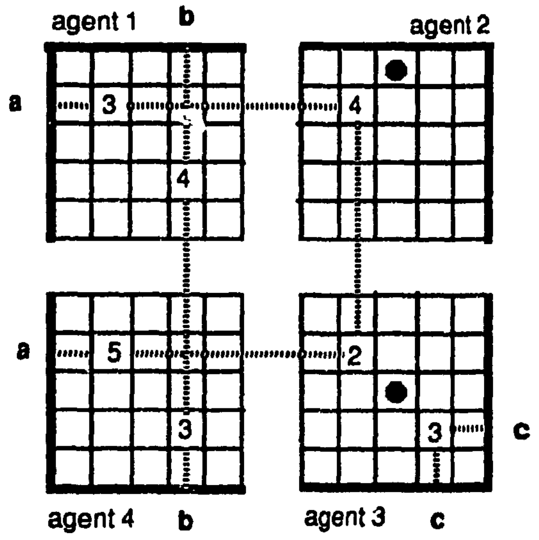
Step 6 - Negotiate a shot: The agent "negotiates" its choice of shot through the exchange protocol which it uses to determine the global value of a multiquadrant shot. Whenever an agent performs a local evaluation, it transmits any changes in value to the agents which are affected. When it has completed its local evaluation, it performs a global evaluation, depending upon the "relevant changes" it perceives. Relevant changes concern **(a)** the number of balls; **(b)** hypothetical transit locations that coincide with hypothetical shot trajectories through internal boundaries; and **(c)** changes to the global value of a shot transmitted by the other agents. The negotiation process for choosing a shot consists of the following steps:

1. An agent proposes a shot for global evaluation by communicating that shot's transit location to the agent whose quadrant the shot will enter. The agent cannot fire that shot until it receives a response.

2. The agent who receives the proposed shot determines the value of that shot within its own quadrant. The agent has the choice of either using an existing value, or reevaluating the shot. The latter may involve another exchange if the hypothetical trajectory continues into another quadrant.

3. The agent in whose quadrant the shot appears to *exit* returns the shot's cumulated value to its quadrant of origin. An exception occurs when the shot's hypothetical behaviour is not exit, and the final agent is the only one with firing access. Such a shot is *requested* by the agent who initiated evaluation.

The strength of the above strategy resides in the ability of any agent along the evaluation path to influence the time a shot is fired. At the same time, the proposing agent is not compelled to await the end of the exchange as it can propose alternative shots, or decide to fire a shot that remains within its quadrant, depending upon the responses it has received. Thus "busy" agents can postpone the firing of shots that pass through their quadrants while "idle" agents can negotiate a shot with each other, or fire a shot that only has local significance. Duplicated shots are minimised by enforcing a "clockwise" firing privilege whenever two agents evaluate the same shot. These ideas are illustrated in figure 5.4.2.



'a' is a multiquadrant shot which involves all four agents. Its cumulated value is 3 + 4 + 2 + 5 = 14. 'b' is a multiquadrant shot which involves only two agents, and it has a global value of 7. 'c' is not a multiquadrant shot, it has a value of 3, and it has only local significance. If agent 2 is "busy", it may not want shot 'a' to be fired. While waiting for agent 2 to evaluate 'a', agents 1 and 4 may evaluate and fire 'b', and agent 3 may decide to fire 'c'.

Figure 5.4.2: Local and negotiated shots

In all cases an individual agent chooses its activity based on its view of the problem solving state. This view is determined by the conditions which describe the agent's local state, as defined in *figure* 4.3.2, and the conditions which describe the agent's relation to the other agents in the DBB organisation. These conditions and their implications on an agent's reasoning are summarised in figure 5.4.3.

Conditions

C1. All agents have terminated.
C2. All balls have been located.
C3. The number of balls equals the number of unknown squares and all other agents have terminated.
C4. Only one ball remains to be found.
C5 Board changes have occured.
C6. Invalid transit locations received.
C7. Invalidated hypotheses are present.
C8. No more information can be derived from current sources.
C9. Shot values received.
C10. Shot evaluation request received.
C11. Shot evaluation requests sent.
C12. Remaining shots can provide no additional information.

Implications

C1. *Game over*: then the session controller assumes control.
C2. *Quadrant solved*: Then the agent broadcasts a *termination signal.*
C3. *Solve game*: then the agent solves its quadrant (C2).

C4. *Last stage of game*: then the agent employs exhaustive strategies, and serialised firing may be enforced.

C5. Local hypothesis invalidation possible. Local shot reevaluation necessary.
C6. Local hypothesis invalidation possible. Local shot reevaluation considered.
C7. Analysis
C8. Choose shot.
C9. Recompute global value of shots.
C10. Respond to shot evaluation requests.
C11. Consider responses when choosing an "optimal" shot.
C12. Integrate or guess.

Figure 5.4.3: Problem state conditions and implications

The agent represents its local Blackbox problem solving state using the quintuplet of objects defined in section 4.4, namely **S:<G,N,P,E,U>**. The agent represents its relation to the other agents in the organisation using **R:<T,Mi,Mo>**. **T** represents the termination state of the agents, and is used by an agent to determine when the game is over. **Mi** represents incoming messages, and **Mo** represents outgoing messages. **M** is represented using **M:<G',N',P',E',U'>**. The descriptions are given in figure 5.4.4.

Blackbox state: **S: <G, N, P, E, U>** where:

G: The grid corresponding to an agent's quadrant.
N: The number of actual balls available to the agent.
P: Possible shots (distinguished as *external side* and *internal boundary*).
E: Evaluated shots (*local value* and *global value*).
U: Unsolved beams (partial hypothesis within the agent's quadrant).

Organisational state: **R: <T, Mi, Mo>** where:

T: Termination state of the agents in the organisation.
Mi: Incoming messages.
Mo: Outgoing messages.

Message state: **M: <G', N', P', E', U'>** where:

G': The overlap area shared by adjacent agents.
N': The number of balls within and outside an agent's quadrant.
Mi<N'> decrements the *actual balls* within an agent's quadrant.
Mo<N'> decrements the *actual balls* available to the other agents.

P': Possible shots along an agents internal boundary.
E': Local and global values for **<P'>**.
Requests and responses to global evaluation attempts.

U': The transit locations of unsoved beams. These are further qualified as *initial*, *invalid*, and *alternative*.

Figure 5.4.4: Representation of the problem state in DBB

There are obvious interrelationships between S:<G,N,P,E,U> and R:<T,Mi,Mo>. The salient characteristic is that any changes to S are reflected in R, and vice-versa, wherever S and R coincide. Changes that the agent makes to S appear in its Mo, while changes communicated to the agent appear in its Mi. The interaction of the agents is controlled by specifying when Mo is actually communicated and when the receiving agent integrates the information contained in Mi into its reasoning.

5.5 Organisation of the DBB System

The Blackbox problem can be decomposed in a number of ways, and each decomposition could lead to a different type of problem solving organisation, based on reasoning strategies such as the *island driven* or *knowledge driven* approaches described in section 2.3. For simplicity, a geographic decomposition that remains stable throughout execution is chosen. This decomposition has the advantages that it is easy to describe, simple to administer, and straightforward to implement.

The geographic partitioning produces four agents assigned to four equivalent subproblems in geographically different locations. The agents are organised as a **team** in which each agent has an identical decision process but a distinct view of the problem. A team organisation is chosen because each agent is identical, and each agent must perform all activities involved in solving the problem. Such an agent can easily assume different organisational roles by having its "sphere of influence" modified. The team also provides a situation in which cooperative decision procedures can be studied. Furthermore, the team organisation appears the least biased in terms of how the solution progresses. Based on all these reasons, the team appears the most suitable organisation for developing the prototype.

Each agent maintains its own view of the global problem solving state derived from its local state and the information it sends and receives from the other agents. The agents individual views of the global state may be inconsistent. Although a consistent global view could be a critical requirement for solving certain distributed problems, it is not critical in DBB. Resolving inconsistency is an inherent part of intermediate solutions in DBB. Thus, the DBB agents do not have to take extra measures to maintain a consistent global view. Certain changes that an agent makes to its local state will affect the global state, and they must be communicated to the other agents. The agent must also assimilate changes to the global state communicated by the other agents in a reciprocal manner. The agent's role relates its local problem solving state to the global one. It determines what changes are relevant, how either state is affected, and what responses are to be conveyed over the network.

A design structure for realising such an agent is illustrated in figure 5.5.1. The proposed structure incorporates the organisational structure of figure 5.1.1. with the DBB knowledge requirements described in figure 5.4.1. The agent's general problem state is represented by its "Blackbox state" S:<G,N,P,E,U>, and its "organisational state" R:<T,Mi,Mo>, given in figure 5.4.4. The agent's role describes the agent's responsibilities and communication policies. The first relates the agent's organisational functions to the relevant problem states outlined in figure 5.4.3, thus describing the agent's expected reaction when specified conditions occur. Thc second describes how the agent communicates or "perceives" these conditions, specifying the relation between network messages, and Mi and Mo. All the above are embedded within the testbed component, which provides the user facilities described in figure 5.3.2. This constitutes the structure of an agent within the DBB system.

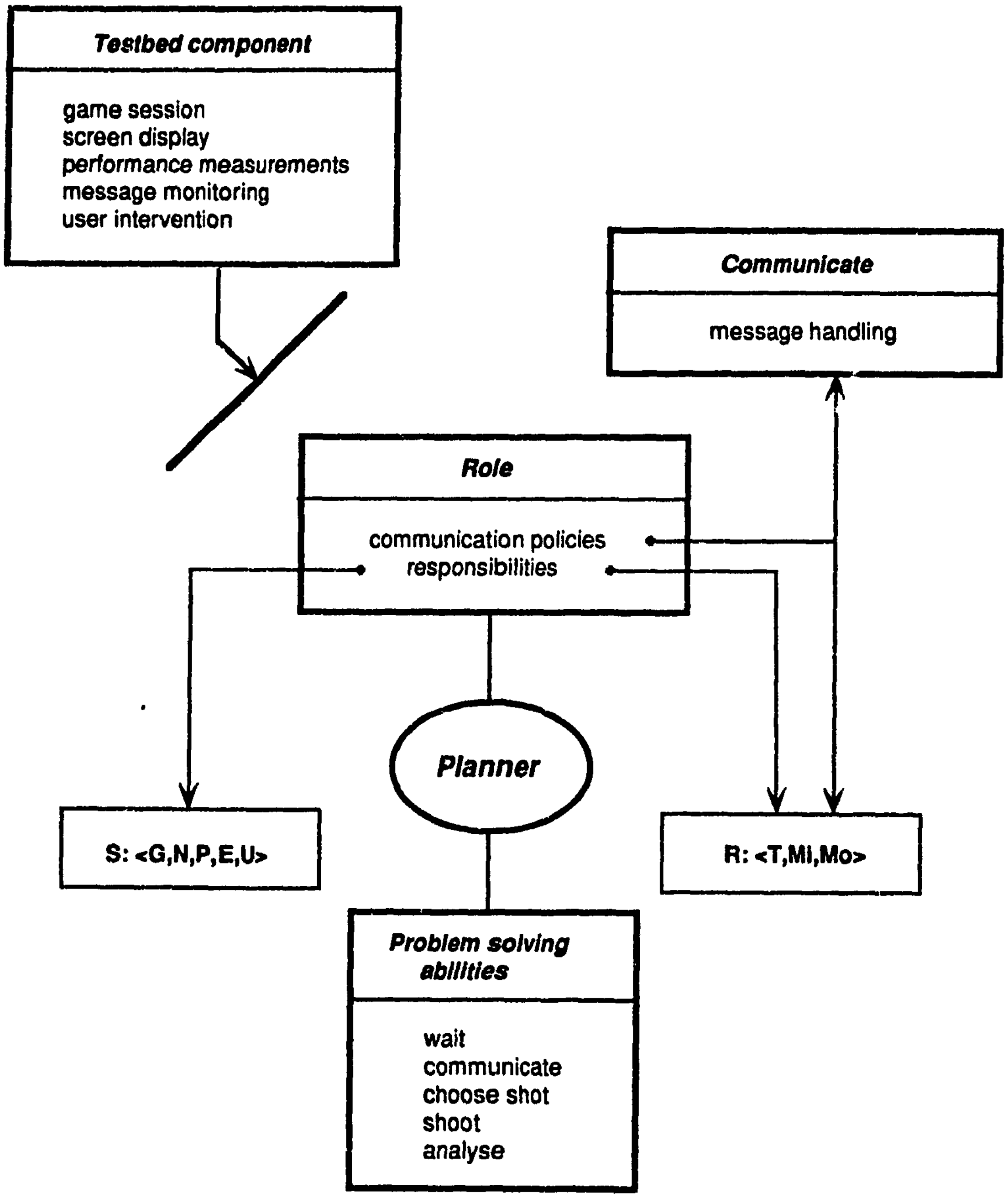


Figure 5.5.1: Structure of a DBB agent within the system

The following examples demonstrate the interrelationships within the structure of figure 5.5.1. Suppose an agent is in the process of performing analysis. In the course of its analysis, it modifies the contents of the overlap region that it shares with adjacent agents. This constitutes a locally produced change in S that coincides with R. Such changes are governed by the agent's responsibilities. Whenever such a change occurs in S it is moved to R. Furthermore, as a change occuring in the *central squares* concerns all agents rather than one in particular, the communication policy may establish that such a change be communicated with higher priority.

Meanwhile, the other agents are sending various messages resulting from their own activity. These messages are received by the message handler, which performs a "cursory analysis" to determine the message type, and place the messages into appropriate message buffers. How a particular message is handled will be determined by the communication policy, according to the agent's responsibilities. This in turn governs how the agent reacts to changes in the problem state. For the sake of our example, let's assume that the incoming message is a "shot value". As the agent is currently involved in analysis, it is not interested in evaluation, and will therefore ignore the incoming message. Thus the agent "shelves" the "shot value" until a later moment.

If, on the other hand the message were a "shot request", the agent might employ a totally different modus operandi, because shot requests indicate the sending agent's readiness to fire. The communication policy may establish that all consecutive shot requests should be counted. The agent's responsibilites may establish that when the agent has received e.c. three consecutive shot requests from one agent, it should respond. Thus, once three requests have arrived, they will be moved from the message buffer to R, where they are visible to the agent. The agent will then consider these requests even though it is in the process of analysing beams.

The DBB design structure proposed in figure 5.5.1, uses an "object oriented" approach to represent the knowledge for realising the interactions described in the preceding examples. A distinction is made between an agent's problem specific knowledge, and its organisational knowledge. The problem solving state associated with each is encapsulated within an "object", which maintains its representation between invocations, and defines the operations that transform the state, its constituent objects, and those to which it is related. The layering provided by an object oriented design makes the system easier to conceptualise, and will facilitate modifications in the course of the system's development.

5.6 Two Alternative Organisations for DBB

As stated earlier, one of the future goals of our research team is to experiment with alternative organisations. The team organisation described in the preceding sections provides a good starting point from which the functionality of the individual agents can diverge. This will occur naturally during system refinement, analogous to the way that a group of people redefine their roles in relation to each other as they become familiar with the way each operates. The organisations discussed in this section are offshoots of the team organisation. They offer a solution for choosing globally optimal shots at the cost of unbalancing the work load and introducing new interdependencies among the agents.

One alternative organisation considered is based on a simple hierarchy for making decisions related to the firing of shots (figure 3.6.1). Within the hierarchy, one agent is assigned the role of *director*, and has the responsibility of decreeing shots for its subordinates. This implies that the director must have the relevant information to choose "optimal" shots. Below, I describe two ways of realising the relation between a director and its subordinates, labeled hierarchy 1 and hierarchy 2. With the first, the director is responsible for evaluating the shots, and must therefore collect all the necessary information. With the second, the director only collects a subset of the information. In both cases the decision process is centralised. The strategies differ in the amount of information that the director must collect, and in the way the director determines which subordinate is ready to shoot.

In the case of hierarchy 1, the director is responsible for evaluating shots, and must therefore collect and maintain the necessary information. This consists of the contents of the *entire grid*, all *possible shots*, and *the number of actual balls*, corresponding to S:<G,N,P> in the problem state representation of all the agents. The director collects this information by having subordinates send all changes to the contents of their quadrants, to the number of balls that remain to be found, and to possible shots corresponding to the exit locations of fired beams. The director performs its evaluation <E>, and then chooses one or more "optimal" shots to send to the concerned agents for firing.

The flow of information described above can be realised by redefining the agents' responsibilities and communication policies. Then all subordinates will send relevant changes <N>, <P>, and <G> to the director. Also, subordinates are no longer responsible for choosing shots. Instead they must wait until the director sends them an evaluated shot <E> before they can fire. This hierarchical structure provides better control over the firing of shots. However, it also raises certain questions related to coordinating the director and its subordinates. When should a director evaluate shots? When should a director send a shot to an agent? Should the subordinate passively wait until the director sends it a shot, or should it indicate that it is ready to fire? The flow of information does not provide any incidental knowledge about the state of the agents.

With hierarchy 2, the director is only responsible for choosing "optimal" shots, thus its representation of the problem state is limited to <E> on the global board. Each subordinate is responsible for determining the local value <E> of possible shots <P> within its own quadrant <G>. It then sends all evaluated shots <E> to the director. The director combines the partial values <E> received from all its subordinates to determine the global value of shots <E>. It then chooses the "optimal" shot(s), and sends them to the subordinates. This exchange is realised by redefining the agents' reponsibilities and communication policies accordingly.

The director in hierarchy 2 retains control over firing while subordinates communicate much less information. The amount can be reduced even further by having subordinates only send changes in evaluated values. The exchange also carries implicit information, since the director is aware that a subordinate is searching for shots as soon as the director receives evaluations from it. However, the implicit information does not resolve coordination problems between the director and its subordinates. Should the director solicit evaluations from its subordinates, or should it choose shots with the possibly inaccurate values that it has at hand? And how do agents indicate that they are ready for a shot when they have no changed values to send?

Both hierarchy 1 and hierarchy 2 described above have an obvious disadvantage due to the division of tasks. The directors in both organisations must choose shots in addition to solving their own quadrants as their subordinates do. If we assume a more or less equal work load for each agent, at the moment that the director is ready to deal with its responsibilities i.e. choose shots, its subordinates will have also completed their work. Thus the subordinates must wait until their director has completed its evaluation before they can proceed. Hierarchy 2 will create less delays than hierarchy 1 due to the director's lighter load, but the director will still have more work than its subordinates. One way of overcoming this problem is to use a multiagent scheme (figure 2.2.3) whereby an agent is designated director according to its work load.

The above questions concerning how to coordinate the system only scratch the surface of the issues involved. New communication requirements will lead to new inter-dependencies that will introduce more timing problems in the agents' interactions. This is characteristic of the coordination problem in any distributed system, and particularly virulent in a DPS system where individual processing time is so highly unpredictable. This makes it difficult to foresee the full implications of any solution until it is in operation.

5.7 Evaluation of the DBB Organisation

Evaluation will play a primary role in DBB, both during its development, and in the subsequent comparisons that one wants to make between the different organisations. However, comparisons cannot be made until each organisation has been thoroughly verified. Unanticipated coordination problems could emerge when the problem solving expertise is in place and the effect of undesirable interactions appears in testing. Evaluation in DBB will involve several successive stages, with the success of each depending upon the thoroughness of the preceding one.

The first stage consists of verifying and validating the agents' Blackbox expertise. Although we have validated the expertise using the single expert described in chapter 4, it is necessary to ensure that no errors are introduced during its transition to the multiagent version. Ideally, the performance of an individual agent should be identical to that of the single expert when solving the same problem. Thus, the performance measures obtained by the single expert can be used to define limits for the solutions that can be expected from the group. The limits will serve to identify situations in which the group's performance differs significantly from that expected.

The next stage consists of verifying the knowledge pertaining to the agents' organisational roles. At this stage, the agents' responsibilities and communication policies will be observed and finetuned in an iterative fashion. This is where the limits described above will be applied, to identify the situations which require attention. The possible causes of inferior performance will fall into three categories: **(1)** errors due to bugs; **(2)** deficiencies in the knowledge related to the agents' responsibilities; and (3) deficiencies in the agents' communication policies. Identifying which is the cause may be difficult.

Once verification is complete, the organisation's performance can be evaluated. In this case, establishing that the organisation finds correct solutions is only one aspect, as we also wish to determine why a given organisation behaves in a certain way, and how its performance can be improved. This will require identifying the organisation's weak points, and finding alternative ways of resolving them. It is at this stage that we can examine whether the agents "cooperate", and how the organisation's structure can be modified to produce better cooperation. Any modifications will necessarily involve further verification, with the cycle being repeated until a satisfactory result is obtained.

Establishing suitable measures for the performance of an organisation is necessary in order to finetune a single organisation, and to compare the performance of different organisations. One measure is the "goodness" of the solutions obtained by the organisation. This could be based on measurements of the number of shots, the score, the number of errors in board hypotheses, and processing time, as with Blackbox. Since concurrent firing is possible, it will be necessary to establish the relative value of duplicated shots against that of additional processing time due to more controlled firing.

Another way of measuring an organisation's performance is to measure its rate of convergence onto the solution and the amount of communication involved. This will reveal more about how the organisation operates, and why it behaves in the way it does. The rate of convergence within the organisation will indicate how well the organisation serves to focus and coordinate the agents' individual activity. An organisation with a good focusing strategy will reduce the amount of "distraction" that occurs within the system. A well coordinated organisation is one in which the agents interact in a timely way, and thus avoid redundant work and delays. Since "distraction" results in unproductive processing, it can be measured by the amount of work that produces no changes in the solution, performed in response to messages. The "goodness" of coordination can be measured by the amount of time agents spend on finding identical solutions (i.e a ball location within a shared area), and the amount of time agents are "idle" while awaiting information.

As in any organisation, effective communication is crucial for optimal performance. The impact that communication has on the system will depend upon the granularity, frequency, and amount of information communicated, and the characteristics of the communications medium. Since all the organisations will operate with the same medium, the constraints it imposes are considered part of the problem. How effectively the agents communicate can be measured by (1) the number of messages exchanged; (2) the amount of information that the messages convey; and (3) all delays related to exchanging the information. These will reveal the "bottlenecks" and other sources of inefficiency that make one organisation more suitable than another in a particular situation.

Evaluating an organisation of expert systems involves considerably more work than the evaluation of each expert system within that organisation. Only after the constituent experts have been thoroughly verified and validated on an individual basis can verification of the organisation begin.

This must simultaneously establish both the validity of the agents' interactions, and the impact of these interactions on the organisation's performance. Finetuning an organisation to its attainable optimum is necessary, as it would otherwise be unfair to make comparisons between the organisations.

Bibliography

[1] F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, "Building Expert Systems", Addison-Wesley, 1983.

[2] E. Rich, "Artificial Intelligence", McGraw-Hill, 1983.

[3] E. Cassirer, "The Problem of Knowledge - Philosophy, Science, and History since Hegel", Yale University Press, 1950.

[4] D.S. Levine, "Survival of the Synapses", *The Sciences*, New York, Nov/Dec 1988, pp. 46-53.

[5] M.A. Fischler and O. Firschein, "Intelligence: The Eye, the Brain, and the Computer", Addison-Wesley, 1987.

[6] J.R. Searle, "Is the Brain's Mind a Computer Program?", *Scientific American*, Vol. 262, January 1990, pp. 26-31.

[7] H.C. von Baeyer, "The Aesthetic Equation", *The Sciences*, Jan/Feb 1990, pp. 2-5.

[8] C. de Koven, "Survey of Automated Knowledge Acquisition", Technical report, Concordia University, Montreal, fall 1988.

[9] D. O'Leary and R. O'Keefe, "Verifying and Validating Expert Systems", Tutorial program, *11th International Joint Conference on Artificial Intelligence*, Aug. 1989.

[10] H.P. Nii, "Blackboard Systems - Part One: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures", pp. 38-53. "Blackboard

Systems - Part Two: Blackboard Application Systems and a Knowledge Engineering Perspective", pp. 82-107, *The AI Magazine*, Summer 1986.

[11] M.S. Fox, "An Organizational View of Distributed Systems", *IEEE Transactions on Systems, Man, and Cyber'etics*, Vol. smc-11, No. 1, January 1981.

[12] M. Minsky, "The Society of Mind", Simon and Schuster, New York NY, 1986.

[13] R.M. Steers, "Introduction to Organizational Behavior", pp. 242, Scott, Foresman and Company, 1981.

[14] K. Davis, "Evolving Models of Organizational Behavior", pp. 4-14 in *Organizational Behaviour*, J.W. Newstrom and K. Davis, McGraw-Hill, 8th ed. 1989.

[15] N. Nyiri and J. Redekop eds. "Uses and Abuses of Systems Theory, Perspectives", Interdisciplinary Research Seminar, Wilfrid Laurier University, 1985.

[16] V. Sathe, "Culture and Related Corporate Realities", Richard D. Irwin Inc., Homewood Ill. 1985.

[17] E.H. Durfee, "Coordination of Distributed Problem Solvers", Kluwer Academic Publishers, Boston 1988.

[18] M.W. Alford, J.P. Ansart, G. Hommel, L. Lamport, B. Liskov, G.P. Mullery, and F.B. Schneider, "Distributed Systems - Methods and Tools for Specification - An Advanced Course", *Lecture Notes in Computer Science*, No. 190, Springer-Verlag, 1985.

[19] W. Chu, L. Holloway, M. Lan, and K. Efe, "Task Allocation in Distributed Data Processing", *Computer*, Vol. 13 pp.57-69, November 1980, reprinted in *Distributed Computing: Concepts and Implementations*, P. McEntire, J. O'Reilly, and R. Larson, IEEE Press, New York, 1984.

[20] E.H. Durfee, V.R. Lesser, and D.D. Corkill, "Trends in Cooperative Distributed Problem Solving", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 1, No. 1, March 1989.

[21] V.R. Lesser and D. Corkill, "DVMT: a Tool for Investigation of Distributed Problem Solving Networks", *Blackboard Systems*, R. Englemore and T. Morgan eds., Addisson-Wesley, 1988, pp. 355-386.

[22] R. Campbell, "Background for the Uninitiated", in *Paradoxes of Rationality and Cooperation*, R. Campbell and L. Sowden eds, The University of British Columbia Press, Vancouver 1985.

[23] M.L. Ginsberg, "Decision Procedures", in *Distributed Artificial Intelligence*, M. Huhns ed, San Mateo, CA: Morgan Kaufmann, 1987, pp. 3-28.

[24] J. Halpern and M. Yoram, "Knowledge and Common Knowledge in a Distributed Environment", *Proceedings of the 3rd ACM Symposium on the Principles of Distributed Computing*, 1984, pp. 50-61.

[25] Y. Moses, D. Dolev, and J. Halpern, "Cheating Husbands & Other Stories: A Case Study of Knowledge, Action, and Communication", *Proceedings of the 4th Annual ACM Symposium on the Principles of Distributed Computing*, August 1985, pp. 215-223.

[26] M. Benda, V. Jagannathan, and R. Dodhiawala, "On Optimal Cooperation of Knowledge Sources - An Empirical Investigation", technical report, Boeing Advanced Technology Center, Boeing Computer Services, Seattle, WA, July 31 1986.

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

[28] S. Cammarata, D. McArthur, and R. Steeb, "Strategies of Cooperation in Distributed Problem Solving", *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, Vol. 2, August 1983, West Germany.

[29] D. Corkill, V.R. Lesser, and E. Hudlicka, "Unifying Data-Directed and Goal-Directed Control", *Proceedings of the National Conference on Artificial Intelligence* (AAAI), 1982, pp. 143-147.

[30] V. Lesser, D. Corkill, J. Pavlin, L. Lefkowitz, E. Hudlicka, R. Brooks, and S. Reed, "A High-Level Simulation Testbed for Cooperative Distributed Problem Solving", *Proceedings of the 3rd International Conference on Distributed Computer Systems*, 1982, pp. 341-349.

[31] D. Corkill and V. Lesser, "The Use of Meta-Level Control for Coordination in a Distributed Problem Solving Network", *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, 1983, pp. 749-756.

[32] E. Durfee, V. Lesser, and D. Corkill, "Increasing Coherence in a Distributed Problem Solving Network", *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, 1985, pp. 1025-1030.

[33] E. Durfee and V. Lesser, "Using Partial Global Plans to Coordinate Distributed Problem Solvers", *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, 1987 pp. 875-883.

[34] C. De Koven, T. Wieland, and K. Pitula, "A Distributed Version of the Game of Blackbox", Project Report, Concordia University, February 1989.

[35] C. De Koven and T. Radhakrishnan, "An Experiment in Distributed Group Problem Solving", to be presented at the IFIP Conference on Multi-user Interfaces and Applications in Greece, Sept. 24-26, 1990.

[36] K. Pitula, T. Rhadhakrishnan, and C. Grossner, "Distributed Blackbox: A Testbed for Distributed Problem Solving", *Proceedings of the Ninth Annual Conference on Computers and Communications*, Phoenix 1990, pg. 741-748.