The Institute for the
Learning Sciences
Northwestern University

OPPORTUNITIES AND PLANNING IN AN UNPREDICTABLE WORLD

Louise Margaret Pryor

Technical Report #53 • June 1994

Established in 1989 with the support of Andersen Consulting

NORTHWESTERN UNIVERSITY

Opportunities and Planning in an Unpredictable World

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Louise Margaret Pryor

EVANSTON, ILLINOIS

June 1994

ABSTRACT

Opportunities and Planning in an Unpredictable World

Louise Margaret Pryor

An agent operating in an unpredictable world cannot simply construct plans ahead of time and expect them to work perfectly. Instead, it must adapt its plans to the circumstances that it actually encounters. To do this, it must acquire information about the actual state of the world, and recognize when unpredicted situations affect its goals. Both problems are addressed in this thesis.

The requirement for information arises out of the need to make decisions, and is thus a goal-directed activity. Plans to achieve information goals can be constructed in the same way as plans for other goals. This thesis describes Cassandra, a contingency planner whose plans include provisions for explicit decisions and the resulting information acquisition; and PARETO, a plan execution system whose plans also make explicit provision for information acquisition arising out of the need to make decisions.

Plan adaptation during execution requires a unifying framework within which the plan construction and plan execution processes can be integrated. In this thesis the argument is made that the consideration of opportunities provides such a framework, and a mechanism is presented that enables an agent to recognize unexpected opportunities on the fly and respond to them appropriately and in a timely manner. Its implementation in PARETO is described.

This mechanism is based on *reference features,* features that are both cheap and functional. These features appear to be prevalent in everyday life. Reference features form the basis of a powerful focusing mechanism that can help an agent balance deliberation and action by indicating both when deliberation would be productive and what the deliberation should be about. Their use by PARETO in recognizing and reasoning about opportunities is described.

ACKNOWLEDGMENTS

My principal thanks are due to my advisor, Gregg Collins. Without his help and encouragement I would never have got this far. He has always been ready to listen to my wilder ideas and to encourage the saner ones. The focus provided by his comments has been invaluable, and his insistence on clear, coherent explanations has improved my writing immeasurably.

I have benefited immensely from the community at Northwestern University's Institute for the Learning Sciences. The Institute was founded and is directed by Roger Schank, to whom I owe a great debt. He and Larry Birnbaum taught my first AI courses, and encouraged my early enthusiasm. Larry has always been ready for a good argument and his questions have consistently made me think. Ken Forbus and Paul Cooper, members of my committee, have both helped me clarify my ideas on many issues. Chris Riesbeck made useful comments on earlier drafts of this thesis. Other faculty at the Institute have helped to foster an atmosphere in which lively discussions have flourished.

I have had many useful discussions with other students, too many to mention here. However, I owe especial thanks to Matt Brand, Will Fitzgerald, Bruce Krulwich, and Eric Jones. In addition, Alex Kass provided early encouragement, and Chip Cleary made useful comments on parts of this dissertation.

Jim Firby's encouragement and practical assistance with the RAPs system have been invaluable, and his criticisms have always been constructive. Thanks, too, to Dan Weld and Tony Barrett for the UCPOP code.

The friendship of Matt Brand, Danny Choy Edelson, Vivian Choy Edelson, Jim Firby, Joanne Firby, Will Fitzgerald, Steve Hanks, Jeannette Hermann, Ann Holum, Liz Jessup, Alex Kass, Chris Owens, and Mark Wukas helped make my years in the USA such a pleasurable experience. Those based in Chicago were especially generous with their friendship and hospitality during the process of moving back to the UK. Many people in Birmingham, especially Russell Beale and David Pym, have provided valuable moral support during the last few months. Back in Chicago, Will Fitzgerald's help with the logistics of transatlantic printing has been vital.

Without the support and encouragement of family and friends back in the UK the whole thing would have been impossible; especial thanks are due to Caroline Baker, Vicky Clough, Denis Clough, Gerry Fitzsimons, Peter Pryor, Pippa Rogerson, and, above all, to my parents.

Finally, I would like to express the thanks that are due to my uncle, Duncan Smith, who first encouraged my intellectual curiosity and who never accepted my answers as they stood. His example helped me decide to be a student again; he saw the beginning of the road that led to this dissertation and I wish he could have seen the end.

CHAPTER 1

INTRODUCTION

1.1 Introduction

The world is unpredictable—it is impossible to tell in advance exactly what its state will be at any future time. An autonomous agent must be able to cope with this unpredictability. For example, a robot roaming the surface of a strange planet will have to be able to decide for itself where it should take soil samples, what areas it should explore and what routes to take. The samples that should be taken may depend *inter alia* on the terrain that it encounters, atmospheric conditions, and the results of tests performed on earlier samples, none of which can be predicted in sufficient detail to allow hard and fast decisions to be made in advance. Similarly, an intelligent program managing an investment portfolio must be able to take into account the relative price movements of a large number of stocks and shares, as well as overall market conditions and the requirements of the investor. Again, prediction in advance is impossible—the program must decide what to do on the fly.

In this dissertation I describe my investigation of the problem of how an intelligent agent can achieve its goals in an unpredictable world. In the course of this investigation I built two systems:

- Cassandra, a system that constructs contingency plans
- PARETO, a system that notices and responds to opportunities on the fly as it pursues its goals.

I shall briefly describe each of these in turn.

Cassandra[1] extends traditional AI planning techniques by allowing for alternative contingencies in its plans. For example, suppose you don't know whether your car will start tomorrow morning, but you know that you must go to work in any case. Cassandra would construct a *contingency* plan for use in this situation, along the lines of the following:

1. Try starting the car.
2. See whether the engine is running.
3. Decide what to do based on whether the engine is running
4. Either drive the car to work
   Or take the bus.

Cassandra, which is described in detail in chapter 2, uses a simple extension of the standard STRIPS action representation. It is the only planner of its type to represent decisions explicitly in its plans, allowing Cassandra's plans to include steps to acquire all the information needed to decide what to do in the various contingencies. In spite of the advantages that Cassandra has over other planners, its scope is limited in that the type of planning that it performs is only viable for small, well-defined problems with

[1] Cassandra was a Trojan prophet who was fated not to be believed when she accurately predicted future disasters.

few sources of unpredictability. An alternative approach is therefore required if agents are to operate successfully in unpredictable environments.

The major part of the work described in this dissertation is therefore concerned with PARETO,[2] a system that abandons the approach taken in Cassandra of trying to allow for unpredictability in advance. Instead, the emphasis in the design of PARETO is on recognizing the need for and making decisions on the fly. As an example of the kind of reasoning PARETO is meant to perform, suppose you happen to see a sharp knife as you are looking for some scissors with which to cut some string. You would probably realize that you could in fact use the knife to achieve your goal, and would abandon your plan to find some scissors. PARETO recognizes and takes advantage of such opportunities. It does so by using a set of cheap and functional features called *reference features*.

I argue in this dissertation that the class of opportunities that can be recognized by using reference features covers many of the routine opportunities that arise in the normal operation of an agent. The development of a set of reference features for the domain in which it operates will thus enable an agent to operate effectively in most circumstances. PARETO operates in a simulated world, and its design demonstrates that the technique of using reference features to recognize opportunities can be implemented, but does not address the issue of whether the technique will scale up successfully to real domains.

In later chapters I shall describe the design and operation of Cassandra and PARETO. In the remainder of this chapter I present the background that led to their development. Section 1.2 discusses the problems of using plans in an unpredictable world, highlighting two vital issues: the making of decisions and the acquisition of information. These two issues are then considered in more detail in sections 1.3 and 1.4 respectively. Section 1.5 gives an overview of the remainder of the dissertation.

1.2 Plans for an unpredictable world

An agent needs all sorts of information about the world in which it operates. It needs information to perform mundane actions and to execute complex plans. For example, to make a bowl of cereal in the morning you need to know whether the cereal packet is in the cupboard or on the counter top; whether there is a bowl in the draining rack or whether to take one out of the cupboard; where to find the milk in the fridge; when you pour the cereal and milk into the bowl you must watch the level to stop it overflowing. Similarly, to travel from Chicago to London you need to know where the airport bus stops; what it looks like; when it arrives at the airport; which terminal your flight leaves from; which desk to check in at; which departure gate to go to; where your seat is; and so on.

The traditional approach to planning in AI fails to account adequately for these information requirements, because it assumes that all necessary information is available at the time of choosing a plan. The essence of unpredictability is that some information is not available in advance, so traditional planning systems are ineffective in an unpredictable world and an alternative approach must be taken. There are two possibilities: the traditional approach can be extended, or it can be abandoned. The design of Cassandra extends the traditional approach by allowing for uncertainty in the construction of detailed

[2] Planning and Acting in Realistic Environments by Thinking about Opportunities. Vilfredo Pareto (1848-1923) was an Italian economist, sociologist, and philosopher best known for the notion of *Pareto optimality* and for the *Pareto distribution*, neither of which is used in the work described here.

plans. PARETO abandons the traditional approach by forsaking the emphasis on prediction that is embodied in that approach and instead concentrating on the issues involved in the handling of unforeseen circumstances. This section explains how the problems that lead to the need for these alternative approaches arise.

No action that an agent intends to perform can be completely specified without knowledge of the state of the world in which it will be carried out. How far you stretch your arm to reach the cereal packet depends on where the packet is; the steps you take between the check-in desk and the departure gate depend on the layout of the airport and the whereabouts of the other people moving around it. The effects of actions depend on the circumstances in which they are performed. For example, the amount of cereal in the packet determines how much cereal comes out when you tip it; whether a vehicle that you flag down is a taxi or a private car determines whether it will take you to the airport; whether the immigration official lets you into the country depends on whether the document you present is a valid passport.

An agent that uses a plan to achieve a goal is deciding in advance which of its goals it should pursue, what actions will achieve those goals, and when the actions should be performed. It is effectively deciding on a course of action that will achieve the chosen goal (Collins 1987). The agent makes these decisions on the basis of information that it has at the time that it chooses a plan. For example, you might decide to make a bowl of cereal on the basis of a number of beliefs, including the belief that you have a bowl, the belief that there is cereal in a packet in the cupboard, and the belief that there is milk in the fridge. To make a plan, an agent must therefore have the necessary information on which to base these decisions. You cannot plan to make a bowl of cereal if you have no idea whether you have a bowl or not, or if you are totally ignorant of the location of any milk. You could, of course, take an incremental approach: find out whether you have a bowl, and if you do take it to the table, find out whether you have cereal, and if so pour some into the bowl, and so on. In this case you cannot plan ahead to make the bowl of cereal because you lack the necessary information, but you can acquire the necessary information as you perform the necessary actions. The important point is that when planning for a goal you must use the information available at the time of planning.

Traditional AI planning systems (Fikes and Nilsson 1971; Sacerdoti 1977; Chapman 1987) have effectively decoupled plan construction and plan execution. They operate on the assumption that all needed information is freely available with total accuracy. If this were the case, the course of plan execution could be predicted in minute detail, and there could be no surprises. This assumption thus licenses the production of plans that fully specify every action that is to be performed in order to achieve the given goal. For example, the plan to make a bowl of cereal would specify the angle at which the cereal packet should be held while pouring cereal into the bowl, and how long the pouring action should last.

These systems, known as *classical* planners (Wilkins 1988), rely on three assumptions about the world:

- *Simplicity*: it is possible to know everything about the world that might affect the agent's actions.
- *Stasis*: there will be no changes in the world except those caused by the agent's actions.
- *Certainty*: the agent's actions have deterministic results.

There are very few environments in which these assumptions are valid. In the real world you cannot know the precise location of the check-in gate, or exactly how much cereal there will be in the packet before you start pouring. Your room-mate also uses milk, and might put it back on a different shelf in the fridge from the one you left it on; you can't predict the exact movements of everyone else who is moving through the airport at the same time as you, or the effect of a sudden thunderstorm on the departure time of your flight. Finally, you cannot even guarantee the results of your own actions. You can't control the angle at which you hold the cereal packet sufficiently accurately to know how much cereal will emerge; your case may slip out of your grasp as you maneuver it onto the scales at the check-in desk. In short, the real world is unpredictable.

In general, unpredictable environments have the following characteristics:

- *Complex*: it is impossible to know everything.
- *Dynamic*: changes occur as a result of the actions of other agents or of natural phenomena.
- *Uncertain*: the agent cannot be sure what the results of its actions will be.

The classical planning paradigm is inadequate in an unpredictable world. An agent cannot decide in advance exactly how it will execute all the actions necessary to achieve a goal. There are three key effects of an unpredictable world that together lead to the breakdown of the classical approach.

First, the decisions embodied in the choice of a plan may turn out to be wrong, because of the inaccuracy of the information on which they are based. For example, your roommate may have finished the milk unbeknownst to you; your beliefs about the presence of milk in the fridge are therefore wrong, and you will not be able to perform all the actions required to make a bowl of cereal. This means that an agent cannot blindly follow preordained plans come hell or high water; instead, it must monitor its plan execution so that it recognizes when it is running into problems.

Second, it may be impossible to make all the necessary decisions in advance. The information needed to make some of these decisions may simply not be available at the time a plan is chosen. For example, you cannot predict the movements of all the other people who will be in the airport at the same time as you, so you cannot decide in advance on the exact steps you will take on your route from the check-in desk to the departure gate. When constructing its plans, the agent must therefore be able to recognize those decisions that must be deferred owing to the lack of information. When executing its plans, it must be able to make those decisions on the basis of the information that it has acquired since the planning process took place. In effect, the agent must interleave plan construction and plan execution.

Third, decisions may arise that have not been foreseen in the planning process. For example, the telephone might ring while you are preparing your cereal, forcing you to decide whether to answer it or whether to ignore it and continue with your breakfast. Similarly, the flight to London may be overbooked, giving you an opportunity to volunteer for compensation in return for delaying your trip. The agent must recognize those circumstances that lead to need to make unforeseen decisions during plan execution and must, if necessary, be able to acquire the information that will support them. The agent must be able to change its plans during their execution to reflect unforeseen situations.

The design of an agent that is to operate in an unpredictable world cannot therefore rely on being able to separate plan construction and plan execution, but must address the issues of interleaving the two. It must be able to:

- handle plan failure;
- make deferred decisions;
- make unforeseen decisions; and
- acquire the information needed to make the necessary decisions.

My work has concentrated on the issues of decision-making and information acquisition. The design of Cassandra addresses the issues of deferred decisions and their information requirements, while that of

PARETO addresses the issues connected with unforeseen decisions. In the next section deferred and unforeseen decisions are considered in more detail, together with their information needs, and the methods of acquiring information that may be used to fulfill those needs are discussed in section 1.4.

1.3 Decisions in plan execution

The key to handling both deferred and unforeseen decisions is the acquisition of information about the current state of the world. Deferred decisions are those that cannot be made in advance because the necessary information is not available or too difficult to obtain; to make these decisions as it executes its plans, the agent must acquire the necessary information. The need to make unforeseen decisions arises out of the occurrence of unexpected situations; the agent must be able to recognize such circumstances, and needs information about the current state of the world to do so. In this section I will first consider these two types of decisions separately, and then consider their similarities and differences.

1.3.1 Deferred decisions

As we have seen, a lack of information may make it impossible to make a particular decision in advance. When this happens, the agent cannot fully specify the plan that is to be followed until the necessary information is available. Returning to the example of flying from Chicago to London, it is clearly far beyond the realm of possibility to predict the exact details of every action that must be taken. Imagine, for example, trying to decide ahead of time exactly where you should place your carry-on baggage in the overhead bin, or precisely when you should move to let your neighbor get out of his seat. Moreover, it is often impossible to predict even at a broad level what actions should be performed. For instance, at Gatwick airport there are two possible ways of getting from your plane to immigration. If your plane lands at the satellite terminal, you have to take the monorail to the main terminal. If, on the other hand, your plane lands at the main terminal, you have the choice of walking along a corridor or taking the moving walkway along the same corridor. You cannot tell until the plane lands whether you will be taking the monorail, moving walkway, or neither.

The unpredictability of the world virtually guarantees that every plan will be incomplete because of the inability to decide in advance exactly how to perform actions or what actions to perform. In both cases, the information that is required to decide on an appropriate course of action is often not available until the time at which the actions must be performed. The planner cannot tell in advance what the future state of the world will be. This section discusses the issues involved in addressing this problem by deferring these decisions until the plan is executed and the information is available. This is the approach taken by Cassandra, which instead of predicting the actual future state of the world, predicts what the possible future states are. Cassandra's plans direct the agent executing them to acquire the information that is needed to determine what the actual state is, so that the decision as to the appropriate course of action can be made.

Cassandra thus recognizes those decisions that cannot be made in advance because they depend on facts about the world that cannot be predicted, and defers them until the necessary information is available. In general, any agent operating in an unpredictable world must be able to recognize those decisions that should be deferred due to missing information. There are three ways in which an agent may lack the information required to make a decision in advance:

- The world's *complexity* means that the agent may completely lack information about certain aspects of the world.
- The world's *dynamism* means that many aspects of the world will change between the time of the plan's construction and its execution.
- The world's *uncertainty* means that the agent cannot rely on its own actions having certain results.

An agent constructing plans for later use must be able to recognize when information on which a decision relies is unreliable because it is subject to one of these three sources of unpredictability. In a sense, all information is unreliable to some degree—there is no guarantee that any predictions that an agent makes will be borne out in practice. However, the world is by no means totally unpredictable: some aspects are more stable than others, and some predictions can be relied on. Indeed, no planning would be possible if this were not so. For example, you can usually rely on the airline not changing the scheduled time of the flight, even though it has been known to happen. More generally, you tend to rely on airports not packing up and moving, on the laws of physics continuing to allow for the possibility of flying machines, and so on.

Constructing plans in an uncertain world therefore involves determining which decisions can be made in advance and which must be deferred owing to the unreliability of the information on which they are based. This determination is outside the scope of the work presented here; recent work on this problem includes the approximate probabilistic projections of Hanks (1990) and the qualitative approach of Wellman (1990).

When an agent constructing a plan recognizes that it must defer a decision, there are several courses open to it:

- It can simply note that it cannot make the decision now, but that it will have to be made when the plan is executed.
- It can predict what information will be required to make the decision, and include actions in the plan to ensure that the agent executing the plan will have acquired the necessary information by the time it has to make the decision.
- It can predict what the decision should be in various contingencies, include actions in the plan to determine which contingency actually obtains, and then include actions in the plan to be performed in each contingency.

In each of these three cases the agent executing the plan must make the deferred decision at execution time, and must acquire the information that it will need in order to make it. In the first case, the plan itself will provide no guidance as to what information is required or how the decision should be made. In the second case, the plan will tell the execution agent what information it will require, and in the third case the plan itself specifies how the decision should be made.

Cassandra follows the third of these approaches, and includes explicit *decision actions* in the plans that it constructs. In effect, Cassandra decides in advance to make a decision in the future, in the same way as it decides in advance to perform any other action in the future. The decision actions in Cassandra's plans may have preconditions involving the acquisition of information, and further actions are included in the plan that achieve these preconditions. Just as, for example, you must be at the departure gate in order to board the airplane, you must know which gate the plane is due to depart from in order to decide how to get there. The acquisition of the necessary information arises directly out of the necessity of making a decision. As this decision is an action in Cassandra's plans, information acquisition in support of deferred decisions can thus be planned in advance. An agent executing the plans constructed by Cassandra is thus directed to acquire the necessary information during plan execution, and then how to use it to make the appropriate deferred decisions. Cassandra is apparently the first contingency planner to include explicit decision actions in its plans.

Rather surprisingly, remarkably little attention has been paid to the problem of constructing plans in unpredictable worlds. One argument has been that the world is so unpredictable that it is pointless to use any plans at all (Brooks 1986; Agre and Chapman 1987). This argument is clearly contradicted by the fact that people make extensive use of plans in their everyday lives. Although the world is unpredictable, it is not overwhelmingly so; although we cannot possibly predict every aspect of future situations, we can predict enough aspects in sufficient detail to be able to decide in advance on broad courses of action. You may not be able to plan exactly what motions you will make in preparing a bowl of cereal, but you can certainly plan to prepare and eat it as part of your breakfast.

McCarthy and Hayes (1969) realized that it may be impossible to plan ahead in full detail for some actions because of unavailable information: they give the example of needing to know the combination to open a safe, and argue that this *knowledge precondition* should be planned for in the same way as other action preconditions. In the approach described in this section, the information is actually required to decide which particular sequence of actions will open the safe. The problem addressed by McCarthy and Hayes is thus a special case of the more general problem of deferred decisions. An important contribution of their paper was to point out the need to plan for the acquisition of information. Since then, this view has been generally accepted in principal, while being on the whole ignored in practice. Although the principle of planning to acquire information to achieve knowledge preconditions is thus uncontroversial, insufficient attention has been paid to the methods of achievement, as section 1.4 describes.

The problem of planning for contingencies was addressed in the seventies by Warren (1976), but after that was neglected until recently (Etzioni et al. 1992; Peot and Smith 1992). None of these approaches consider the problem in the context of making deferred decisions, and none of them give a satisfactory account of the place of information acquisition in the execution of the plans they produce.

Much of the information that an agent acquires during plan execution will be acquired in response to the need to make decisions that could not be made when the plan was chosen. As we have seen, this information acquisition can be planned in advance, just as any other action may be planned. This means that the planner can decide ahead of time exactly how to acquire the required information, by choosing a method from among those available. For example, one way of finding out which gate the flight to London departs from is to walk through the airport until you find it; another is to look at the screens listing flight departures; and a third is to ask the attendant at the check-in desk. In general, planning to acquire information is as complex as planning for any other task, and is an important issue that must be addressed in the consideration of deferred decisions in unpredictable worlds. The issue of the planned acquisition of information is discussed in detail in section 1.4.

1.3.2 Unforeseen decisions

Decisions can be deferred when the planner is able to predict that these decisions will arise. In general, however, it is impractical to predict all the decisions that may have to be made, and impossible to plan in advance for all of them. Situations can thus occur that lead to the need to make unforeseen decisions. For example, a friend might telephone to offer you tickets for a concert this evening, forcing you to decide whether to abandon your plans to go to a film. Similarly, as you are on your way to a favorite restaurant for dinner, you might notice that a new restaurant has opened, allowing you the possibility of trying a different cuisine. Agents must therefore keep track of their environment so that they can recognize when the need arises for decisions that have not been foreseen in their plans. PARETO is a plan execution system that recognizes and makes unforeseen decisions. It acquires information so that it can recognize the need for these decisions, and acquires further information in order to make them. In contrast to traditional planning systems, which emphasize prediction to the exclusion of responding to the unexpected, the design of PARETO concentrates on the need to function effectively in an unpredictable world. PARETO does this through its mechanism for recognizing and responding to opportunities.

The need for a decision arises when there is an aspect of the unexpected situation that affects the agent's goals. Recognizing when the need for a decision arises thus means being able to distinguish those unexpected situations that affect its goals from those that do not. There are two general effects that unexpected situations may have on the agent's goals:

- *Opportunities*: a goal may become easy to achieve.
- *Threats*: a goal may become difficult to achieve.

If the ease of achievement of a goal is not affected by the unexpected situation,[3] the agent need make no decision about changing its plans to reflect the new circumstances. In other words, recognizing the need for decisions comes down to recognizing opportunities and threats. If the agent is to operate effectively in a dynamic world, this recognition must be performed quickly, before the moment for decision has passed. An agent with limited resources must thus be able to recognize opportunities and threats with minimal cognitive effort.

In an unpredictable world an agent will continually encounter unforeseen situations, and its environment will change constantly. Few of these changes will affect the agent's goals in any way. For example, the cars on the road outside your apartment window may move as you make your cereal, but they have no effect on how you eat your breakfast. You can ignore people walking on the other side of the road as you walk to the bus stop in the morning. Unless you want to buy something to eat, you can ignore the fast food outlets you pass, and you need only notice newsstands if you need to buy a newspaper. There is thus a vast amount of information available to an agent, little of which affects its goals in any way.

Agents operating in an unpredictable world face the problem of using minimal cognitive effort to pick out the few aspects of their changing circumstances that are relevant to their goals. This task is by no means trivial. PARETO's approach is to recognize opportunities by using a filtering process based on *reference features*. Reference features are cheap for the agent to infer and represent functional tendencies of situation elements. A filtering process based on reference features can thus be both cheap to apply and highly predictive.

The use of reference features is based on the observation that the world, although unpredictable, has many regularities. Objects tend to behave in certain fixed ways, external events occur in certain patterns, the agent's own goals tend to arise in particular circumstances. In general, objects (and other situation

[3] A goal may also become irrelevant, either because a subsuming goal has been achieved (a special case of an opportunity) or because it has become impossible to achieve (a special case of a threat).

elements) tend to have consistent causal effects. For example, knives and scissors tend to cut things, wet paint tends to leave marks on things, fires tend to heat things. Over time, the functional tendencies of objects become apparent. This is obvious: we rarely have to worry about whether a chair can be used for sitting, or whether a knife can be used for cutting. Chairs *do* support things, and knives *do* cut things—in fact, they are made for these purposes.

These functional tendencies can be used to tag objects that are frequently involved in goal achievement. For instance, if some objects frequently cut other objects, the agent can take note of that fact and mark such objects as *sharp*. When it encounters a *sharp* object, the agent can recognize that goals that involve cutting are potentially achievable, or that goals that involve an object's remaining whole are potentially threatened. These tags are reference features. The characteristics of reference features are described in detail in chapter 3, the reference features that PARETO uses are discussed in chapter 6, and chapters 6 and 7 cover the use of reference features in recognizing threats and opportunities.

Reference features can be used to recognize the many opportunities that are indicated by a single *critical factor*. For example, consider again what happens when you come across a knife as you are looking for some scissors with which to cut some string. Although there are many conditions that must be met in order for you to be able to cut the string, nearly all of them are easy to achieve in most circumstances. For example, it is always easy to find somewhere to put the ball of string while you are cutting a length of it, and there is always somewhere for you to stand while you do so. The critical factor in cutting string is the presence of something that will actually perform the *cutting* action, and the presence of something that will cut string is thus enough on its own to constitute an opportunity for the goal. The presence of an object with the reference feature *sharp* therefore indicates a potential opportunity for the goal of cutting string. Similarly, if you need to pry the lid off a can of paint, it is usually easy to find a suitable place to rest the can while you are opening it, but the presence of an object with which to perform the actual prying is more problematic. The presence of an object with the reference feature *prying* can therefore be used to indicate that there is a potential opportunity for this goal.

In this dissertation I argue that there are many opportunities that depend on a single critical factor, and that an agent must be able to recognize these opportunities if it is to respond appropriately in an ever-changing world. The development of a set of reference features for the domain in which the agent will operate is thus crucial.

Most current research on the problem of recognizing the need to make unforeseen decisions fails to address the issue of opportunism explicitly (Bresina and Drummond 1990; Ferguson 1992; Lyons and Hendriks 1992; McDermott 1992). In general, this area of research relies heavily on projecting the agent's current plans to determine when replanning would be desirable. As projection may involve arbitrarily complex reasoning, this approach thus fails to address the problem of recognizing the need to make decisions quickly enough that the agent can respond appropriately in a dynamic world.

The earliest work on opportunity recognition, by Hayes-Roth and Hayes-Roth (1979), looked at opportunism in plan construction, but did not extend the concept to opportunism in plan execution. Birnbaum (1986) argues that the need for opportunism means that plan construction and plan execution must be highly integrated; the approach I develop in chapter 3, which uses opportunities as a unifying framework for both activities, is consistent with this view.

Birnbaum and Collins (1984) present a theory of opportunism in plan execution based on the idea that opportunity recognition should be goal-driven as opposed to environment-driven. They suggest that each goal should be an active *mental agent*, performing the reasoning necessary to recognize opportunities to achieve itself. The problem with this approach is that it provides no answer to the question of how agents can recognize opportunities with minimal cognitive effort. Sharing the reasoning out among the agent's goals does not necessarily reduce the amount of reasoning required.

Hammond and his colleagues (Hammond et al. 1993) address this problem by associating with each goal those features in the environment that will be involved in achieving it. Such features might include tools and resources used in achieving the goal, locations at which the goal can be achieved, and so on. The agent can then recognize opportunities to achieve the goal when the features associated with it appear in the agent's environment. This approach relies on having specified the plan that will be used to achieve the goal in enough detail that the tools and so on involved in the plan are already known. It does not allow an agent to recognize opportunities for goals that it as yet has not decided how to achieve, and it does not allow an agent to recognize opportunities for goals that require a different method of achievement from the one in the current plan. For example, it is not clear that Hammond's approach would allow an agent to recognize the opportunity discussed above, in which the presence of a knife allows an agent to abandon its plan to find some scissors. The approach that I outlined above, and that is described in greater detail in the remainder of this dissertation, is thus more flexible than that proposed by Hammond and his colleagues.

In the approach that I have outlined, information about the current state of the world is required for two purposes:

- To recognize potential opportunities.
- To decide whether to take advantage of opportunities.

The information that an agent acquires in order to keep track of its changing environment and recognize the need for unforeseen decisions must be cheap: the unpredictability of the world means that information acquisition for this purpose must take place frequently, and the information that is acquired must cover most aspects of the current situation. Information acquisition for this purpose is thus relatively general and unfocused.

Information acquisition in support of actually making decisions is, in contrast, a highly directed activity. The particular information that is required can often be specified in some detail. For example, if you find a bowl of cereal sitting on the table when you emerge from your shower, you must decide whether to continue with your plan to make yourself a bowl of cereal or whether you can use the one on the table. Which course of action you should follow depends on whether your roommate made the cereal for herself, or whether she made it for you. There are several ways in which you could find out the actual state of affairs. You could, for instance, ask her; or you could reason that she never does anything for other people, so she must have made it for herself; or you could reason that she is always keen to help other people, so she won't mind it you eat it; and so on. In general, acquiring the necessary information may be arbitrarily complex and there may be many different ways of doing it. The task of acquiring information in support of decisions is thus a planning task (Pryor and Collins 1991; 1992); when the agent realizes that it must make a decision, it can and should decide in advance how it will acquire the information that is needed to support that decision. This issue is discussed further in section 1.4.

1.3.3 Information acquisition in plan execution

In the previous sections I have discussed how the need to acquire information during plan execution arises out of the need to make decisions. These decisions are of two types:

- *Deferred* decisions—those that could not be made during the planning process due to a lack of information.
- *Unforeseen* decisions—those that were not considered during the planning process.

In both cases, information must be acquired to support the making of the decision. As we have seen, the actions required to acquire information in support of deferred decisions can be planned in advance. Once the need to make an unforeseen decision has been recognized, the agent has a goal to acquire the information required to support that decision, and can plan its information gathering actions accordingly. The agent must also acquire information about the current state of the world so that it can recognize those situations in which unforeseen decisions arise.

There are thus two types of information acquisition that an agent must perform during plan execution:

- It must acquire information that enables it to recognize the need for decisions.
- It must acquire the information that is needed in the making of decisions.

The first type of information acquisition must be performed frequently and on a regular basis; its purpose is not to learn specific facts about the current state of the world but rather is to enable the agent to keep track of its unpredictable environment. The acquisition of information in order to keep track of the agent's environment is thus independent of the agent's other goals: it is not performed in response to the need to make specific decisions. The agent must have a specific goal to acquire the necessary information.

The second type of information acquisition is driven by the agent's goals to make decisions. It is performed in response to the agent's need to decide exactly how it should achieve its goals. The purpose of this information acquisition is to learn the specific facts about the current state of the world that are required in order to make these decisions. This is thus a goal-driven activity and, like other goal-driven activities, it can be planned in advance.

1.4 Acquiring information

We have seen that much information acquisition during plan execution is a goal-driven activity; the goals arise as preconditions of the decisions that the agent must make. In fact, in these situations information acquisition is a *planning task* (Pryor and Collins 1991; 1992). One of the goals of this dissertation is to demonstrate that the acquisition of information need not be given special treatment in planning, but that it can be treated as just another task to be accomplished. Cassandra's plans include steps to acquire information in support of deferred decisions, and the plans that PARETO executes specify how to acquire the information that is required. PARETO also plans to acquire information in support of unforeseen decisions, and to acquire information to keep track of its environment.

Any serious approach to the problems facing an agent operating in an unpredictable world must address the issues of information acquisition: when and how the agent should gather information about its environment. In the work described in this dissertation the acquisition of information is seen as arising out of the need to make decisions. All decisions, including both deferred and unforeseen decisions, have information preconditions which must be planned for and achieved in the same way as the preconditions of any other actions. The explicit treatment of decisions thus demands the explicit treatment of the acquisition of information. The plans constructed by Cassandra include explicit decision actions, and also allow arbitrarily complex subplans to acquire information in support of those decisions. The procedures that PARETO uses to make unforeseen decisions also allow the use of fully generalized information acquisition plans. The design of PARETO also addresses the second purpose of information acquisition—to allow the agent to keep abreast of its environment and to recognize the need for unforeseen decisions. PARETO has a specific goal to acquire information for this purpose, and the achievement of this goal is through the execution of plans. In both Cassandra and PARETO the acquisition of information is thus a planned activity.

If information acquisition is to be considered as a planning task, then ways of acquiring information must be considered in the light of their possible use in plans. There are three ways of acquiring a piece of information: *perceiving* it by direct observation, *retrieving* it from memory, and *inferring* it from other information that is already known (Haas 1986). For example, suppose you have some groceries to buy; there is no point in going to the supermarket unless you know it will be open when you get there. You could find out whether it is open in a variety of ways, including:

- Remembering that the supermarket is open 24 hours a day and reasoning that it will be open now (retrieval and inference);
- Remembering that a friend said that the supermarket has recently changed its hours and is now open late at night, and assuming that it will be open now (retrieval and inference);
- Telephoning the supermarket, asking if it will be open at the time in question, and listening to the answer (perception);
- Telephoning the supermarket, asking when it is open, and then working out if it will be open at the time in question (perception and inference);
- Going to the supermarket and looking to see if it is open (perception);
- Going to the supermarket, reading its opening hours on a notice by the door, and then working out whether it is open or not (perception and inference);
- Remembering having been to the supermarket at the same time the previous week and finding it open, and assuming that it will also be open this week (retrieval and inference).

Examples of this sort could be propounded almost indefinitely. In this list, some of the suggested methods are clearly inefficient in some circumstances, and most of them involve several different actions of various types. All of them have preconditions that must be achieved before they can be used. In short, these methods exemplify plans to acquire information. In general, such plans may be arbitrarily complex and include actions directly in service of acquiring the desired information as well as actions that achieve various action preconditions.

Actions to acquire information have preconditions just like other actions; for example, just as an agent must be holding an object in order to put it down, it must have a fact in memory in order to recall it, it must be close to an object in order to see it (a type of perception), and it must know the relevant antecedents in order to make an inference. It may therefore be necessary to plan for information preconditions in the same way as it may be necessary to plan for any other type of precondition. An agent cannot assume that information acquisition will always be easy or even possible—the acquisition of information has no special status among the agent's possible activities. This observation follows directly from the unpredictability of the world. If the world were predictable, then information acquisition would indeed be free and totally accurate, as assumed by traditional planning systems. In an unpredictable world, the arguments against traditional planning also lead to the necessity of treating information acquisition as a planning task like any other.

Plans to acquire information are no different from plans for any other action; all plans consist of steps that achieve subgoals of the overall goal. Plans to acquire information may contain many different types of actions, some of which are information-gathering actions but others of which are not (such as moving towards an object). Moreover, plans for other goals may include information-gathering actions. An agent must be able to treat information-gathering actions in exactly the same way as it treats other actions.

Previous research in plan execution has on the whole failed to consider the general problem of information acquisition in support of decision making. Neither deferred nor unforeseen decisions have been treated as explicit actions, and the acquisition of information has rarely been treated as an issue of planning to achieve specific information goals. I describe how PARETO treats information acquisition as a planned activity in chapter 5, which also includes examples of the way in which PARETO acquires information in support of decisions. The issues of information acquisition in support of unforeseen decisions are discussed more fully in chapters 6 and 7.

Research in plan execution has on the whole concentrated on perception as a method of information acquisition. While there has been a general acceptance of the fact that information acquisition through perception should be a deliberate activity, researchers have generally ignored other possible methods of information acquisition. In particular, they have failed to take account of the fact that inference and memory retrieval are in many circumstances genuine alternatives to perception, and should therefore be treated on an equal footing when it comes to planning to acquire information. Cassandra, as a domain-independent planning system, is not influenced by the implementation of any of the actions in its plans except those aspects of any action specified in its preconditions and effects. Cassandra is thus able to use any method of information acquisition in its plans. The plans used by PARETO are, in contrast, heavily dependent on the agent design. However, they make full use of all three methods of information acquisition, an aspect of PARETO which is discussed in detail in chapter 5.

In the remainder of this section I discuss each of the three methods of information acquisition—perception, memory retrieval, and inference—in turn.

1.4.1 Perception

Many researchers in the field appear to make the implicit assumption that perception is the only way to acquire information about the current state of the agent's world (Kaelbling 1987; Olawsky and Gini 1990; Simmons 1990; Chrisman and Simmons 1991). Brooks has made his view more explicit (Brooks 1990) than most; interestingly enough, he appears to consider all three methods of information acquisition—perception, memory retrieval, and inference—only to reject all except perception. As he also eschews planning altogether, his approach is the complete antithesis of the view that planned information acquisition should treat all three methods equally.

Brooks's argument is based on the observation that any internal representation of the world grounded in memory and inference is bound to be inaccurate. When Brooks says "the world is its own best model," he is pointing out that the world is, by its very nature, a totally accurate representation of itself. This accuracy makes it tempting to forsake internal representation of the world almost entirely, using perception to gather information at the time when the agent needs it. Unfortunately for such an approach, many aspects of the world are not available for quick and easy inspection.

To adapt an example from Hanks (1990), if there is a drawbridge on the route being considered by a robot delivery truck, the truck would like to know whether the bridge is up or down. Obviously, this information can in principle be gained by simply observing the bridge. However, if the truck is ten miles away, trying to decide whether or not to take the route that uses the drawbridge, this is not very helpful. Luckily, there are other ways in which it could find out whether the bridge is up or down without having to go and look at the bridge. For instance, it could also use any one of the following inferences:

- The bridge was down an hour ago when I passed it, so it is probably still down.
- The bridge never goes up before noon, and it is ten now, so it will be down.
- It's -40 degrees, the river is frozen solid, no ships will be moving, so the bridge will be down.

None of these are as reliable as observing the bridge, but they save the truck from a possibly wasted journey, and should be considered when planning how to acquire the information. Notice that these inferences are themselves based on information acquired through memory retrieval and perception. This is a point that I shall return to later: information acquisition often involves a combination of methods.

Perception on its own is thus not adequate to serve the information acquisition needs of an agent in an unpredictable world. However, the use of perception is crucial: it is the only method by which an agent can ensure that its internal model reflects the actual state of the world.

1.4.2 Memory retrieval

It is generally accepted that an agent executing plans must retrieve information from memory, but memory retrieval is not usually considered as an explicit mechanism for information acquisition. Instead, it is often implicitly assumed in work on plan execution that all information in memory is immediately available at no cost to the agent (Hayes-Roth 1990a; McDermott 1992, for example). Interestingly enough, Brooks, with his explicit rejection of the use of internal models, has addressed this issue—but he also explicitly rejects the notion of plans and hence of plan execution (Brooks 1990). Haas (1986) discusses memory retrieval as an explicit action to be performed by an agent in determining its beliefs, but does not address the problems of constructing plans containing such actions, or other uses of retrieval.

The work of Hammond and his colleagues (Hammond et al. 1993) specifically addresses issues of memory organization and retrieval for agents executing plans in unpredictable worlds. However, the issues they address are concerned more with using memory organization to help in recognizing opportunities and in choosing plans for the agent's goals, rather than with memory retrieval as a form of information acquisition. Dehn (1989) looks at memory organization from the same point of view. Other work in memory organization and retrieval has concentrated more on story understanding (Schank 1982), explanation (Schank 1986; Owens 1990), and plan construction (Hammond 1984; Zito-Wolf and Alterman 1993) than on plan execution. However, this work is by no means irrelevant to the issue of how retrieval from memory can be used as an information acquisition mechanism—it all supports the notion that the retrieval of information from memory depends crucially on the way in which memory is organized and the probe that the agent uses. In general, this work confirms the view that memory retrieval may be arbitrarily complex. There is certainly no guarantee that it is always easier to retrieve a given piece of information from memory than it is to acquire it through other means.

For example, consider the reasons for making a shopping list. You must know what you need from the supermarket, or you wouldn't be able to write it down on the list. However, you are not confident of being able to retrieve this information when you need it; without a list, you could find yourself returning from the supermarket without half the things you wanted to buy. To take another example, suppose you are deciding on the route you should take to get to a friend's house. There are two possible routes, one of which is usually shorter but which has recently been subject to sporadic delays from construction work. Although you drove past the turnoff this morning, you can't remember whether the detour signs indicated that construction work was in progress or whether the route was clear. Although the information is in your memory, and could presumably be retrieved given the right cue, it is at the moment unavailable to you.

There is also no guarantee that information retrieved from memory will be reliable. The information stored may not have been accurate in the first place, it may have become outdated through the dynamism of the world, and the retrieval process may introduce further inaccuracies. For example, construction might have started without detour signs being posted; construction may have started since you drove past; or you may be sure that the road was clear this morning, when in fact you are thinking of yesterday morning.

Since the information acquired through memory retrieval may vary in cost and reliability, an agent must be able to reason about whether it is an appropriate method of information acquisition to use in any given situation, or whether an alternative would be better. In short, memory retrieval must be explicitly considered as a possible method of acquiring information.

1.4.3 Inference

In general, resource-limited agents cannot perform unlimited inference on the facts in their possession—knowing all the antecedents required for an inference, along with the inference rule, is not the same as knowing the conclusion (Hintikka 1962). This lack of consequential closure means that performing inference may result in the acquisition of new information. The forms of reasoning available to an agent include both strict logical inference and forms of plausible reasoning such as non-monotonic reasoning or statistical inference (Davis (1990) gives a good overview). Whatever the form of inference used, the reasoning required to arrive at a given conclusion may be arbitrarily complex and may involve the acquisition of large amounts of supporting information. It cannot therefore be assumed that the use of inference will be more efficient than other methods of information acquisition.

The principle that unlimited inference is not possible is generally accepted, although this acceptance is often not reflected in the design of specific systems. Recent work has explicitly addressed the issue of when inference is desirable by investigating the utility of performing reasoning in terms of its effects on the agent's actions (Dean and Boddy 1988; Boddy and Dean 1989; Horvitz et al. 1989; Boddy and Kanazawa 1990; Russell and Wefald 1991), continuing a long tradition of the use of decision theory in planning (Feldman and Sproull 1977; Horvitz et al. 1988; Hansson et al. 1990; Hartman 1990). This work has not addressed the issues when inference is a more appropriate method of acquiring information than, say, perception.

There has also been little work on how inference should be represented in an agent's plans. Haas (1986) argued that plans should include specific actions to perform inference, but did not suggest any specific design for a planning system that would construct such plans. Hayes-Roth (1990b) appears to make provision for reasoning in the global control plans that direct the attention and activities of her GUARDIAN system, and Firby (1989) suggested an extension of his RAPs system to provide for the representation of reasoning actions in the same format as physical and perceptual actions. Hartman (1990) extends this notion, and integrates planning and execution by treating planning as an action with an uncertain outcome. On the whole, however, this aspect has been neglected.

1.4.4 Methods of information acquisition

We have seen in the previous sections that an agent cannot rely on any one method of information acquisition, but that different methods are appropriate in different circumstances. In particular, it may sometimes be impossible to use a specific method to acquire the necessary information. The ability to use perception is often limited by physical constraints: for example, we can only see objects if there is a clear line of sight, and can only touch objects that are within reach. Retrieval from memory relies on the information being in memory, and inference requires knowledge of the antecedents and the relevant rules of inference. The information of acquisition is thus a non-trivial task, and plans to acquire information may be arbitrarily complex. The design of an agent intended to operate in an unpredictable world must therefore not only address the issue of how the agent can acquire information, but must also address the issues of planning to acquire information.

Both Cassandra and PARETO reflect the view that no one method of information acquisition should be given special treatment and allow fully general planning for all three methods. Cassandra, as a domain-independent planner not linked to any specific agent architecture, simply treats information acquisition actions like any other actions, considering them solely in terms of their preconditions and effects and ignoring the processes that are required to execute them. The plans that PARETO executes specify all the memory retrieval, inference, and perception to be performed. PARETO's plans specify different ways of achieving goals to be used in different circumstances, and information goals are no exception. For example, a plan to acquire a specific piece of information may involve first trying perception if memory retrieval fails, and then falling back on inference.

Cassandra and PARETO are atypical of recent research in this area. In general, work in plan execution has looked at the utility of using a single method of information acquisition compared to the utility of acting without information (Horvitz et al. 1989; Chrisman and Simmons 1991), but has not addressed the problem of choosing from among alternative methods of acquiring information. As the examples in this section have demonstrated, the acquisition of a particular piece of information often requires a complex plan involving a combination of information acquisition methods and other physical actions. Planning to acquire information is just as complex as planning to achieve any other type of goal.

The problem of planning to acquire information has mostly been addressed in terms of perception (Firby 1989; Simmons 1990; McDermott 1992). Other researchers have concentrated more on the problems of an agent faced with a continuous flow of data from perception (Kaelbling 1987; Hayes-Roth 1990a). None of these researchers have seriously considered the issues involved in treating perception, memory retrieval and inference as equally valid methods of acquiring information. Although Cassandra and, especially, PARETO represent an important first step in this direction, allowing as they do the full specification of all methods of information acquisition in their plans, neither of these systems addresses in any detail how to choose between different methods of information acquisition. This is, of course, partly a reflection of the field's general neglect of how to choose between different methods of achieving a goal, but nonetheless represents an important direction for future research.

1.5 Thesis outline

This dissertation is concerned with information acquisition in the course of plan execution. As we have seen, the need to acquire information arises out of the need to make decisions. The decisions that an agent must make are of two types: *deferred* decisions and *unforeseen* decisions. Deferred decisions are those that could not be made in advance because the necessary information was not available; they can be specified in the agent's plans, together with the actions required to acquire the information required to make them. The need to make unforeseen decisions arises out of the occurrence of situations that were not predicted in the agent's plans; the agent must be able to recognize when such situations affect its goals and thus lead to the need for a decision. It must thus acquire information for the purpose of keeping track of its changing environment. Having recognized the need for a decision, the agent must also be able to acquire the information that will support it.

Cassandra, a planner that extends the classical paradigm by being able to plan in uncertain and complex worlds, is described in chapter 2. Cassandra recognizes the need to defer decisions and constructs plans that make explicit provision for making these deferred decisions at the time of plan execution. Cassandra's plans also include provisions for acquiring the information required to make these decisions.

In chapter 3 I discuss how, in order to recognize the need for unforeseen decisions, an agent must be able to recognize potential opportunities, and present a mechanism based on reference features by which it can do so. This mechanism is based on the observation that, even in an unpredictable world, objects tend to have consistent causal effects. These tendencies can be labeled with reference features, which can be used as the basis of a heuristic filtering process that indicates those of an agent's goals for which there are potential opportunities. This mechanism is implemented in PARETO, a plan execution system described in chapters 4 and 5. The plans that PARETO executes include explicit provisions for acquiring information through perception, memory retrieval, and inference.

Chapter 6 describes how opportunities are recognized in PARETO, and in chapter 7 I describe how PARETO uses reference features to indicate potentially problematic interactions between goals. Chapter 8 concludes the description of PARETO by describing how the recognition of opportunities forms the basis of its ability to handle many simultaneous goals.

Finally, chapter 9 brings together the strands presented in the earlier chapters and considers directions for further work.

CHAPTER 2

CASSANDRA: PLANNING TO MAKE DECISIONS

2.1 Introduction

To plan is to decide in advance on a course of action that will, if successful, result in the achievement of a goal. However, in an unpredictable world it is impossible to decide in advance on every detail of a particular course of action. All decisions depend on information about the state of the world, and decisions that are made in advance depend on information about the future state of the world. The information that is needed is not always available, however; at the time that an agent chooses a plan, there may be some decisions that it cannot make because it lacks the necessary information. Such decisions must be deferred until the required information is available.

The planner can provide for deferred decisions in the plans it constructs. Many plans that we use in our everyday lives are of this type; the decisions are often made explicit when the plan is communicated to someone else, e.g., "try taking Western Avenue, but if it's blocked use Ashland," or "crank the lawnmower once or twice, and if it still doesn't start jiggle the spark plug." These plans include separate courses of action to be taken in different contingencies, and provision for deciding which course of action should actually be taken. In other words, they are *contingency plans.*

Providing for deferred decisions by constructing contingency plans has a number of advantages. By including a decision as an action in a plan, an agent can plan in advance to acquire the information that will be needed in order to make it. The agent can thus ensure that it will actually be able to make the decision in question when the need to do so arises. Deciding in advance on a course of action for each contingency means that the agent need not reason from scratch when it must make a decision during plan execution. Instead, it need only determine which of the available courses of action it should pursue. The simplicity of this type of decision as compared to the complicated procedure of constructing a plan means that the agent can respond more rapidly to the unpredictability of its world.

So-called *classical planners*[1] cannot construct contingency plans, due primarily to their reliance on two *perfect knowledge* assumptions:

1. The planner will have full knowledge of the initial conditions in which the plan will be executed, e.g., whether Western Avenue will be blocked.
2. All actions have fully predictable outcomes, e.g., cranking the lawnmower will definitely either work or not work.

By effectively ruling out uncertainty, these assumptions make it impossible even to represent the notion of a contingency plan within a strictly classical framework.

[1] Systems such as STRIPS (Fikes and Nilsson 1971), HACKER (Sussman 1975), NOAH (Sacerdoti 1977), MOLGEN (Stefik 1981), and SIPE (Wilkins 1988). The term "classical planner" is due to Wilkins.

These assumptions are invalid in an unpredictable world. Consider the following example: an agent must transport a pile of clean laundry from the laundry room to its apartment, having had its laundry basket stolen from the laundry room while its clothes were drying. Carrying the laundry without using a basket makes it likely that the agent will drop some items along the way. If the agent drops an item without seeing it fall, it will lose that item. Possible plans for coping with this contingency include:

1. The agent carries the laundry in its arms and walks backwards so that it will notice when something drops.
2. The agent carries the laundry in its arms and walks forwards, then retraces its route to search for dropped items after it has delivered the laundry to its apartment.
3. The agent carries the laundry in its arms and walks forwards, turning around every so often to look for dropped items.

Each of these is an example of a contingency plan. In every case, there is a part of the plan, namely retrieving a dropped piece of laundry, that will be executed only when a certain contingency arises. Each plan contains at least one step the sole purpose of which is to determine whether the contingency holds or not.

From this example we can see a number of issues that must be addressed in building a contingency planner:

- The planner must be able to anticipate uncertain outcomes of actions, such as that an item of laundry may or may not drop when it is carried.
- The planner must be able to recognize when an uncertain outcome threatens the achievement of a goal, such as that a dropped item will not be carried back to the apartment.
- The planner must be able to make contingency plans for all possible outcomes, such as picking up something that has dropped.
- The planner must be able to schedule information acquisition actions that detect the occurrence of the contingency, such as looking to see if something has dropped.
- The planner must produce plans that represent the contingencies in which actions should be performed; for example, if nothing has dropped there is no need to pick anything up.

In this chapter I describe Cassandra,[2] a contingency planner whose design addresses these issues. In the rest of this section I consider representation issues: how Cassandra's plans are represented, and how decisions are represented in them. Section 2.2 summarizes the non-contingent planning algorithm on which Cassandra is based. It is followed by a discussion of the issues that must be considered when modifying the algorithm for contingency planning and then, in section 2.4, a description of the modified algorithm. Section 2.5 looks at how other recent contingency planning systems have addressed the issues.

An important issue that is *not* addressed in the design of Cassandra is the problem of determining whether a contingency plan should or should not be constructed in the face of a particular source of uncertainty. Since everything in the real world is uncertain to some degree, the number of contingencies against which the agent could potentially plan is unlimited. The agent must therefore decide which sources of uncertainty are actually worth worrying about. Such a decision requires consideration of the relative likelihoods of the various eventualities and the relative costs of planning for the diverse contingencies in advance, among other things. There appears to be no current theory that adequately

[2] A fuller description of Cassandra and the issues surrounding its design may be found in (Pryor and Collins 1993), on which much of this chapter is based.

accounts for all the complexities involved, although parts of the problem have been addressed separately (Boddy and Kanazawa 1990; Hanks 1990; Haddawy and Hanks 1992, for example). Cassandra's role is to construct contingency plans once it has been decided which sources of uncertainty merit consideration.

2.1.1 Cassandra's plan representation

The main components of Cassandra's representation of contingency plans are:

- An action representation that supports uncertain outcomes.
- A plan schema.
- A system of labels for keeping track of which elements of the plan are relevant in which contingencies.

Action representation

Cassandra represents actions using a modified version of the STRIPS operator (Fikes and Nilsson 1971). An operator is defined by the *preconditions* for executing an action and the *effects* that may become true as a result of executing it. For each possible effect there is in addition an associated set of *secondary preconditions* (Pednault 1988; 1991), which specify the conditions under which the action will have that effect. The preconditions of an action are sometimes known as enabling preconditions; they are the conditions which must hold in order for the action to be executed. The secondary preconditions of an effect do not affect the executability of the associated action, but instead specify the context in which the relevant effect will result from the execution of the action. The use of secondary preconditions is crucial to Cassandra's ability to use a single format to represent all sources of uncertainty.

In Cassandra's representations, uncertainty is assumed to stem from intrinsically uncertain action outcomes, i.e., outcomes that cannot be predicted on the basis of any factor that is included in the planner's representation of the situation. This formulation ignores uncertainty that might stem from outside interference during the execution of the agent's plans.[3] All other sources of uncertainty can, however, be handled within this framework. In particular:

- Uncertainty about initial conditions is handled as a special case of uncertainty about the outcomes of actions; this is possible because Cassandra, in common with many classical planners, treats the initial conditions as though they were the effects of a phantom "start step" action.
- Uncertainty about an effect due to uncertainty about the preconditions of that effect is represented indirectly; it is assumed that such uncertainty stems from the intrinsic uncertainty of some effect of a prior action. Cassandra correctly propagates uncertainty through the plan.

Cassandra represents the intrinsic uncertainty of effects by assigning them *unknowable preconditions,* which are constructed using the pseudo-predicate :unknown. By giving an effect an unknowable precondition, we indicate that it cannot be determined in advance whether that effect will result from executing the action. As an example of an operator with uncertain effects, consider the action of throwing a die while playing a board game, a representation of which is shown in figure 2.1. In order to execute the action, the agent must be holding the die, and as a result of executing the action the agent will cease to be holding the die, which will land on the table. The die might end up flat, in which case one of its six faces

[3] This is actually a limitation of classical planners in general; all change in the world is assumed to be caused directly by the actions of the agent.

```
Action:              (throw ?die)
Preconditions:       (holding ?agent ?die)
Effects:             (:effect (:and  (:not (holding ?agent ?die))
                                     (on table ?die)))
                     (:when (:unknown ?tilt T)           ; secondary precondition
                      :effect (tilted ?die))             ; effect
                     (:when (:and  (:unknown ?tilt F)
                                   (:unknown ?face F1))
                      :effect (facing ?die face-1))
                     (:when (:and  (:unknown ?tilt F)
                                   (:unknown ?face F2)
                      :effect (facing ?die face-2))
                     (:when (:and  (:unknown ?tilt F)
                                   (:unknown ?face F3)
                      :effect (facing ?die face-3))
                     ...
```

Figure 2.1 Representation of operating a faulty drill

will be facing up, or it might land tilted against the edge of the board, in which case no on face will be on top. There are thus seven uncertain effects (some are omitted from the figure for the sake of brevity).

Obviously, these effects are related. In particular, the seven effects result from only two sources of the uncertainty: one source of uncertainty determines whether the die lands flat, and the other determines which face is up. In order to express this sort of relationship, the :unknown pseudo-predicate must take two arguments, one standing for the source of the uncertainty (?tilt and ?face in the example), the other for a particular outcome of the uncertainty (for example, T for the outcome in which the die is tilted, F for the outcome in which it is flat). The source of uncertainty is indicated by a variable in the operator schema because each instantiation of the operator will introduce a new source of uncertainty. The planner must bind this variable to a unique name (i.e., a skolem constant) when the operator is instantiated.

If a set of effects are represented as stemming from the same source of uncertainty, the following propositions are assumed:

1. Each outcome name (e.g., T or F, or face-1, face-2, etc.) designates a unique outcome for a given source of uncertainty.
2. Different outcomes of a given uncertainty are mutually exclusive.
3. The set of named outcomes is exhaustive.

These propositions license two critical judgments that Cassandra must be able to make in order to construct a viable contingency plan:

1. That two actions or effects may not co-occur because they depend upon different outcomes of the same uncertainty.
2. That a goal will necessarily be achieved by a plan, because it will be achieved for every possible outcome of every relevant uncertainty.

Basic plan representation

Cassandra's plan representation is an extension of that used in UCPOP (Penberthy and Weld 1992) and SNLP (Barrett et al. 1991; McAllester and Rosenblitt 1991), which is in turn derived from the representation used in NONLIN (Tate 1977). A plan is represented as a schema consisting of the following components:

- A set of *steps*, each of which is an instantiation of an *operator* (see previous section).
- A set of *effects*, namely, the anticipated effects of the steps that are scheduled.
- A set of *links* connecting steps that achieve effect with the steps for which those effects are preconditions. Links in effect denote *protection intervals*, i.e., intervals over which particular conditions must remain true in order for the plan to work properly.
- A set of *variable bindings*, made in the course of instantiating the operators from which the plan is constructed.
- A *partial ordering* on the steps.
- A set of *open conditions*, which are preconditions that have not yet been established.
- A set of *unsafe links*, which are causal links the conditions of which could be falsified by other effects in the plan.

A plan is *complete* when there are no open conditions and no unsafe links.

Representing contingencies

When a plan that has been produced by a conventional planner is carried out, it is assumed that every step in that plan will be executed. In contrast, the whole point of contingency planning is to produce plans in which some of the steps may *not* be carried out. This has several important consequences for the planning algorithm:

- Plan steps must be marked in such a way that the agent executing the plan can determine which steps to perform in a given contingency.
- The planner must be able to determine whether a given effect of a scheduled action will arise in a given contingency, in order to decide whether that effect can be used to achieve a goal.
- The planner must be able to determine the contingencies in which a particular step and a particular link co-occur, in order to determine whether (and when) the step threatens to clobber the link.

Keeping track of which elements of the plan are relevant in various contingencies requires a good deal of bookkeeping, which Cassandra accomplishes by attaching a label to each element of the plan designating the contingencies in which that element plays a role. In order to construct such labels, Cassandra must have a means of designating a particular contingency. A contingency is uniquely described by a pair of symbols, one standing for the source of the uncertainty that gives rise to the contingency, the other standing for a particular outcome of that uncertainty. For example, as described above, an instance of the operator for throwing a die introduces two sources of uncertainty, one of which (whether the die lands flat or tilted) has two possible outcomes. For a given instance, the source of uncertainty might be designated TILT1; as we have seen, the two outcomes are designated T (the die lands tilted) and F (the die lands flat). The contingency in which the die lands tilted would in this instance be designated [TILT1: T], while the contingency in which the die lands flat would be designated [TILT1: F]. Similarly, the outcomes for the other source of uncertainty, which face is up, would be designated [FACE1: face-1], [FACE1: face-2], and so on. These labels can be combined to represent the effects of more than one source of uncertainty: for example, the die will land with a "one" facing up in the joint outcome [TILT1: F][FACE1: face-1]. The next section considers the problems that arise in combining sources of uncertainty in this way. This labeling system allows Cassandra to determine by inspection whether different contingencies are associated with alternative outcomes of the same source of uncertainty.

Contingency labels can be attached to actions, effects, and goals. The labels can be either positive or negative. Labelings have the following interpretations:

- *Positive contingency label on an action*: The action must be executed in that contingency.
- *Negative contingency label on an action:* The action must not or cannot be executed in the contingency.
- *Positive contingency label on an effect:* The effect must be established in the contingency.
- *Negative contingency label on an effect:* The effect must not or cannot be established in the contingency.
- *Positive contingency label on a goal*: The goal must be achieved in the contingency.[4]

Contingency labels allow quick determination of whether a particular plan element is relevant in a particular contingency. Positive contingency labels indicate elements that are definitely part of the plan for that contingency, negative contingency labels indicate elements that are definitely *not* part of the plan for that contingency, while elements that are unlabeled with respect to a given contingency may or may not occur in that contingency.

2.1.2 Representing decisions

In general, plans involving contingencies *branch*. When an agent executes a branching plan, it must at some point decide which branch to take. Rather than treating this decision as an explicit part of the plan, previous work has in effect simply assumed that the agent will execute those steps that are consistent with the contingency that actually obtains (Warren 1976; Peot and Smith 1992). However, the agent cannot make this determination automatically; in order to know which contingency holds during execution, an arbitrarily large amount of work may be necessary, in particular the work of gathering information upon which the decision can be based. To ensure a viable plan, the planner must be able to guarantee that the steps required to gather information do not conflict with those required to carry out the rest of the plan. Therefore, the planner must in general be able to include information gathering steps, as well as any other steps that support decision making, in the plan it is constructing. Cassandra achieves this by representing decisions explicitly as plan steps. The preconditions of these decision steps include goals to be in possession of information relevant to making the decision; the scheduling of actions to obtain information is thus handled by the normal planning process.

For instance, consider the contingency plan alluded to above: *"try taking Western Avenue, but if it's blocked use Ashland."* During the execution of such a plan, the agent must at some point decide which alternative branch of the plan to execute. The decision step in this case would have the precondition of knowing whether Western Avenue is blocked or not, which would cause the planner to schedule an information-gathering action to check the traffic status on Western. This operation might in turn have the precondition of being on Western.

Assuming the goal of the plan is to be in Evanston, the final plan might be as depicted in figure 2.2. Note that solid lines in the diagram represent links, with the operation at the tail of the link achieving a condition for the operation at the head of the link. Heavy lines represent control flow after a decision. In

[4] There is no need for negative contingency labels on goals; the negative labels on effects are used to prevent a goal from being achieved by an effect in an incompatible contingency.

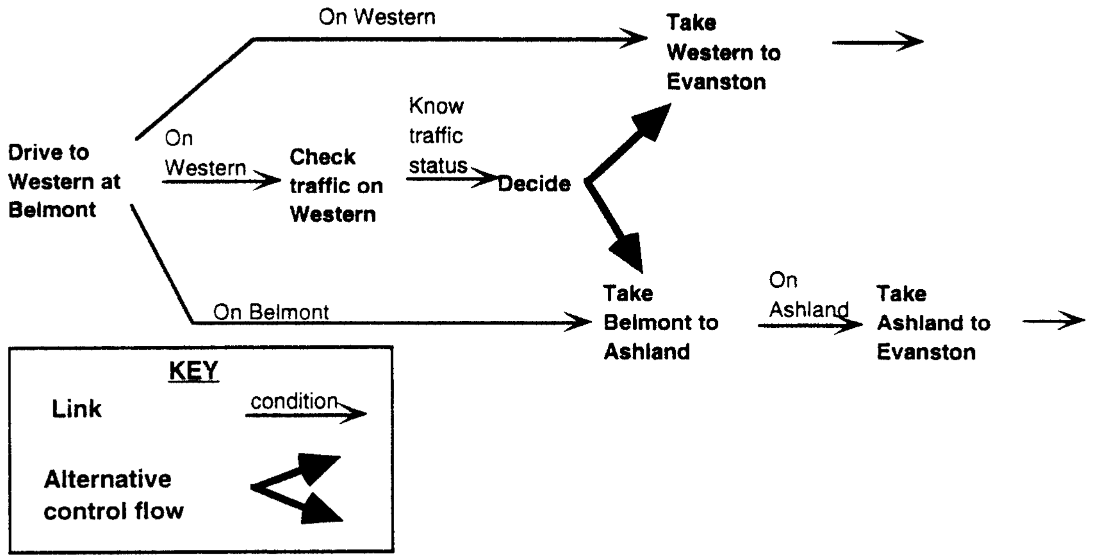


Figure 2.2 A plan that includes a decision step

this case, the agent will take Western to Evanston if it decides on one contingency, and will take Belmont to Ashland and Ashland to Evanston if it decides on the other.

Notice that in order to determine the appropriate precondition for a given decision step, the planner must have some way of determining exactly what it will need to know in order to make the decision at execution time. This somewhat complex determination depends in part on how the decision-making process is to be carried out. In Cassandra, decisions are modeled as consisting of the evaluation of a set of condition-action rules of the form:

if *condition 1* then *contingency 1*
if *condition 2* then *contingency 2*
. . .
if *condition n* then *contingency n*

Each possible outcome of a given uncertainty gives rise to one decision rule; the condition of this decision rule specifies a set of effects that the agent should test in order to determine whether to execute the contingency plan for that outcome. For example, the decision rules for the driving plan example would look like this:

if *Western Avenue is blocked*	then *use Ashland route*
if *Western Avenue is not blocked*	then *use Western route*

Cassandra's derivation of inference rules in decisions is explained in detail in section 2.3.2.

The preconditions for a decision step are goals to know the truth values of the conditions in the decision rules. For example, the precondition for the decision step in the driving plan is to know whether Western Avenue is blocked. These goals are treated in the same way as are the preconditions of any other step. Thus, Cassandra requires no other special provisions to allow the construction of information-gathering plans.

Cassandra's separation of information gathering from decision-making allows one information-gathering step to serve several decisions. This allows a flexible use of information-gathering actions; there is no effective difference between such actions and any other action that may appear in a plan.

New step Add a new step to the plan that has an effect that will establish the open condition. Add the step preconditions and the secondary preconditions of the effect as open conditions. The open condition becomes a completed link.

Reuse step Make the open condition into a complete link from an effect of an existing plan step. Add the secondary preconditions of the effect as open conditions.

Figure 2.3 Resolving open conditions

2.2 Planning without contingencies

In this section I shall briefly review the basic planning algorithm on which Cassandra is based. This follows closely that used in UCPOP (Penberthy and Weld 1992), which is in turn based on SNLP (McAllester and Rosenblitt 1991) The principal difference between UCPOP and SNLP is the use of secondary preconditions, an adaptation originally suggested by Collins and Pryor (1992a). Readers wishing for more details on the evolution of this class of planners should see also (Tate 1977; Chapman 1987; Barrett et al. 1991; Collins and Pryor 1992b; Hanks and Weld 1992; Barrett and Weld 1993).

Cassandra does not attempt to construct a contingency plan until it encounters an uncertainty.[5] Up until this point, it constructs a plan in much the same manner as other planners in the SNLP family. In fact, if no uncertainty is ever introduced into the plan, Cassandra will effectively function just as the UCPOP planner would given the same initial conditions. Planning proceeds through the alternation of two processes: *resolving open conditions* and *protecting unsafe links.* Each of these processes involves a choice of methods, and may therefore give rise to several alternative ways to extend the current plan. All possible extensions are constructed, and a best-first search algorithm guides the planner's exploration of the space of partial plans.

The initial plan consists of two steps: the *start* step, with no preconditions and with the initial conditions as effects, and the *goal* step, with the goal conditions as preconditions and with no effects.[6] The planner attempts to modify its initial plan until it is *complete*: i.e., until there are no open conditions and no unsafe links.

2.2.1 Resolving open conditions

The planning process is driven by the need to satisfy open conditions. Initially, the set of open conditions consists of the input goals. In the course of planning to satisfy an open condition, new subgoals may be generated; these are then added to the set of open conditions. The planner can establish an open condition in one of two ways: by introducing a new step into the plan, or by making use of an effect of an existing step (see figure 2.3). If a new step is added, the preconditions of the step become open conditions. The secondary preconditions of the effect that establishes the condition become open conditions as well. Finally, each time an open condition is established, a link is added to the plan to protect the newly established condition.

[5] Cassandra does not address the problem of which sources of uncertainty should be planned for (see section 2.1).

[6] Representing initial conditions and goals in this way is merely a device for simplifying the description of the planning algorithm.

A link L is unsafe if there is an effect E in the plan (other than the effect LE1 that establishes the condition in the link and the effect LE2 that is either established or disabled by the link) with the following properties:

Unification One of the postconditions in E can possibly unify with the condition that L establishes or the negation of the condition that L establishes.

Ordering The step that produces E can, according to the partial order, occur both before the step that produces LE2 and after the step that produces LE1.

Figure 2.4 Unsafe links

One way of establishing a condition is simply to notice that the condition is true in the initial state. Because the initial conditions are treated as the results of the *start* operator, which is always a part of the plan, this method can be treated as establishment by use of an action already in the plan; indeed, this simplification is the motivation for representing the initial conditions in this way.

2.2.2 Protecting unsafe links

Whenever an open condition is established, links in the plan may be jeopardized either because a new step threatens an existing link, or because a new link is threatened by an existing step. The situations in which a link is unsafe are shown in figure 2.4. In general, if there is an effect in the plan that could possibly interfere with the condition established by a link, that link is considered unsafe.

There are three general methods of protecting a threatened link: *ordering, separation,* and *preservation* (see figure 2.5). Ordering means constraining the threatening action to occur either before the beginning or after the end of the threatened link. Separation means constraining the way in which variables may be bound in order to ensure that the threatening action will not in fact interfere with the threatened link. Preservation means generating a new subgoal to disable the effect that threatens the link.

2.3 Contingency planning

Cassandra proceeds as described in the previous section until the plan is completed or until an uncertainty is introduced. This section describes how uncertainties are introduced, and how they are handled. As an example of plan involving an uncertainty, consider the classic "bomb in the toilet" problem (McDermott 1987, citing Moore), in which the goal is *disarm bomb,* and the initial conditions are *bomb in package1* or *bomb in package2*. The uncertainty in this case lies in the initial conditions: depending on the outcome of the uncertainty, the *start* operator can either have the effect that the bomb is in package1, or the effect that the bomb is in package2.

Separation Modify the variable bindings of the plan to ensure that the threatening effect E cannot in fact unify with the threatened condition.

Ordering Modify the ordering of the steps in the plan to ensure that the step producing E occurs either before the step that produces LE1 or after the step that produces LE2.

Preservation Introduce a new open condition in the plan to disable E. This new open condition is the negation of E's secondary preconditions.

Figure 2.5 Resolving unsafe links

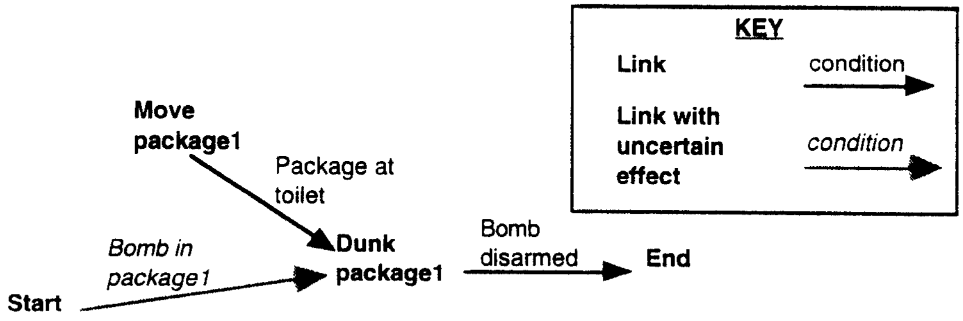


Figure 2.6 The introduction of uncertainty into a plan

2.3.1 Contingencies

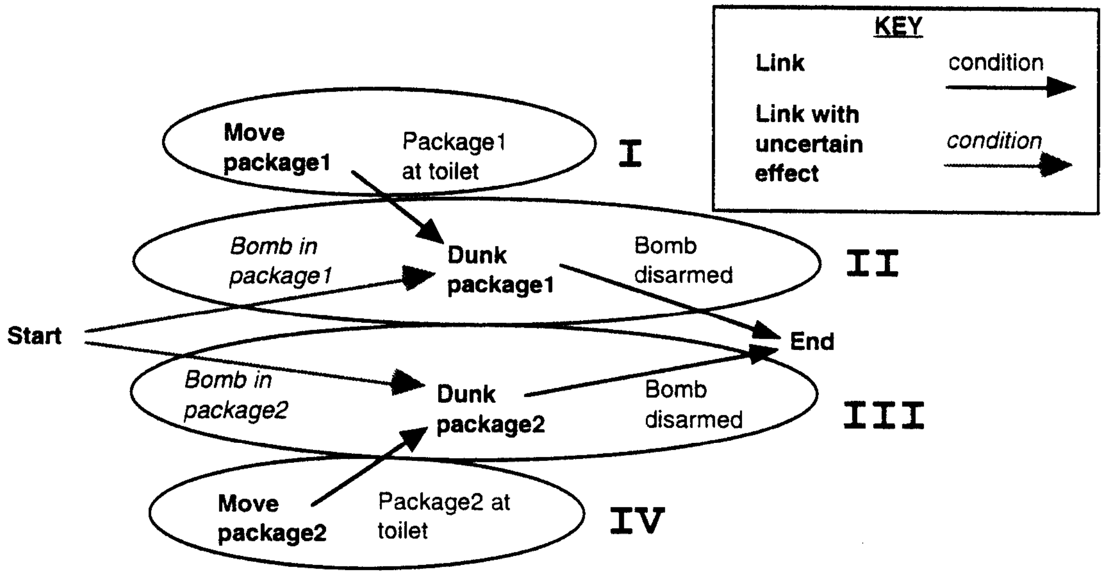
An uncertainty is introduced into a plan when an open condition in the plan is achieved by an uncertain effect, i.e., an effect with an unknowable precondition. In the bomb-in-toilet example, for instance, the planner may achieve the condition *bomb is disarmed* by selecting the *dunk* operator, which has the preconditions *the package is at the toilet,* and *the bomb is in the package*. The condition *the bomb is in the package* can be established by identifying it with *the bomb is in package1,* which is an effect of the *start* operator. However, this condition is uncertain, as the planner can determine by noting that it has an unknowable precondition. Cassandra will attempt to deal with this uncertainty by introducing a new contingency (or new contingencies) into the plan. The state of the plan just after the introduction of the uncertainty is illustrated in figure 2.6.

Cassandra notices an uncertainty when its current plan becomes dependent upon a particular outcome of that uncertainty. In effect, the plan that it has built so far becomes a contingency plan for that outcome. In order to build a plan that is guaranteed to succeed, the planner must construct contingency plans for all other possible outcomes of the uncertainty as well. This means splitting the plan into a set of branches, one for each possible outcome of the uncertainty.

Each contingency plan must ultimately result in the achievement of the overall goal of the plan, and each must enforce the assumption of the particular outcome of the uncertainty. Cassandra thus begins the construction of a contingency by making a copy of the overall goal of the plan, assigning it a label indicating the outcome of the uncertainty that it assumes, and adding it to the set of open conditions. One such goal is constructed for each alternative outcome of the uncertainty.

In planning for these otherwise identical goals, the planner must make certain that no element of the plan for a given goal relies upon an outcome of the uncertainty other than the one assumed by that goal. In other words, for a particular goal, Cassandra must ensure that neither this goal nor any of its subgoals is achieved by any effect that depends, directly or indirectly, upon any outcome of the uncertainty other than the one in the goal's label. In order to make the enforcement of this constraint as efficient as possible, Cassandra labels each element of the plan that depends upon a particular outcome of an uncertainty with all the contingencies in which that element can *not* be used—namely, those associated with other possible outcomes of that uncertainty (see discussion of labeling, above). These *negative labels* make it simple for the planner to check whether a given effect can be used to establish a particular goal.

As Cassandra adds new elements to the plan it is constructing, it must ensure that they are labeled appropriately. In addition, when a new source of uncertainty is introduced into a plan, all the elements



Element label classes

I —In 'package1' contingency
II —In 'package1' contingency; out of 'package2' contingency
III —In 'package2' contingency; out of 'package1 contingency
IV —In 'package2' contingency

Figure 2.7 A contingency plan to disarm a bomb

that depend on it must have the appropriate labels attached to them. Cassandra maintains the labels on plan elements by copying them in two different ways. In particular:

- Positive labels, which indicate the contingencies in which a plan element is required in order that the goal is achieved, are copied from goals to their subgoals. If a plan element has a positive label, any element on which that element depends must also have the positive label.
- Negative labels, which indicate the contingencies in which a plan element may not appear, are copied from subgoals to the goals that they achieve. If a plan element has a negative label, any element that depends on that element must also have the negative label.

In the bomb in the toilet example, when the plan is made dependent upon the uncertain outcome *bomb in package1*, a new copy of the top level goal *disarm bomb* is added to the set of open conditions, with a label indicating that it belongs to the contingency in which the bomb is in package2 (see figure 2.7).[7] The existing top level goal and all its subgoals will be labeled to indicate that they belong to the contingency in which the bomb is in package1. The effect *bomb in package1*, the action *dunk package1*, and all effects of the action *dunk package1* will be labeled to indicate that they cannot play a role in the contingency in which the bomb is in package2.[8]

[7] Note that the contingency is described in this way for clarity of exposition. The actual label is constructed as described above.

[8] Notice that the action *move package1*, although it plays a role in the contingency plan when the bomb is in package1, does not in fact depend upon the bomb being in package1. It could in principle be made part of the plan for disarming the bomb in the contingency in which the bomb is in package2, were it to prove useful for anything.

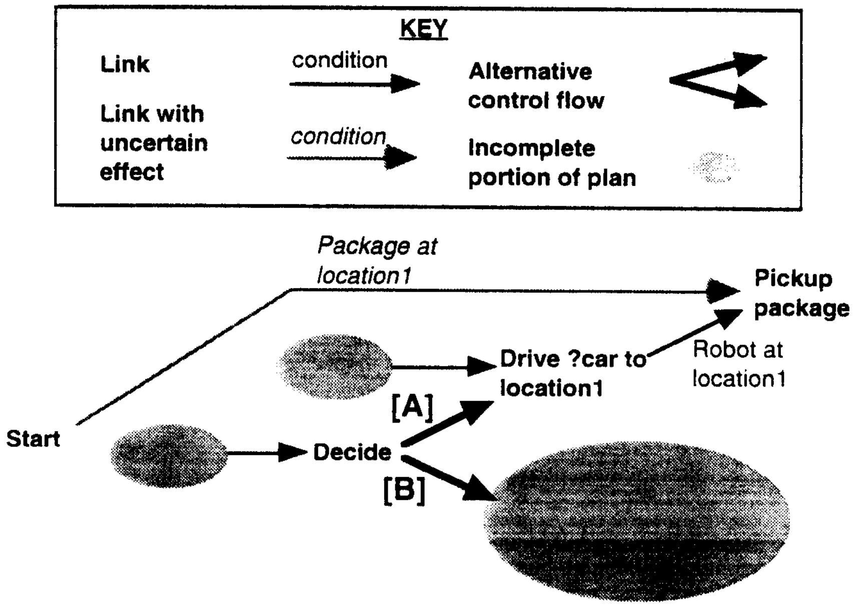


Figure 2.8 A partial plan to pick up a package

When the planner attempts to achieve the new open condition *disarm bomb*, it may choose the dunk operator once again (notice that it is prohibited from using the effect of the existing dunk operator). This instance of the dunk operator in turn gives rise to a subgoal to have the bomb be in the package that is dunked. This can only be achieved by identification with the effect *bomb in package2*. The plan thus constructed is depicted in figure 2.7 (the decision step has been omitted for clarity).

A plan may involve two or more sources of uncertainty. When this happens, Cassandra's algorithm composes the various contingency plans created in a straightforward manner. For example, suppose the agent is given the goal of picking up a package that is at one of two locations, and that one of two cars will be available for it to use. Assume the planner first encounters the uncertainty about the location of the package. This will result in the construction of a plan with two contingencies. Call these contingencies A and B (see figure 2.8).

At some point during the construction of the plan for contingency A, the planner will encounter the uncertainty concerning which car will be available, and will make the current plan dependent upon one particular outcome of that uncertainty. Since the new source of uncertainty arises in the context of planning for contingency A, Cassandra must in effect bifurcate contingency A into contingency A1 (the package is at location 1 and car 1 is available) and contingency A2 (the package is at location 1 and car 2 is available). The planner must replace all existing contingency A labels with contingency A1 labels. It must then introduce a new copy of the top-level goal labeled with contingency A2 (see figure 2.9).

It is tempting to conclude that the planner should copy the original contingency A plan for use in contingency A2. However, doing so would result in a loss of completeness, since it is possible to construct plans in which very different methods are used to achieve the goal in A1 and A2. In fact, it could be necessary. For example, extreme differences between the two cars might affect the routes on which they could be driven or the places in which they could be parked.

Notice that the uncertainty concerning the availability of the cars does not necessarily affect contingency B. If the package were close enough that the agent could get there without using a car, for

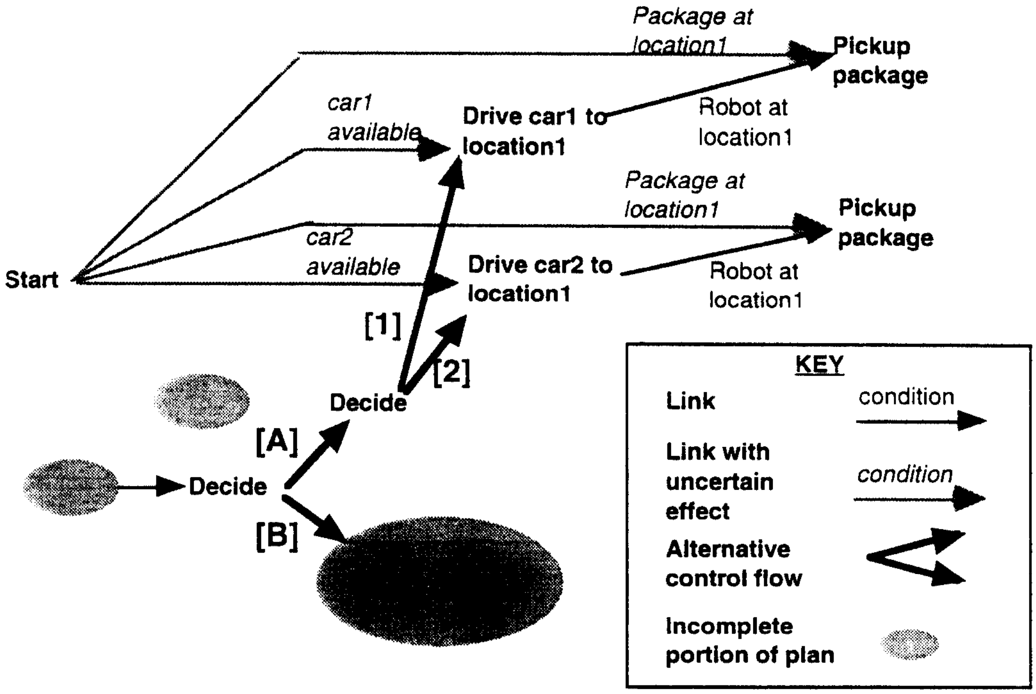


Figure 2.9 A plan with two sources of uncertainty

example, the final plan might have only three contingencies: location 1 with car 1, location 1 with car 2, and location 2 (on foot). However, if the agent must drive to location 2 as well, the planner will eventually encounter the uncertainty over the availability of cars once again while planning for contingency B.

Although it has previously encountered this same source of uncertainty, the planner must recognize that this occurred in the context of a different contingency, and bifurcate contingency B in exactly the same way as was done previously for contingency A. In general, for a pair of uncertainties, if one uncertainty arises in every contingency of the other, the planner will generate one composite contingency for every member of the cross product of the possible outcomes of the uncertainties. The combinatorics of this approach are extremely unwieldy, and some method of avoiding the need to be exhaustive would be desirable. This could in theory be accomplished by avoiding the duplication of subplans that this may involve by re-merging identical plan branches. Unfortunately there is apparently no method of doing this that does not complicate the planning process considerably. This and related points are discussed in more detail in (Pryor and Collins 1993).

2.3.2 Decision steps

When Cassandra introduces a new contingency, it adds a decision step to the plan, which represents the act of determining which contingency plan should be carried out when the plan is executed. The decision step is added to the plan with the following ordering constraints:

- The decision step must occur after the step with which the uncertainty is associated.
- The decision step must occur before any step with a subgoal whose achievement depends on a particular outcome of the uncertainty.

Formulating decision rules

As discussed in a previous section, the action of deciding which contingency to execute can be modeled as the evaluation of a set of condition-action rules of the form:

If *condition1* then *contingency1*
If *condition2* then *contingency2*
If *condition3* then *contingency3*
....

Cassandra annotates each decision step in a plan with the set of decision rules that will be used to make that decision. These rules will be used by the agent during the execution of the plan.

In order to evaluate a decision rule, the agent must be able to determine whether the rule's antecedent holds. For each antecedent condition of a rule, therefore, the agent acquires a goal to know the current status of that condition. These goals are the preconditions of the decision step. The preconditions of a decision step become open conditions in the plan in the same way as do the preconditions of any other step.

This raises the question of how Cassandra formulates the antecedent conditions for its decision rules. Since the intended effect of evaluating the rules is to choose the contingency that is appropriate given the outcome of a particular uncertainty, the conditions are effectively meant to check for which of the possible outcomes of the uncertainty actually occurred. This simple idea is made more complicated, however, because the agent cannot directly determine the outcome of an uncertainty, but must infer it from the presence or absence of effects that depend upon that outcome. Moreover, the fact that a contingency plan assumes a particular outcome of an uncertainty means only that it cannot depend on a different outcome of that uncertainty, not that it must causally depend on the outcome that it assumes. For instance, let us reconsider the Western/Ashland example. The plan to take Ashland does not actually depend on Western being blocked; it could be executed successfully regardless of the amount of traffic on Western. In general, this means that it may not be necessary to check every outcome assumed by a given course of action.

Cassandra adopts the approach of having the agent check those effects of a given outcome of an uncertainty that are actually used to establish preconditions in the contingency associated with that outcome. In other words, it seeks only to verify that the contingency plan can, in fact, succeed.[9] The condition part of a decision rule is thus a conjunction of all the direct effects of a particular outcome that are used to establish preconditions in the contingency plan for that outcome. Morgenstern (1987) draws attention to the ability of an agent to execute a plan if it can "make sure" that all the events in the plan are executable: this is the approach that has been used in formulating Cassandra's decision rules. Steel (1993) points out that it is not always possible for the agent executing a plan to distinguish the different outcomes of an uncertainty directly.

In the bomb-in-the-toilet example, Cassandra will introduce a decision step to determine whether the bomb is in package1 or not. Since the uncertainty is in the initial conditions, the decision will be constrained to occur after the *start* step. It must also occur before either of the *dunk* actions, since these depend upon particular outcomes of the uncertainty. The *decide* step will have a precondition to know

[9] It might seem that a more straightforward approach would involve checking all the effects of a given outcome. However, effects may depend upon other conditions. For example, consider our faulty drill. If the drill runs, it can have the effect of drilling a hole, but only if a bit is installed in the drill. Thus, the fact that a particular outcome occurs does not mean that all effects dependent upon that outcome will occur. However, all effects that establish preconditions in the contingency plan for that outcome must occur (or the plan will fail for other reasons).

```
Action:              (toss-coin ?coin)

Preconditions:       (holding ?agent ?coin)

Effects:             (:when (:unknown U1 H)          ; secondary precondition
                     :effect (and  (flat ?coin)
                                   (heads ?coin)))

                     (:when (:unknown U1 T)          ; secondary precondition
                     :effect (and  (flat ?coin)
                                   (tails ?coin)))

                     (:when (:unknown U1 E)          ; secondary precondition
                     :effect (on-edge ?coin)))
```

Figure 2.10 Representing the action of tossing a coin

whether the bomb is in package1. If the planner has actions available that would allow it to determine this—X-raying the box, for example—it will fulfill this precondition with one of those actions, and decide on that basis which contingency of the plan to execute.

At the point at which Cassandra constructs a decision rule, only one precondition in the plan is known to depend upon a particular outcome of the uncertainty that gave rise to the decision, in particular, the one that led to Cassandra discovering the uncertainty in the first place. The decision-rule set that Cassandra builds thus looks like this:

If *effect1* then *contingency1*
If T then *contingency2*
If T then *contingency3*
...

Cassandra must modify this initial rule set each time an effect depending directly on the source of uncertainty is used to establish an open condition in the plan. In particular, Cassandra must determine the contingency in which that open condition resides, and conjoin the effect with the existing antecedent of the decision rule for that contingency.

Consider, for example, what happens when a coin is tossed. We might say that in theory there are three possible outcomes of this action: the coin can land flat with heads up, flat with tails up, or on its edge (figure 2.10). Suppose the planner is given a goal to have the coin be flat. This can be established by using the *flat-heads* effect of tossing it. Since this is an uncertain effect, Cassandra introduces two new contingencies into the plan, one for the outcome in which the coin lands tails up, and another for the outcome in which it lands on its edge. The introduction of these contingencies mandates the introduction of a decision step. The initial rule set looks like this:

if *(flat coin)*	then [O1: H]	rule for heads up
if T	then [O1: T]	rule for tails up
if T	then [O1: E]	rule for edge

At the same time, a new open condition *(know-if (flat coin))* is introduced as a precondition of the decision, and new goal conditions are introduced that must be achieved in contingencies [O1: T] and [O1: E].

The planner next establishes the goal condition in contingency [O1: T]. The condition cannot be established by the *flat-heads* effect, because it arises in the incompatible contingency [O1: H]. However, the *flat-tails* effect can be used. The decision rule associated with the tails up contingency is thus modified as follows:

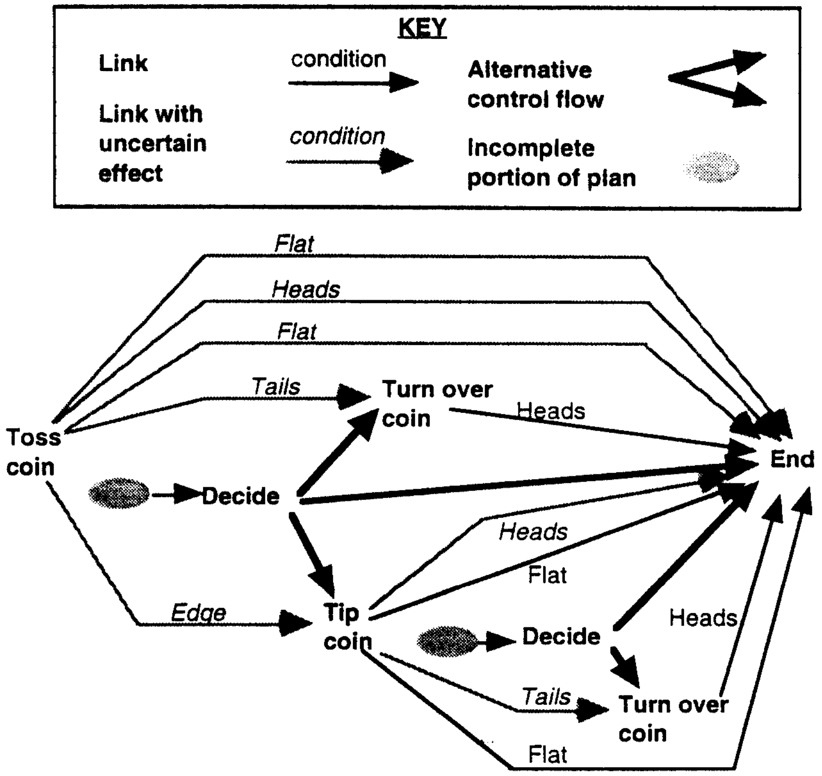


Figure 2.11 A plan with two decisions

if *(flat coin)*	then [O1: H]	rule for heads up
if *(flat coin)*	then [O1: T]	rule for tails up
if T	then [O1: E]	rule for edge

Finally, the goal condition is established in contingency [O1: E]. Neither the flat-heads nor flat-tails effects can be used, as they are known to occur in incompatible contingencies. Instead, a new step is introduced into the plan: the *tip* action is used to get the coin flat. The tip action requires the coin to be on edge, and this precondition can be established from the *on-edge* effect of the toss action. Since this effect depends directly upon the uncertainty O1, the decision rule for the edge contingency is modified as follows:

if *(flat coin)*	then [O1: H]	rule for heads up
if *(flat coin)*	then [O1: T]	rule for tails up
if *(on-edge coin)*	then [O1: E]	rule for edge

Since the plan is complete, this is the final decision rule. Notice that this rule does not discriminate the *heads up* outcome from the *tails up* outcome. Note that because the plans for these contingencies do not depend upon any uncertain effects other than the coin's being flat, the decision rule does not in fact discriminate between the outcome *heads up* and the outcome *tails up*. In fact, as discussed previously, either outcome will do, so there is no reason to make this discrimination. Which plan is executed in either of these conditions depends solely upon the order in which the agent that is executing the plan chooses to evaluate the decision rules.

A somewhat more complex problem arises if we give the planner the goal of having the coin be flat and heads up. In this case, the planner can establish both effects using the *toss* action. This will again lead to the introduction of two new contingencies into the plan, one for when the coin lands tails up, and one for when it lands on edge. Although the planner could establish *(flat coin)* in the tails up case, it would fail

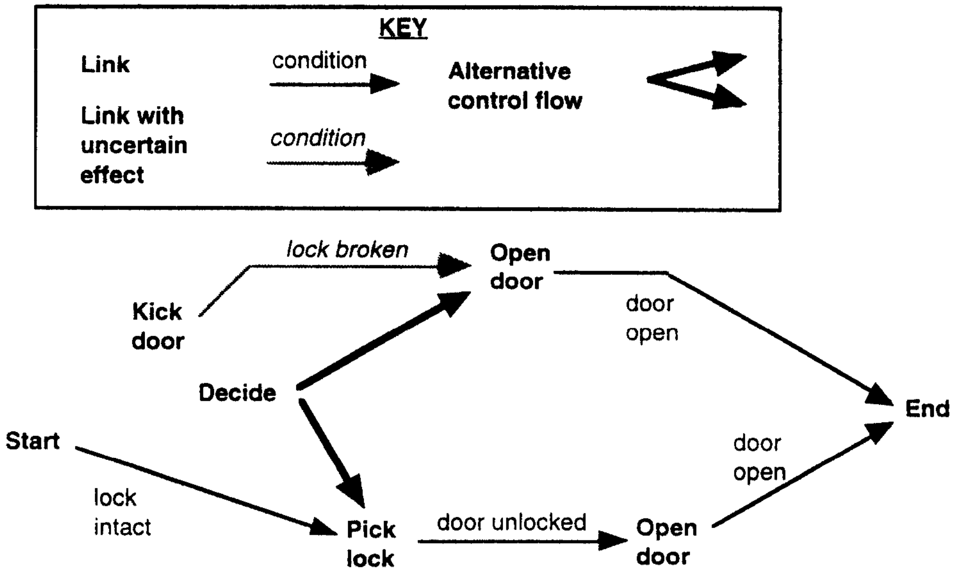


Figure 2.12 Opening a door

to complete the plan, because the coin would not be heads up. However, the planner could use the *turn over* action, which will leave the coin flat and heads up given that it was flat and tails up to begin with. At this point the decision rules are as follows:

if *(and (flat coin) (heads-up coin))*	then [O1: H]	rule for heads up
if *(and (flat coin) (tails-up coin))*	then [O1: T]	rule for tails up
if T	then [O1: E]	rule for edge

Cassandra must then plan for the goal in the outcome in which the coin lands on its edge. The planner can establish both of these effects as a result of the *tip* action. However, the result *heads up* is an uncertain effect of the tip action, since the coin might just as easily land tails up. The planner must therefore add another new contingency for when the coin lands tails up after being tipped. In this instance, the goal can be established by using the turn over action, and the tails up precondition of this action can be established by the uncertain result of the tip action. The final decision rule set for the first decision is as follows:

if *(and (flat coin) (heads-up coin))*	then [O1: H]	rule for heads up
if *(and (flat coin) (tails-up coin))*	then [O1: T]	rule for tails up
if *(on-edge coin)*	then [O1: E]	rule for edge

If the on edge contingency is pursued, the planner must make another decision, this one stemming from the uncertain result of tip. Assuming that we name the second source of uncertainty O2, the rules for this decision are:

if *(heads-up coin)*	then [O2: H]
if *(tails-up coin)*	then [O2: T]

The plan is depicted in figure 2.11.

The fact that Cassandra allows decision rules that do not fully differentiate between outcomes of an uncertainty raises a somewhat subtle issue. Consider the partial plan for opening a locked door shown in figure 2.12. The action of kicking a door has, let us say, two possible outcomes, one in which the lock is broken and one in which the agent's foot is broken. A plan for the contingency in which the lock is broken is simply to open the door. A plan for the alternative contingency is to pick the lock and then open the door. Since the second plan does not depend causally on any outcome of the uncertainty (the agent's foot does not have to be broken in order for it to pick the lock and open the door), the decision rules based on the above discussion would be:

if *(lock-broken)*	then [O: L]	rule for lock broken
if T	then [O: F]	rule for foot broken

However, notice that in this case the *pick* action depends on the lock being intact, while the *kick* action may have the effect that the lock is no longer intact. In other words, the *kick* action potentially clobbers the precondition of *pick*. However, the planner can arguably ignore this clobbering, because the two actions belong to different contingencies. This is valid, however, only if the structure of the decision rules guarantees that the agent will not choose to execute the contingency involving *pick* when the outcome of *kick* is that the lock is broken. The decision rules above clearly do not enforce this. The solution in such a case is to augment the decision rule for the contingency in which the lock is not broken to test whether the lock is in fact intact. This results in the following decision rules:

if *(lock-broken)*	then [O: L]	rule for lock broken
if *(not (lock-broken))*	then [O: F]	rule for foot broken

Cassandra augments decision rules in this way whenever it notices that a direct effect of an uncertainty could clobber a link in a different contingency.

2.4 A contingency planning algorithm

In the previous section I discussed how Cassandra handles uncertainty by introducing contingency branches and decisions into the plan. I did not, however, consider in detail the modifications of the basic planning algorithm that are necessary to build such contingency plans. In this section I describe the necessary modifications to the basic algorithm, and consider the properties of the extended algorithm.

2.4.1 The extended algorithm

The introduction of uncertainty necessitates some changes in the procedures Cassandra uses to resolve open conditions (see figure 2.13—the changes from the basic UCPOP methods that were shown in figure 2.3 are in italics). These changes incorporate the techniques of plan labeling, introducing new goals for contingencies, and modifying decisions described in previous sections.

When Cassandra uses an effect of an existing step to establish an open condition, it must check that the effect can co-occur with the condition that it is to establish. This is done by inspecting the contingency labels of the open condition and the effect that is being considered. The establishing condition must have no negative label for any contingency for which the open condition has a positive label.

There is also a new type of open condition to be dealt with—the :unknown conditions that are used to represent uncertainty. Cassandra recognizes that uncertainty is involved in a plan only when it encounters one of these special preconditions as an open condition. It then uses the procedures described in the previous section to introduce a new decision step (if the source of uncertainty that is encountered is new) or to modify an existing decision step, and to introduce new goals if necessary.

Cassandra has one decision step for each source of uncertainty. When it encounters an open :unknown condition for which there is no existing decision step it introduces a new one which it then modifies, otherwise it simply modifies the old one. The rules in the decision must reflect the preconditions in the

New step Add a new step to the plan that has an effect that will establish the open condition. Add the step preconditions and the secondary preconditions of the effect as open conditions. The open condition becomes a completed link. *Update the plan labels.*

Reuse step Make the open condition into a complete link from an effect of an existing plan step. *The effect must be possible (as indicated by its negative contingency labels) in every contingency in which the new link will be necessary (as indicated by the positive contingency labels of the open condition).* Add the secondary preconditions of the effect as open conditions. *Update the plan labels.*

Add contingency *If the open condition is of type :unknown with a new source of uncertainty, add a new decision step to the plan. Modify the appropriate rule in the decision step to include the condition established by the uncertain outcome. Add the new rule antecedent as an open condition. Add new goals for the other outcomes of the source of uncertainty as open conditions. Update the plan labels.*

Modify decision *If the open condition is of type :unknown with an existing source of uncertainty, update the decision for that source of uncertainty by modifying the appropriate rule in the decision step to include the condition established by the uncertain outcome. Add the new rule antecedent as an open condition. Update the plan labels.*

Figure 2.13 Resolving open conditions in Cassandra

plan that depend on the outcome of this source of uncertainty. A precondition comes to depend on the uncertainty through depending on the effect with the :unknown secondary precondition. Cassandra therefore inspects all the links established by the effect, and ensures that the decision includes the appropriate rules. Whenever the effect is used to establish a new link, the :unknown precondition becomes a new open condition again, and when it is resolved the condition in the new link will be included in the rules, as described earlier.

When a new antecedent is added to a rule, a new open condition is added as a precondition for the decision step. The open condition takes the form *(:know-if A)*, where A is the antecedent, and signifies that the agent must know the truth value of the antecedent.

Whenever Cassandra resolves an :unknown open condition, it also ensures that the necessary new goal conditions are introduced. All the goals that depend on the effects of that contingency are found. Any that do not mention the source of uncertainty concerned are duplicated as explained above.

Cassandra must update the plan labeling whenever a new link is introduced into the plan. The new link and the step or effect established by the new link inherit the negative contingency labels of the link's establishing effect. The new link and its establishing effect inherit the positive contingency labels of the step or effect that the link establishes. These labels must be propagated through the plan to ensure that all elements in the plan have accurate labels. New open conditions must also be labeled appropriately. They inherit the positive contingency labels of the ste There are two changes that must be made in the way the algorithm handles unsafe links. The definition of an unsafe link must be changed, and there is an additional method of resolving them. The changes are shown in italics in figures 2.14 and 2.15.

The definition of an unsafe link must be changed so as to allow for the :know-if preconditions of decision steps. A link that establishes a condition of the form *(:know-if known-condition)* is threatened by any effect that changes the status of *known-condition*.

In contingency plans, a link is threatened by an effect only when the two can co-occur. Consider, for example, a source of uncertainty S with two outcomes, T and F. If an effect has a negative label [S : T], and a link has a negative label [S : F], then there is no contingency in which the effect threatens the link.

A link L is unsafe if there is an effect E in the plan (other than the effect LE1 that establishes the condition in the link and the effect LE2 that is either established or disabled by the link) with the following three properties:

Unification One of the postconditions in E can possibly unify with the condition that L establishes or the negation of the condition that L establishes, *or if L establishes a :know-if condition. with that condition's argument or the negation of that condition's argument.*

Ordering The step that produces E can, according to the partial order, occur both before the step that produces LE2 and after the step that produces LE1.

Labeling *There is at least one contingency in which both E and L can occur.*

Figure 2.14 Unsafe links in Cassandra

As well as changing the definition of an unsafe link, the introduction of contingencies suggests a new method of resolving unsafe links: simply forbid the link and the threatening effect to co-occur. This is a simple extension of the idea of disabling a threat by using the secondary preconditions of the threatening effect. In a contingency planner, the threat can be disabled in specific contingencies in three ways: by using secondary preconditions as before, by adding new negative labels to the step that produces the effect (to signify that it must not occur in that contingency), or by adding new negative labels to the link (to signify that it cannot occur in that contingency). In the latter case, the contingency plan involving the protected link now depends on the source of uncertainty, and the decision rules must be updated appropriately.

2.4.2 Properties

Cassandra is a partial order planner directly descended from UCPOP, which is sound, complete, and systematic.

Like UCPOP, Cassandra is sound: all plans that it constructs are guaranteed to achieve their goals. If no uncertainties are involved in the plan, Cassandra is in fact equivalent to UCPOP, and therefore constructs sound plans. When a source of uncertainty is introduced into the plan Cassandra is constructing, the procedures for adding in new goals ensure that the goal is achieved in every possible outcome of the uncertainty. Cassandra is thus sound if the outcomes of every source of uncertainty are fully specified.

Cassandra is also complete: if there is a viable plan, Cassandra will find it. Again, this is a simple extension of UCPOP's completeness. If there are no uncertainties involved, Cassandra will always find a

Separation Modify the variable bindings of the plan to ensure that the threatening effect E cannot in fact unify with the threatened condition.

Ordering Modify the ordering of the steps in the plan to ensure that the step producing E occurs either before the step that produces LE1 or after the step that produces LE2.

Disable *Make sure there is no contingency in which the effect E and the link L can co-occur by performing one of:*

- *Introduce a new open condition in the plan to disable E. This new open condition is the negation of E's secondary preconditions.*
- *Add negative labels to the step producing E to forbid it to occur.*
- *Add negative labels to L to signify that it cannot occur. Update the decision rules if appropriate.*

Figure 2.15 Resolving unsafe links in Cassandra

plan in the same way as UCPOP. The introduction of a source of uncertainty into a plan leads to the addition of new contingent goals. Cassandra will find a plan for each of these new goals in the appropriate contingency. Thus, if the goal can indeed be achieved in every contingency, Cassandra will find a plan that achieves it.

UCPOP is systematic: it will never visit the same partial plan twice while searching. Cassandra, as described in this chapter, is not systematic; it may visit some partial plans in the search space more than once. Consider again the plan to disarm a bomb discussed earlier. In this plan, there are two different ways of establishing the goal to disarm the bomb: by dunking package 1, and by dunking package 2. The two methods of establishing the goal are used in two different contingencies. Cassandra can initially choose either way of establishing the goal, leading in each case to the introduction of a contingency and the necessity of replanning to achieve the goal in the other contingency. Both search paths arrive at the same final plan, so the search is not systematic.

Cassandra could be made systematic by insisting on handling the contingencies only in a certain order, the search path that uses the other order being treated as a dead end. There is currently some uncertainty as to the desirability of systematicity (Langley 1992; Peot and Smith 1992), and so this extension has not been added.

2.5 Related work

Cassandra is constructed using UCPOP (Penberthy and Weld 1992) as a platform. UCPOP is a partial order planner that handles actions that have context-dependent effects and universally quantified preconditions and effects. UCPOP is an extension of SNLP (Barrett et al. 1991; McAllester and Rosenblitt 1991; Barrett and Weld 1993) that extends the basic STRIPS representation to a subset of Pednault's ADL representation (Pednault 1988; 1989; 1991).

An early contingency planner was WARPLAN-C (Warren 1976). Contingency planning was more or less abandoned between the mid seventies and the early nineties, until SENSp (Etzioni et al. 1992) and CNLP (Peot and Smith 1992). Both SENSp and CNLP are members of the SNLP family: SENSp is, like Cassandra, based on UCPOP, and CNLP is based directly on SNLP.

Not surprisingly, Cassandra, SENSp and CNLP are in many respects very similar. All three use the basic algorithm from SNLP, and all use extended STRIPS representations. However, each modifies these basic components in a different way. In this section I briefly discuss the similarities and differences between these three systems.

2.5.1 Representation

Because SENSp and Cassandra are based on UCPOP, they both use secondary preconditions to represent context-dependent effects. CNLP does not have this extension to the basic STRIPS representation. Cassandra uses a minor modification of this extension to represent both the uncertain causal effects of actions and unknown initial conditions.

Uncertainty does not appear to be represented explicitly in SENSp. Instead, a distinction is made between ordinary variables and *run-time* variables. Run-time variables are treated as constants whose values are not yet known: in Cassandra's terms, they can be seen as variables whose values depend on uncertain outcomes. It is not clear from (Etzioni et al. 1992) whether this scheme can handle the uncertain effects of actions as well as unknown initial conditions.

In CNLP, uncertainty is represented through a combination of uncertain outcomes of actions and the effects of observing the outcome, instead of localizing the representation of uncertainty in the action that produces it, as in Cassandra's representation. A three-valued logic is used for conditions: a condition may be true, false, or unknown. For example, the action of tossing a coin would have the postcondition *unk(side-up ?x)*. Peot and Smith then introduce special *conditional* actions with an unknown precondition and several mutually exclusive sets of postconditions. In this example, the operator *observe* would have the precondition *unk(side-up ?x)* with three possible outcomes: *(side-up heads)*, *(side-up tails)*, and *(side-up edge)*.

The representation used in CNLP thus represents the uncertainty as stemming from the action that observes the result, rather than from the action that in fact produces the uncertainty. One consequence of this is that CNLP cannot use the same observation action to observe the results of different actions. For example, CNLP would require different actions to observe the results of tossing a coin (which has three possible outcomes) and tipping a coin that had landed on its edge (which has two possible outcomes). This is a serious drawback in CNLP.

2.5.2 Information acquisition

Cassandra is apparently the first planner in which decisions are represented as explicit actions in the plans that it constructs. Knowledge goals are not represented as arising specifically from the need to decide between alternative branches of the plan in either SENSp or CNLP. As a result, neither distinguishes effectively between sensing or information-gathering actions on the one hand, and decision making on the other.

Actions that achieve information goals may have preconditions in both CNLP and Cassandra. However, they may not have preconditions in SENSp: this restriction is required in order to maintain completeness. This is a serious limitation. As I argued in chapter 1, information acquisition is a planning task like any other. There is no reason to assume that information goals can always be achieved by a single action that has no preconditions.

The confusion between the source of uncertainty and the observation of uncertain results also limits the ways in which information goals can be achieved in CNLP: they must be achieved through the special observation actions that specify the uncertain outcomes. In CNLP, there is no way to distinguish the need to know whether a particular plan branch will work from the need to know the actual outcome of an uncertainty, as there is in Cassandra.

2.5.3 Plan branching

Cassandra represents plan branches through its labeling scheme, which is similar to that used by CNLP. CNLP'S scheme is somewhat simpler than Cassandra's in that it attaches labels only to plan steps. Cassandra attaches labels to effects and links as well as to steps. This is necessary in a system that uses secondary preconditions to represent context-dependent effects.

The use of labels allows Cassandra, like CNLP, to consider contingency branches in parallel rather than separately. SENSp, in contrast, constructs separate plans that each achieve the goal in a particular contingency. It then combines the separate plans at a later stage, keeping the branches totally separate.

In SENSp plan branches arise from the introduction of observation steps that bind the run-time variables. In CNLP plan branches arise from the use of an outcome of the observation step in which the uncertainty is represented. In both these planners, therefore, plan branches arise from the introduction of steps that have the effect of achieving information goals. In Cassandra, on the other hand, plan branches arise from the use of an uncertain outcome to establish a subgoal in the plan, and knowledge goals arise from the need to decide between branches. This is a fundamental difference between Cassandra and the other two planners.

The simple representation of uncertainty and the explicit representation of decisions in Cassandra thus allow a coherent approach to the problems of contingent planning. Both SENSp and CNLP are limited by their representations,[10] which both incorporate a certain degree of conceptual confusion. The representations used in these two planners limit the types of plans that they can construct: in particular, neither of them allow full planning for information goals.

2.6 Conclusions

Cassandra is a sound and complete partial-order contingency planner that can represent uncertain outcomes and construct contingency plans for those outcomes. It can construct plans in situations which have caused problems for previous systems. Cassandra is equally effective at handling actions with uncertain effects and unknown initial conditions. Cassandra explicitly plans to gather information and allows information-gathering actions to have a full range of preconditions. The coherence of its design leads to simpler plans than those produced by other planners, and provides a solid base for more advanced capabilities such as the use of varying decision-making procedures.

In the design of Cassandra, I have shown how the necessity of deferring decisions can be recognized during the plan construction process. Cassandra uses a simple representation of uncertainty to determine when a decision must be deferred due to missing information. It decides in advance on a course of action to be taken in each eventuality, and includes an explicit action in the plan to make the decision between these courses of action. In order to decide whether to take a given course of action, the agent executing the plan must determine whether the preconditions for that plan branch are met. The preconditions for the decision action are thus to have the information needed to make this determination, and Cassandra plans to achieve them in the same way as it plans to achieve other preconditions of actions.

Cassandra, like any planner within the classical planning paradigm, is severely limited in its applicability. There are two major elements of the traditional approach to planning that lead to problems: the emphasis on *completeness*, and the reliance on *prediction*. The combination leads to impractical planning systems that produce unwieldy plans.

The emphasis on completeness leads to the need for exhaustive search through the space of possible plans. Unfortunately, there is as yet no good theory of domain-independent search control and many theories of domain-dependent search control have serious drawbacks (Collins and Pryor 1992b). A

[10] See (Collins and Pryor 1993) for a discussion of other problems in the representation used by SENSp.

traditional planning system such as Cassandra must therefore rely on weak methods of search control (Cassandra uses an A*-like search with a simple syntactic measure of plan cost). Classical planning has been shown to be intractable (Chapman 1987), and without effective methods of search control there is no reason to believe that average case complexity will be significantly better than worst case complexity. My experiences with other classical planners bear this expectation out—these planners often take minutes or even hours to solve extremely simple blocks world problems (Collins and Pryor 1992a; 1992b). This problem is compounded in Cassandra by the increased complexity of the search space due to the introduction of uncertainty and the resulting branching of plans. In practice, therefore, Cassandra can only be used to construct fairly simple plans, with no more than a couple of sources of uncertainty.

The second problem with classical planning is its reliance on prediction. Cassandra, for example, produces a course of action for every predicted contingency, but its plans provide no guidance as to what to do if circumstances arise that have not been predicted. It is clearly impossible to predict every possible contingency, and in any case the search problem makes it impractical to construct plans that account for large numbers of contingencies. It can therefore never be guaranteed that an agent based on Cassandra (or any similar planner) will not encounter unexpected situations. The only recourse open to it when this happens is to replan from scratch, which, given the size and complexity of the plans required in any realistic environment, will severely compromise its ability to respond appropriately and in a timely manner. In short, the approach exemplified in the design of Cassandra is inadequate in an unpredictable world, and an alternative must be found.

The basic problems with the classical approach thus arise from the complexity of the plan construction process and its reliance on prediction, which together mean that an agent using this approach cannot respond adequately to unpredicted circumstances. What is needed is the ability to be flexible and responsive in the face of unpredictability. The agent must be able to recognize when unpredicted circumstances arise, and must be able to adapt its plans as appropriate. In particular, it must be able to respond to unexpected threats and opportunities. The agent must be able to reason about its plans and goals on the fly. Any viable alternative to the classical approach must thus address the issues that arise in considering the execution of plans. The limitation of the classical approach to the issue of plan construction is a serious drawback. Given that the unpredictability of the world means that plans cannot be comprehensive and must be subject to change during their execution, the design of an effective agent should place more emphasis on reasoning about what to do during plan execution, which is when unpredictability is encountered, and less emphasis on the pointless task of attempting to construct all-encompassing plans.

The bulk of the work described in this dissertation has therefore concentrated on how an agent can recognize and respond to unpredicted circumstances, rather than on how an agent can make predictions. In chapters 4 and 5 I describe PARETO, a plan execution system whose design takes a radically different approach from the classical planning approach followed in Cassandra. A classical planner, such as Cassandra, puts a great deal of effort into constructing large and complex plans that account for all the agent's goals in every contingency. As an agent's plans will have to be adapted or abandoned when unpredicted circumstances are encountered, much of this effort is wasted. PARETO therefore takes the opposite approach. It uses separate plans for each of its goals, and expends little effort on choosing them. PARETO then combines the plans for its various goals at execution time, changing them as appropriate when it finds itself in unexpected situations. The emphasis in PARETO is on recognizing and responding to unpredicted circumstances, rather than on prediction.

In the next chapter I address the issues involved in recognizing when unexpected circumstances should be responded to, and present a methodology based on opportunities that allows PARETO to operate effectively in an unpredictable world. Chapters 4 and 5 describe the basic operation of PARETO and the world in which it operates, and chapters 6, 7, and 8 explain in detail its mechanisms for handling unexpected situations.

CHAPTER 3

OPPORTUNITIES AND REFERENCE FEATURES

3.1 Introduction

The traditional approach to AI planning assumes that all the necessary information has been gathered before a plan is constructed, and that nothing unpredictable can happen during the execution of the plan. In this approach a single plan is constructed that when executed will achieve all the agent's goals. The traditional approach is impractical in a complex and unpredictable world, in which it may be impossible to gather all relevant information, and in which unpredicted events may occur. An agent executing plans in an unpredictable world will encounter unexpected situations, and must be able to recognize when these situations affect its goals and adapt its plans appropriately. Instead of putting all its effort into trying to predict the unpredictable, an agent should concentrate on coping with the unexpected when it occurs, as it surely will.

The inevitability of the unexpected means that any plans that are made in advance will have to be changed during execution (Alterman 1986; Firby 1987; Hammond 1989). Expending a great deal of effort on constructing elaborate and detailed plans by trying to predict exactly what will happen is therefore often unproductive. A more effective approach is to expend some effort on choosing simple plans, and to expend more effort on adapting those plans as unforeseen circumstances are encountered. This approach was used in the design of PARETO, which is described in some detail in chapter 4.

An agent that follows this approach must be able to determine when it should change the plan that is intended to achieve a given goal, and must be able to determine when it should change the goal that it is currently pursuing. Neither of these determinations is trivial. Any aspect of the world may in principle affect any of the agent's goals; a change in any aspect of the world may thus make it desirable to change plans. If the agent is to respond to its environment in a timely manner it cannot afford to make a detailed analysis of every circumstance that it encounters. In this chapter I present a solution to this problem, in the form of a filtering mechanism that an agent can use to decide when it should make a detailed analysis of its circumstances to determine whether it should change its plans.

My approach is based on the observation that the world, although unpredictable in detail, is in essence very regular (Agre 1988; Chapman 1990). Objects in the world behave consistently over time, and causal mechanisms, while varying in application, operate according to constant principles. The filtering mechanism that I have developed uses the fact that particular aspects of the world are often associated with the achievement or frustration of certain types of goals. For example, the presence of a sharp object is frequently implicated in the ability to achieve a goal of cutting something, and often frustrates a goal to maintain a smooth polish on a surface. The *sharp*ness of an object describes how that object affects goals whose achievement will be either furthered or hindered if the object cuts something else. There are many concepts such as *sharp* that describe effects on goals; they form the basis of the filtering mechanism described in this chapter.

This chapter starts by considering the problem in more detail. Section 3.2 investigates how plans are intended to achieve their goals, and shows how the same principles apply in coping with unpredictability. I conclude that the consideration of opportunities provides a mechanism that unifies plan construction and plan adaptation, and that in order to operate effectively in an unpredictable world an agent must be able to recognize opportunities. Section 3.3 describes the approaches that other researchers in the field have used, and explains why they are inadequate. Sections 3.4 and 3.5 describe the approach that I have used in the design of PARETO.

3.2 Planning for opportunities

Instead of using a single plan that would achieve all its goals in every possible situation, PARETO uses a separate plan for each of its goals, with each plan allowing for only a few possible situations. Any agent using such an approach must be able to perform the following two important functions:

- It must be able to replace the plan that it is currently using for a given goal with one that is more appropriate to the circumstances in which it finds itself.
- It must be able to switch its attention between the plans for its various goals as circumstances dictate.

In this section I describe a framework that unifies the treatment of planning and plan execution in unpredictable worlds (Birnbaum 1986; Hartman 1990). The foundation of this framework is the observation that planning in advance can be seen as a theory of predicting opportunities. This view of planning implies that if an agent can recognize unpredicted opportunities, it can then treat them as if they were expected. The major issue in coping with an unpredictable world is thus the issue of how to recognize opportunities.

3.2.1 Achieving a single goal

The ease with which a goal may be achieved and the benefits of doing so vary wildly over its lifetime. Planning in advance for a given goal can be seen as using predictions of the future values of these factors to choose an acceptable method of achieving that goal. In an unpredictable world these decisions cannot always be reliable, and must be revised if necessary.

It is usually more effective to choose to pursue a given goal when the costs of doing so are low and the benefits are high, rather than simply choosing the best way to achieve the goal at a given moment. For example, consider a goal to buy gas for your car. Suppose you notice that you are low on gas as you are on your way to an important meeting and if you stop to buy gas, you will be late. Stopping to buy gas is thus inconvenient—*i.e.*, the cost is high. The benefit—essentially the reduced probability of running out of gas—is also high, but may be outweighed by the cost. When you are in the meeting the cost goes up, since walking out of the meeting and driving off to buy gas would be a most undesirable course of action, while the benefit does not change. If you have no other urgent appointments, it becomes more convenient to buy gas once you are out of the meeting and on your way home, so the cost decreases, and the benefit increases as you drive, as the likelihood of running out of gas becomes more acute. Once you arrive home (if you have not yet refilled the tank) the cost of buying gas rises again, as it is now inconvenient to make a special journey, while the benefit is now greater as you have used more gas on your journey. The cost and benefit continue to vary over time, depending on the exact situation in which you find yourself, until you decide to achieve the goal—for instance, if the tank is very nearly empty, you see a gas station on the same side of the road that appears to have low prices and you are in no particular hurry to get anywhere.

In this example, although you could choose to pursue the goal of getting gas at any moment over the lifetime of the goal, you would be unlikely to do so when it would be very inconvenient, such as when you are in the middle of your important meeting or in the middle of the night when you are fast asleep, unless the benefit of doing so was exceptionally high. Although it would be possible to pursue the goal at those moments, and indeed there may be many different ways of achieving it, you are much more likely to wait until it becomes advantageous to achieve the goal. Because the benefit of filling the tank does not change while you are not using the car, you can be fairly sure that when you next get in the car there will be a time when the cost is lower and the benefit about the same. In short, you will wait until there is a good *opportunity* for the goal.[1]

Planning can be seen as predicting when opportunities will occur and what form they will take, and deciding in advance to take advantage of them. Although it is impractical to perform detailed predictions for all possible circumstances, it is often possible to perform simplified predictions. For example, it would be routine to plan when to refill your car with gas based on your knowledge of the amount remaining in the tank, the locations of gas stations, and your travel plans over the next day or so.

Responding to unforeseen circumstances similarly involves recognizing unpredicted opportunities. A plan that is designed to achieve a particular goal should be revised when the opportunity that was predicted does not in fact occur, or when a better opportunity arises that was not considered during the planning process. Suppose your important meeting is unexpectedly canceled, for example, and there turns out to be a gas station right next door to where the meeting was to have been. This unexpected turn of events would nullify the assumptions on which you based your decision to wait until the next day before buying gas. It is now very easy to refill your tank immediately. You should abandon your previous plan and take advantage of the unexpected opportunity that has presented itself. In short, you should respond to the unexpected opportunity by following a plan that takes advantage of it.

The consideration of opportunities thus provides a unifying framework within which to consider both the problem of planning and the problem of plan revision. An agent that can recognize and take advantage of the opportunities that it encounters can operate as effectively as one that can predict all opportunities in advance. Such an agent can respond to both expected and unexpected opportunities in the same way: by choosing plans that take advantage of them. In the case of an expected opportunity, this means continuing to follow the plan that predicted the opportunity in the first place. In the case of an unexpected opportunity, the plan that the agent is following must be revised.

[1] Though note that a really critical meeting first thing in the morning might have you getting up after all.

3.2.2 Many simultaneous goals

The consideration of opportunities also provides a framework that supports decisions about when to switch attention from one goal to another. An agent can respond flexibly to the events that it encounters in an unpredictable world if it uses separate plans for each of its goals instead of using a single monolithic plan for all its goals. However, the use of this approach means that the agent must periodically suspend many of its goals. For example, to carry out a recipe that says "soak the beans overnight," you need not sit around waiting for the beans to be ready, but can do other things in the meantime. At any one time, most of an agent's goals are suspended: only a few are being actively pursued. Any given goal may be suspended for long periods. Eventually, however, the agent must either abandon a suspended goal (if it has become irrelevant or impossible to achieve) or must actively pursue it. An agent that uses separate plans for each of its goals must decide when it should change from following the plan for one goal to following the plan for another.

The basis on which these decisions should be made can be seen by considering what would happen in the idealized case in which the agent has perfect knowledge of the world and can therefore plan ahead fully and accurately. In this case the agent can use a single global plan that accounts for all its goals under all possible circumstances. Such a plan will inevitably switch the focus of the agent's attention among its many goals. Actions that further each goal will be scheduled for the times at which they will be most advantageous, in other words for the times at which opportunities will arise. In an unpredictable world, an agent cannot always decide in advance when opportunities will arise. Instead, if it can recognize opportunities for other goals as they occur it can consider at that time whether it should change the current goal that it is pursuing.

3.2.3 Recognizing opportunities

In principle, the ability to recognize opportunities in an unpredictable world enables an agent to operate effectively by revising its existing plans and switching its attention between its various goals as circumstances dictate. To make this approach effective in practice, however, the agent must be able to recognize opportunities without performing a detailed analysis of the possible implications of every unexpected circumstance that it encounters. Performing such an analysis would prevent the agent from responding in a timely manner to the dynamism of its world.

Opportunity recognition is complex in two ways. First, there is an enormous number of elements in every situation that no agent can possibly predict (Birnbaum 1986; Brand and Birnbaum 1990). In principle, any of these elements might affect any of the agent's goals, and none can be ruled out *a priori*. For example, when you are getting gas for your car unpredicted factors might include: other cars on the road, pedestrians on the sidewalk, cars parked along the road, conversation on your car radio, trees by the road, streetlights, a gas station, cyclists, and the shops along the road. In the event, most of these factors have little effect on your goals. However, in other circumstances these very same factors might be highly relevant: for example, if you knew that the gas station was just beyond a particular shop, or you actually ran out of gas and so had to coast to the side of the road, avoiding the parked cars.

Second, there are many factors involved in achieving a goal, each of which may be affected by a situation element. Suppose, for example, that you are using an old screwdriver to achieve a goal to open a can of paint. There are many preconditions that must be true in order for the goal to be achieved: the screwdriver must be available, it must fit between the can and the lid, it must be long enough to act as a lever and strong enough to stand the strain, there must be a surface on which you can rest the can, the surface must be strong enough to support the can, rigid enough to prevent the can from moving, and the surface must be a convenient height. Any of these preconditions may be affected by unpredicted situation elements.

In order to recognize an opportunity, an agent must be able to analyze its current situation so that it can determine whether its goals are affected by unpredicted situation elements. The design of an agent that is to operate in an unpredictable world must address the problem of when the agent should perform a detailed analysis so that it can alter its plans appropriately. An agent cannot perform unlimited analysis, so must be able to concentrate its resources on those circumstances in which it will be most productive.

The solution used in the design of PARETO is a filtering mechanism that indicates those situations in which there are likely to be opportunities and that will therefore repay further analysis. This filtering mechanism is based on two key observations. First, there is a large class of opportunities that are indicated by the presence of a single critical factor. For example, an opportunity to get gas is indicated by the presence of a gas station; an opportunity to open a can of paint by the presence of a screwdriver; an opportunity to cut a piece from a ball of string by the presence of a pair of scissors; and so on. Second, the causal properties of situation elements, and thus their effects on goals, are stable and in general predictable. Scissors tend to cut things; screwdrivers can be used to pry things open; ice is slippery; and so on. PARETO's filtering mechanism and the assumptions that lie behind it are described in sections 3.4 and 3.5.

In this section I have argued that the consideration of opportunities provides a unifying framework that supports both planning ahead and responding to unpredicted circumstances, and that an agent must be able to recognize opportunities if it is to operate effectively in an unpredictable world. An agent should revise its plans when it encounters an unexpected opportunity, but cannot afford to analyze every situation exhaustively. The centrality of opportunities to the execution of plans in an unpredictable world has received little attention by other researchers in the field, but the impracticality of exhaustive analysis to determine whether an agent's plans should be changed has long been recognized. The next section examines several approaches that have been taken to this problem, and argues that in some important respects these approaches are inadequate.

3.3 Related work

The impracticality of unlimited replanning has long been recognized as a serious problem in the design of intelligent agents. There are two aspects to the problem: the necessity of limiting the amount of reasoning that is performed, and the necessity of determining when this limited reasoning should be carried out. Traditional AI approaches have concentrated primarily on limiting the amount of reasoning that is performed, either by using quantitative approximations (Hanks 1990) or by using qualitative techniques (Wellman 1988; 1990). Work in *anytime algorithms* (Dean and Boddy 1988; Boddy and Dean 1989) has addressed the issues of designing reasoning algorithms that will produce an answer in limited time. None of this work addresses the issue of *when* this reasoning should be performed.

In Hayes-Roth's GUARDIAN system (Hayes-Roth 1990b) *global control plans* are used to direct the agent's reasoning towards important goals. These control plans are changed by global control decisions, which appear to be triggered by the receipt of sensory information. However, Hayes-Roth gives no details of how these decisions are triggered or the process by which they are made. Presumably GUARDIAN's mechanism for making these decisions involves minimal reasoning, as they appear to occur rapidly when necessary, but there is no discussion of this aspect. GUARDIAN thus limits the amount of reasoning that need be done by focusing it on a subset of the agent's goals, but there is no clear answer to the question of when such reasoning should be performed.

Maes (1991) describes a network-based architecture with parameters that adjust the speed with which the agent reacts to changes in its environment. If the environment changes slowly, the agent can perform more reasoning before responding; in very unpredictable environments, the agent must react with little or no reasoning. The parameters change only in response to the unpredictability of the environment, and are not affected by the particular situation in which the agent finds itself. The balance between acting and reasoning changes on a global basis, and no attempt is made to direct the reasoning towards specific goals.

There are two approaches that are worth considering in more detail because they specifically address the problem under consideration. First, the reactive planning paradigm takes a radical approach to the problem: reasoning is eschewed altogether. As I shall demonstrate in section 3.3.1, reactive systems lack flexibility, but some aspects of their design point the way to a more powerful approach. Second, several agent architectures have been proposed that address the problem of replanning in response to changing circumstances by replanning in parallel while the agent continues to perform actions. These approaches share some of the disadvantages of reactive planning, and fail to address fully the problem of *when* to reason (section 3.3.2).

3.3.1 Reactive planning

The problems of reasoning in response to changing circumstances are the motivation behind work in the so-called *reactive planning* paradigm. In this section I argue that reactive systems lack flexibility and that they make inefficient use of the regularities of the world.

Reactive planning systems dispense with reasoning altogether. Since reasoning is complex and time-consuming, the argument goes, it is impossible to reason in detail and still be able to react in a timely fashion—therefore it is better not to reason at all. Since plans are of no use in an unpredictable world, the argument continues, an agent should instead react directly to the situation in which it finds itself. In these reactive agent models, the way in which the agent reacts is specified by a set of *reaction rules* relating environmental information to actions. If more than one rule is applicable, an *arbitration mechanism* chooses what action should be performed.[2] Work in this paradigm includes the *subsumption architecture* of Brooks (1986), Agre and Chapman's Pengi (1987), and Schoppers's *universal plans* (Schoppers 1987).

The reaction rules of these systems are essentially the results of compiling the reasoning that would be necessary in every possible situation in which the agent will find itself (this is in fact how

[2] Different workers in the field use different terminology, and may not express the rules declaratively, but the basic idea is the same. The terminology I use here is based on that of Agre and Chapman.

Schoppers's universal plans are constructed). The regularities in the world are used during the compilation process to allow the production of a set of rules that will cover all relevant situations. Unfortunately, the way in which the rules are constructed means that they are inflexible, and the agents that use them can operate only in restricted circumstances.

The efficacy of reactive systems relies on the inclusion of reaction rules and arbitration mechanisms for every possible situation that might arise (indeed, the basis of Schoppers's work is that this is guaranteed to be the case). This is possible because all these systems operate in very simple worlds. Agre and Chapman's Pengi plays a video game, in which the only objects in the world are blocks of ice, a penguin, and killer bees. Brooks and his colleagues have built several robots, the most complex of which collects empty soda cans from around the office (Brooks 1990). All of them simplify the world through limited perception. Schoppers uses a blocks world augmented with unpredictability, so that the robot may drop blocks and blocks may move without the robot's volition.

In order to operate in more complex worlds, the decision rules and especially the arbitration mechanisms in these systems would have to be made far more complex. For example, if there were any creatures other than bees and penguins in Pengi's world, a whole new set of reaction rules would have to be designed. New behaviors in Brooks's robots require whole new layers of hard wiring. There currently appear to be no principled methods of extending these systems. Any new source of complexity must be considered on an individual basis. It is uncertain whether the existing mechanisms would scale up successfully to significantly more complex worlds.[3]

All these systems rely on the many regularities in the world. The reaction rules only work because the world is not entirely unpredictable. For example, in Pengi's world the only elements are blocks of ice, bees, and a penguin. All blocks of ice have the same form, and all bees are like all other bees. These elements interact with each other in a limited number of ways: blocks of ice can slide, they come to a halt when they collide with other blocks of ice, bees and the penguin are killed by sliding blocks of ice. Bees and the penguin can kick blocks of ice to make them move, and bees can kill the penguin. The existence, form, and function of these objects are regularities in the agent's world. In general, the rules in reactive planning systems rely on regularities of form and function of a limited range of elements, and would be impotent in the face of situations that fall outside the constraints imposed by these regularities. For example, if the world suddenly changed so that blocks of ice could move of their own volition, or could slide through other blocks of ice, Pengi's reaction rules would prove inadequate.

The problem is that the power that the world's regularity could provide is used only in the construction of the reaction rules, which takes place off-line, so to speak. Reactive systems such as those discussed have no way of making use of the regularity of the world at execution time because they do nothing at execution time except fire reaction rules. Instead of trying to use the world's regularity to enhance the agent's power to reason quickly and to the point, these systems use it to push all reasoning out of the agent's behaviors and into the agent's design. Later on in this chapter I shall show how the world's regularities can be used to help the agent determine when reasoning would be productive and to focus the reasoning that it performs.

[3] This particular argument is not applicable to Schoppers's work. However, his exhaustive approach, while principled, is likely to be totally impractical in any worlds with significant amounts of complexity.

3.3.2 Replanning in parallel

A second approach to the problems of reasoning in response to changing circumstances is to perform reasoning in parallel with action (Bresina and Drummond 1990; Ferguson 1992; Lyons and Hendriks 1992; McDermott 1992). The *execution module* of these systems is responsible for performing actions and sensing the environment. It executes plans and reacts to changes in the environment. Meanwhile, the *reasoning module* analyzes the agent's current plans and transforms them to reflect the changing circumstances in which the agent finds itself.[4] The parallelism of this approach allows the agent to respond to the environment with no delay when necessary, while being able to reason about the effects of changes. However, as we shall see in this section, these systems suffer from some of the same lack of flexibility as the reactive systems that were considered in the previous section, and they require detailed reasoning in order to determine those goals for which replanning is desirable. They therefore cannot always respond to unforeseen events in a timely manner.

In order for these systems to respond rapidly to unexpected events, the necessary reaction rules must be included in the execution module, either by being specified in the plans being executed or by being hardwired into the module's design, depending on the precise architecture that is used. Quick reactions therefore depend on the inclusion of the appropriate reaction rules. If there is no reaction rule that applies to the particular situation that is encountered, the system cannot respond without reasoning in detail about the effects of the situation on its goals. The reasoning module may take some time to arrive at a revised plan that takes the unexpected event into account, by which point the time for action might have passed. These systems thus cannot react in a timely fashion in situations that were not foreseen in the design of their reaction rules: they suffer from exactly the same problem as the reactive systems discussed in the previous section.

Even with the advantages of parallelism these systems cannot replan for all their goals in response to every unexpected event that occurs, but must focus on those goals and situations in which replanning will prove most productive. Both McDermott and Ferguson have considered this problem in some detail, and the solutions they arrived at are surprisingly similar considering the differences in the architectures of their systems.

The architecture of McDermott's system follows the outline described above. The reasoning module of McDermott's system, XFRM, constructs a global plan incorporating the individual plans for all the agent's goals. The global plan is passed to the execution module, which uses it to guide the agent's actions. Meanwhile, XFRM tries to improve the expected utility of its plan. It projects the outcome of the global plan based on the current circumstances, and uses *critics* (Sussman 1975) to spot aspects of the plan that could be improved. It then transforms the plan to address the indicated problems. Whenever XFRM thinks that it has improved the plan, it passes the new version to the execution module, which up until now has been executing the old version of the plan.

Although the architecture of Ferguson's TouringMachines is very different from that of McDermott's system (it has more in common with Brooks's (1986) subsumption architecture) it has the same overall effect. TouringMachines have three concurrently-operating control layers: a *reactive* layer, a *planning* layer, and a *modeling* layer. Each layer is independently connected to the agent's

[4] There is no standard terminology for describing these systems. The architecture of Ferguson's TouringMachines differs from this description in that each of its three layers (reaction, modeling, planning) can perform actions and receive sensory input, but the overall effect is similar to that of the more traditional architecture that I have described.

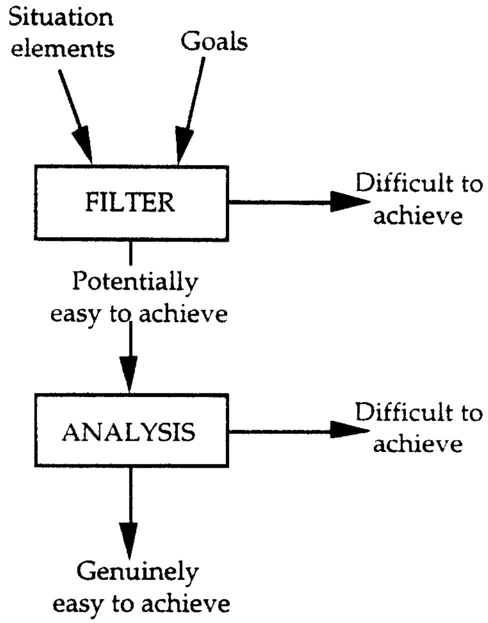


Figure 3.1 A filter for potential opportunities

sensory and effector apparatuses. The reactive layer provides fast capabilities for coping with immediate or short term events. The planning layer generates and executes plans of action to achieve the agent's long term goals. The modeling layer detects and foresees potential goal conflicts. Replanning in TouringMachines occurs in response to requests from the modeling layer, which projects the likely results of executing the current plan.

Both these systems rely on the results of projection to determine when replanning takes place and what form it takes. The projection itself generally requires a detailed analysis of the current circumstances and the future success or failure of the agent's goals. These systems thus only partially address the problem of when to reason. The reasoning required to project plans always takes place: it is only the reasoning required to replan that occurs only when it is needed.

In general, then, systems that reason in parallel with execution fail to avoid all the problems of reactive systems, and have no effective mechanisms for determining when reasoning should occur.

3.4 The critical factor hypothesis

An agent operating in an unpredictable world should be able to use the basic regularity of the world to determine how to respond to circumstances as they change, instead of simply using the regularity to add reaction rules to its plans ahead of time. It must have a quick method of recognizing those circumstances in which reasoning will prove productive. Such an agent need not specify in its plans every contingency in which unplanned reaction would be appropriate, and it need not constantly project the results of its plans in case replanning proves desirable.

None of the approaches described in the previous section is satisfactory. They all fail to make full use of the fact that the world, although unpredictable, is basically extremely regular. Objects tend to behave in certain fixed ways, external events occur in certain patterns, the agent's own goals tend to arise in certain circumstances. Instead of using this regularity to specify reaction rules ahead of time, an agent should use it to avoid relying on extensive projection to determine how to respond to circumstances as they change.

To detect opportunities, an agent must be able to pick out the few goals that are favorably affected by the current situation from the potentially enormous number of goals that remain unaffected. PARETO uses a filtering mechanism to indicate those goals that are likely to be easily achievable in any given situation, and then analyzes the indicated goals in more detail (see figure 3.1). The use of a filtering mechanism allows PARETO to consider only those goals whose investigation is likely to repay the effort. A filtering mechanism such as this will only succeed if most opportunities are predictable from a single factor. The essence of the filtering approach is simplicity; complex reasoning involving a large number of factors will not be effective.

In general, an opportunity arises as the result of the confluence of a number of complex factors. For example, many investment opportunities only exist because of the combination of many independent factors, including the geopolitical situation, trade agreements, consumer legislation, stock market conditions, the investor's tax position, and so on. The recognition of such opportunities is difficult, requiring as it does detailed analysis of all the relevant factors.

More everyday opportunities also depend on many factors. For example, suppose you have been looking for a particular book, but have been unable to find it in your local book shops. While visiting a neighboring town for some other purpose, you pass a book shop and see the book in the window. If you have time to spare, this constitutes an opportunity to buy the book. There are many aspects of the situation that help make the goal achievable, including the following: the presence of the book, the fact that you have the means of paying for the book (cash or credit card), the presence of an assistant in the shop to handle the purchase, the fact that there is a level sidewalk in front of the shop to provide easy access, the fact that the book shop door will open when you push on it, and so on.

It is notable that in the latter example there are many factors, such as the fact that you are carrying money, that contribute to the achievability of the goal but whose presence would not be enough to signal the presence of an opportunity. Indeed, most of the factors listed are of this sort. However, there is one factor, the presence of the book, that is different: the presence of the book is on its own enough to indicate an opportunity to achieve the goal. This is an example of the *critical factor hypothesis*: the presence of a single factor is often crucial for the existence of an opportunity in a given situation.

The critical factor hypothesis relies on the basic regularity of the world. Many situation elements are stable across many different situations. You usually do carry money, doors generally open, sidewalks are usually level, and so on. In fact, there is more stability at the functional level than at the purely physical: while you may not always carry much cash, you will usually have a credit card or check book with you; there are many different types of door, swing doors, revolving doors, automatic doors, and so on, but there is always a way of entering a shop (when it is open for business, at least). Given this functional stability of the world, it is usually easy to meet most of the preconditions for a given goal. The presence of situation elements that allow you to meet these preconditions therefore does not greatly affect the utility attached to the achievement of a goal; if these particular elements were not present, there would be others that would do as well. However, this is not true of all the preconditions of a goal; often, there is one precondition that is more difficult to achieve. In our example this is the availability of the book. The presence of a situation element that enables the achievement of this precondition is then sufficient to indicate the presence of an opportunity.

If the critical factor hypothesis is true, then the investment opportunity described above is an exception: there are few opportunities that depend equally on many complex factors. The hypothesis simplifies the task of recognizing opportunities enormously: the agent need only spot critical factors in order to spot opportunities. The hypothesis does appear to hold for the vast majority of opportunities, as illustrated by the following examples.

Suppose, for example, you want to pry the lid off a can of paint. You see a screwdriver which is long enough and strong enough to act as a lever, there is clear space on a table nearby, the screwdriver is within reach, there is room for you to stand by the table, and the table has a flat level surface. Most of these situation elements are insignificant in terms of the existence of an opportunity for the goal: for instance, the presence of a table is not uncommon, facilitates the achievement of many different goals and does not of itself constitute an opportunity to open the paint can. There are many different ways in which the requirement of having somewhere to rest the can while opening it could be met, and it just so happens that in this situation the presence of the table does the trick. However, the presence of the screwdriver is more significant; there are comparatively few objects that are suitable for use as a lever, and there are comparatively few goals whose achievement a screwdriver facilitates.

Similarly, consider the example presented earlier of putting gas in your car. Again, you usually have money with you; there are many turnings off your route that you could take, and many places that you could stop and get out of your car. There are, however, comparatively few places at which you can buy gas: the presence of a gas station is the critical factor in the presence of an opportunity to achieve your goal.

There are countless similar examples. When you want to cut a piece of paper, for instance, it is easy to find somewhere to place the paper while you cut it; the critical factor is a tool with which to perform the cutting. The occurrence of a bright sunny day constitutes an opportunity to have a picnic (in England at least; in sunnier climates the weather ceases to be a critical factor). A visiting exhibition provides an opportunity to see pictures that are usually in museums in distant cities. Spotting a friend across the street provides an opportunity to chat to her. The critical factor hypothesis plays a major part in the impulse buying that has such a large influence on the arrangement of goods in supermarkets; if a particular product is close at hand, the retailers reason, consumers might buy it, whereas they are unlikely to seek it out specifically.

There is thus a large class of opportunities whose existence is indicated by the presence of a single critical factor. The ability to recognize and handle these opportunities is necessary if an agent is to operate successfully in an unpredictable world.[5] A filtering mechanism based on critical factors will enable an agent to recognize these opportunities. If the agent determines that a factor that is critical for one of its goals is an element of the current situation, it can take the presence of the factor as an indication that the goal is likely to be easily achievable and so is worth analyzing in more detail. Figure 3.2 shows how such a filter might work for an agent whose goals include one to get gas, and who

[5] There may be opportunities that do not belong to this class. The mechanism I present in this dissertation cannot handle such novel opportunities.

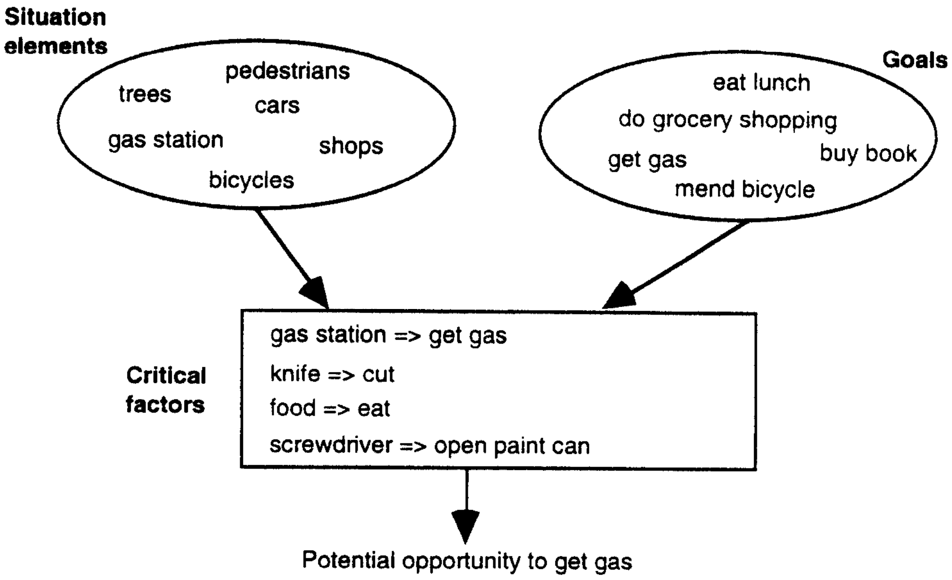


Figure 3.2 A filter based on the critical factor hypothesis

spots a gas station in its current situation. An effective filter that operates along these lines can avoid the excesses of computational intractability.

3.5 Reference features

A filtering process is only effective if it is both cheap to apply and highly predictive. A filter that is too expensive to apply will be no more efficient than performing a detailed analysis, and a filter that is inadequately predictive will cause the agent either to miss many potential opportunities or to waste cognitive resources on analyzing many goals for which there are no opportunities.[6] The critical factor hypothesis means that an agent need only recognize the presence of the critical factor for a given goal to realize that the goal in question is likely to be easy to achieve. However, the filtering process cannot consist of performing a detailed functional analysis of all the elements of the current situation to determine whether they are critical factors for any of the agent's goals. If you want to cut some string, for instance, you shouldn't consider every object in turn, wondering whether it will serve your purpose; it would be counter-productive to analyze the causal properties of the chairs and table in the room, or of the cups and glasses on the table, to determine if they can be used to cut string.

Indeed, we don't perform detailed analyses in these situations. We need not perform functional analyses because we use the fact that the causal properties of objects tend to be stable across situations. Knives and scissors tend to cut things, chairs to support things, and cups to hold liquids. Over time, the functional tendencies of objects become apparent. This is obvious: we rarely have to worry about whether a chair can be used for sitting, or whether a knife can be used for cutting. Chairs *do* support things, and knives *do* cut things—in fact, they are made for these purposes.

[6] It is in the nature of a heuristic filtering process such as the one under consideration that *some* opportunities will be missed and that *some* goals for which there are no opportunities will be analyzed.

These functional tendencies can be used as features that are both cheap to compute and highly predictive of effects on goals. The functional tendencies of objects can be labeled; for instance, objects that cut other things are labeled as being *sharp*. These labels are called *reference features*. We have many other descriptive terms like this that label functionally interesting properties of objects. Words such as *absorbent, sturdy* and *fragile* indicate that the object with the property either affects other objects in some way or is itself affected. We use these properties all the time when we are planning to achieve goals: we would use something sharp to cut another object, something absorbent to mop up a spill, and something sturdy to support a heavy object, but on the other hand would avoid using something fragile in situations in which it might break. Reference features provide the basis of the filtering mechanism used in PARETO.

3.5.1 Reference features are cheap to compute

An efficient filter must be cheap to apply: this means that it must be based on features that are either cheap to compute or computed anyway for some other purpose. Features that are cheap to compute tend to be those that are directly perceivable; for example, features such as the geometry and orientation of an object. Features that are computed in any case include the type of an object, and in some circumstances the substance of which the object is made.

Reference features are often associated directly with perceptual cues. Many of the reference features that we use are closely connected with physical features that are easy to perceive. For example, a *sharp* object has a characteristic shape with a well-defined edge between two nearly parallel planes. In such cases, the sight of an object is enough to bring the reference feature to mind. Indeed, the reference feature may be associated with objects that the agent has never previously encountered, as long as the perceptual cue is presented. For example, a novel object such as a jagged rock with the relevant shape may be perceived as *sharp* even if the agent cannot recognize the object itself.

These perceptual cues have obvious links to Gibson's theory of *affordances* (Gibson 1979). In Gibson's terms, the affordances of an environment are the functionalities an environment offers to an animal in it. For example, a horizontal, flat, extended and rigid surface affords *support*, while a vertical, flat, extended and rigid surface is a *barrier*. A pointed elongated object of moderate size and weight affords *wielding* and *piercing*, and a rigid object with a sharp edge affords *cutting*. Gibson claims that affordances are high-order optical invariants that are used directly by animals in perceiving their environments. He considers them as being relative to the animal: a surface is only a support if it is sufficiently extended and rigid to support the animal, for example.

There are two main differences between affordances and reference features. First, reference features are not limited to interactions between the agent and its environment, as appears to be the case with affordances. A surface can have the reference feature *support* even if it could not possibly bear the agent's own weight, as long as its ability to support other objects (smaller and lighter than the agent) is useful to the agent in performing its habitual tasks.

Second, reference features need not be directly linked to perceptual features: we do not have to feel or even see a knife in order to apply the term *sharp* to it, for example. The reference feature *sharp* appears to be directly linked to the *knife* concept. Moreover, there are some reference features that are not linked to perception at all, such as *brittle* and *elastic*. Reference features are part of the knowledge that we have about objects and other elements in our world.

3.5.2 Reference features are highly predictive

As well as being cheap to apply, an efficient filter must be highly predictive. The filtering process used in PARETO is based on the critical factor hypothesis—it relies on the observation that for most opportunities there is a single factor whose presence indicates the achievability of the goal in question. Most opportunities are the result of the achievement of many different preconditions, most of which are usually trivially easy to achieve, but a few of which are less commonly achieved. A critical factor for a given goal is one that achieves those preconditions that are usually difficult to achieve. For example, the critical factor in achieving the goal of cutting a piece of string is the presence of a knife; it is usually easy to find somewhere to put the string while you are cutting it.

A critical factor in the achievement of a goal is characterized in functional terms, that is, in terms of its causal effects on goals. For example, a knife is a critical factor for the goal of cutting string because it cuts things. Other situation elements that would also be critical factors for this goal are other objects that can cut things—a pair of scissors, for example. Features that are used to recognize critical factors must therefore be functional.

Reference features indicate the functional characteristics of objects, and are linked either directly to perceptual cues or to physical characterizations of objects. For example, we recognize an object such as a rubber band as being made out of rubber by its physical characteristics, and its elasticity is linked directly to its rubber composition. Reference features are close to Sloman's proposal for compact functional descriptions derived from the possibilities for motion, and constraints on motion, provided by physical descriptions of objects (Sloman 1989). However, they are more general, being concerned with any sort of relevant functionality, rather than with motion alone.

Of course, reference features do not fully describe the causal properties of objects, but are instead convenient summaries. Not all sharp objects are suitable for use in all tasks that involve cutting. We rarely use knives to cut fabric or scissors to cut meat. The concept *sharp* does not specify exactly what form of cutting takes place: scalpels make relatively clean cuts, whereas bread knives do not. Reference features form a simple classification of objects by their causal properties. This makes them eminently suitable for use in a filtering process. A more complex classification would be too elaborate and would be too time-consuming to apply.

Reference features form the basis of an effective filter for opportunities because they constitute an intermediate level of conceptualizing the world between the physical vocabulary provided by perception and the functional vocabulary required to reason about goals. They are thus both cheap to compute and highly predictive.

3.5.3 Using reference features

Reference features are labels that represent collections of related properties. Something that is *sharp*, for example, cuts softer objects, scratches harder ones, snaps taut strings, bursts taut membranes, pierces surfaces, and splits brittle objects. These are all different effects, but there are many objects in the world that share all or most of them, and it is those objects that we call sharp. An agent that has a goal the achievement of which involves any of these effects can choose something sharp with which to accomplish it; alternatively, an agent that has goal the achievement of which would be thwarted by any of these effects would probably try to avoid sharp things. The concept of *sharp*ness is intimately bound up with the effects that sharp objects have on other objects. It is useful primarily in the context of reasoning about those effects: that is, it is useful in the context of planning. It provides us with a short cut analysis of how our goals will be affected by the use of an object: any goals that require any of the effects listed above (effectively the breaking of the structural integrity of another object) may be aided by the use of something sharp, while any goals that depend on something's structural integrity may be adversely affected.

The concept *sharp* is very useful just because many commonly arising human goals involve structural integrity. If, however, we lived in a world in which structural integrity was unimportant, we might well not even have such a concept, let alone find it useful. Interestingly enough, this is the situation in some specialized domains of human experience. For example, if you are the end user of a computer the physical arrangement of files on a disk is totally unimportant to you. Whether all the information is physically connected does not affect you in the slightest. Most computer users know little and care less about the problems caused by *fragmentation*: a term in common use by operating system designers. The reference features that an agent finds useful thus depend on the tasks that it habitually performs. Agents performing different tasks in the same world may attach completely different reference features to objects in their environments. Properties that are significant to one agent may be completely meaningless to another.

Reference features, then, are used to label objects with their *functionally interesting* properties. In particular, reference features label the critical functionalities of objects—the functionalities that tend to be required by critical elements in opportunities. For example, the critical element in an opportunity to cut a piece of string must cut other objects; this functionality is indicated by the reference feature *sharp,* which indicates that the object to which it is attached may aid in the achievement of goals that require cutting, scratching, snapping, bursting, piercing, or splitting, and may pose a threat to any goals whose achievement depends on none of these effects occurring. A reference feature is associated with the types of goals that the object can help to achieve, and those that it can thwart. This provides the basis for a heuristic filtering process. If an object has a reference feature that indicates that it may aid in the achievement of one of the agent's goals, then there is potentially an opportunity to achieve the goal: it may well be easily achievable. If an object has a reference feature that indicates that it may thwart a goal, then there is probably not an opportunity to achieve that goal. The filtering process is heuristic because reference features are not infallible: not all sharp things are equally useful in all goals that involve cutting, scratching, or whatever. However, reference features provide accurate indications most of the time.

3.5.4 Generalizing reference features

In the previous section reference features were characterized as labeling functionally interesting properties of objects in the agent's world. Functionally interesting properties, and hence reference features, are not limited to physical effects. For example, a common goal is to acquire and retain wealth. Gold, jewelry, paintings and other things that serve these goals are termed *valuable.* Valuable objects are of interest not for their physical attributes per se, but simply because of the fact that they are worth a lot of money. In general, abstract attributes, such as value or beauty, may have consistent functional tendencies in the same way as more physical attributes, and reference features may tag these tendencies.

Surface interactions:	smooth slippery sticky rough gritty greasy dull shiny glossy	Load-bearing interactions:	flimsy sturdy tough heavy light soft hard solid	Containment interactions:	bulky dense tiny large small big sheetlike stringlike hollow container
Cutting interactions:	sharp blunt soft hard	Breaking:	brittle fragile robust delicate	Deformation:	rigid flexible elastic bouncy
Liquid interactions:	impermeable permeable absorbent viscous	Temperature:	hot cold frozen	Vision and light:	transparent translucent solid
Use as tools:	round (cylindrical) prying pointed graspable	Tasks and plans:	urgent fiddly robust time-consuming	Stability:	stable unstable precarious balanced lopsided
Motion:	flat barrier	Miscellaneous:	valuable worthless		

Figure 3.3 Some common reference features

Moreover, it is not only objects that can have functionally interesting properties. The weather, for example, significantly affects the tasks of many people. Roads become slippery when wet, forcing a change in driving habits. A bright sunny day provides a good opportunity for outdoor activities. Similarly, a particular route that you often take might be a traffic-free short cut on your way to work. If you are a mountain climber or a skier, different terrains allow you to use different techniques. An agent may attach reference features to any element of the environment that might affect its habitual tasks.

Tasks may have reference features as well. If the use of reference features is to form the basis of an effective heuristic filter, the agent must have a method of determining which particular functional tendencies (as indicated by the reference features of situation elements) are likely to be involved in the achievement of its tasks. It must know what functionality is required of a critical element for achieving each task. For example, a task to divide an object in two is facilitated by the functional effects of cutting or splitting, both of which are tagged by the reference feature *sharp*. Reference features can thus be used to tag the critical functional effects that facilitate particular tasks.

3.5.5 Reference features are ubiquitous

As might be expected, there are many reference features that are so useful to so many people that there are specific words to refer to them—words such as *sharp, fragile, elastic,* and so on. These are all words that refer directly to the functional characteristics of situation elements that are simple to compute.

There are other reference features for which we have no specific words; for example, thin flexible *sheetlike* things such as newspaper, old rags, or tissue paper can be used to wrap other objects, and thin flexible *stringlike* things such as cord, wire, or rope can be used to bind other objects together. There are also words that have more than one meaning; something *soft*, for example, might be easy to cut or might be unlikely to dent something else (it would usually, but not always, be both). Figure 3.3 shows a number of reference features commonly encountered in everyday life.

3.6 Summary

In this chapter I have discussed how an intelligent agent operating in an unpredictable world can handle unexpected situations. Such an agent should expend some effort on choosing simple plans for its goals, but should spend more effort on revising its plans to adapt them to the unforeseen circumstances that it is bound to encounter. A simple view of planning is that it consists of predicting when opportunities will occur, and deciding in advance to take advantage of them. Responding to unforeseen circumstances involves recognizing unpredicted opportunities and taking advantage of them. The consideration of opportunities thus provides a unifying framework for planning and plan revision. An agent should respond in the same way to unpredicted opportunities as it does to those for which it has planned.

In order to respond in a timely manner to unexpected situations an agent must be able to recognize opportunities without performing large amounts of detailed analysis. This means that it must use a heuristic filter to indicate those goals for which there are likely to be opportunities: in particular, a filtering process can be used to indicate those goals that are likely to be easily achievable.

In this chapter I introduced *reference features* as the basis of such a filter. Reference features are both functionally relevant and easily computable. They form a bridge between the physical characterization of the world in which the results of perception are represented and the functional characterization used to reason about goals and plans. Reference features are used to indicate those of the agent's goals that are likely to be easy to achieve in the current situation. The goals indicated by the filtering process can then be analyzed in more detail to determine whether there are genuine opportunities to achieve them, and, if so, whether the agent should take advantage of the opportunities.

The next two chapters describe the basic architecture of PARETO, a system I have built to demonstrate the practicality of a filtering mechanism based on reference features. PARETO recognizes and takes advantage of unforeseen opportunities. Chapter 6 describes the filtering mechanism that PARETO uses, and chapter 7 shows how PARETO also uses reference features to indicate potential threats.

CHAPTER 4

PARETO: A PLAN EXECUTION SYSTEM

4.1 System requirements

The work that I present in this dissertation is concerned with the problems faced by an agent operating in an unpredictable world. In previous chapters I have discussed how the need to cope with unexpected circumstances leads to the use of simple plans which can easily be adapted during their execution, and argued that an agent should concentrate its effort on correcting and modifying its plans as the need arises, rather than elaborating plans in great detail initially. In chapter 3 I described a unifying framework based on the recognition of opportunities that allows an agent to respond appropriately to unexpected circumstances encountered during plan execution. An agent operating in an unpredictable world must have some means of acquiring information about the actual state of the world, and a central theme of the work described in this thesis is that information acquisition is a goal-driven activity.

The investigation of these issues of opportunity recognition and information acquisition requires a plan execution system that can operate in an unpredictable world. Since the process of plan adaptation and execution is not the main focus of my work, I chose to use an existing system as a framework. Such a system, to be adequate for the task, must support the use of separate, simple plans for each of its many goals, while allowing the gathering of information to be treated as a planning task. More specifically, the system must exhibit the following properties:

- The ability to operate in an unpredictable world—one that is complex, dynamic, and uncertain.
- The ability to pursue many goals simultaneously, using a separate plan for each goal.
- The ability to support the planned acquisition of information, and allow specific information-gathering actions.
- The ability to use information that it acquires to adapt its plans to current circumstances.

There are several existing systems that meet some or all of these requirements. Hayes-Roth's GUARDIAN system (Hayes-Roth 1990a), for instance, operates in a simulated intensive care unit and acquires information about the state of the patient. Georgeff's Procedural Reasoning System (Georgeff and Ingrand 1989) has been used to control an autonomous robot, and supports both goal-directed reasoning and the ability to react rapidly to unanticipated changes in its environment.

The system from which I chose to work is Firby's RAPs[1] system (Firby 1989), which operates in a simulated robot truck delivery world. One of the motivating factors behind Firby's design was the need to allow the adaptation of plans to the circumstances in which they are executed, and the concomitant need to find out what those circumstances are. The RAPs system has the following desirable characteristics:

- It was designed to handle many simultaneous goals.

[1] Reactive Action Packages.

- The plans that it executes, called RAPs, have a well-defined syntax, and, when used with Firby's execution algorithm, clear semantics. RAPs can thus be written that enable the system to handle new tasks or that introduce new ways of handling old tasks, including tasks to acquire information.
- The plan execution algorithm has very clear decision points for deciding how to accomplish a task and which goal to pursue.

PARETO, the system that I developed using Firby's system as a framework, required that the RAPs system be enhanced to allow more deliberation to take place during plan execution. The original RAPs system was essentially a reactive system with a small, fixed set of rules for deciding how to choose the next action to be performed. It could only take advantage of opportunities that had been explicitly foreseen in the construction of its plans. In contrast, PARETO can recognize and take advantage of a large class of opportunities using a flexible mechanism based on reference features.

In the remainder of this chapter I describe the basic operation of PARETO. First, I show PARETO in action, followed by an overview of the system architecture. PARETO's method of plan execution is described in section 4.2, section 4.3 describes the plans that PARETO uses, and finally section 4.4 shows some examples of PARETO in action. Chapter 5 describes TRUCKWORLD, the simulated world in which PARETO operates, showing how it is indeed unpredictable, and investigates how PARETO acquires information. A more detailed description of PARETO can be found in appendix A and of its world in appendix B. A trace of a complete run is shown in appendix C.

4.1.1 System overview

PARETO operates a simulated robot delivery truck. In PARETO's world, there are several building sites whose workers use the truck to run errands. These errands are always requests for objects to be delivered by the truck. Some requests are very specific, such as "fetch a hammer," while others are less detailed, such as "fetch something to carry my tools in," which can be fulfilled by delivering either a tool bag or a box. As delivery requests arrive, the truck travels around the world looking for objects that will meet the requests, and delivering them to the appropriate locations.

The world consists of a number of locations linked by roads along which the truck can travel (see figure 4.1). The building sites to which the truck makes deliveries are in the three locations towards the top of the map, maple-ave, sheridan-rd, and newport-ave. In the center can be found the base, where the truck can usually find fuel, and the bank, which contains several safes and some gold bars. The other locations are where the truck can find the objects that it uses to fulfill its delivery goals. For each such object there is a location at which it can usually be found—its *source location*. The simulator replenishes the supply of each object at its source location on a random basis. PARETO's world is described in detail in chapter 5 and appendix B.

TRUCKWORLD is unpredictable (see chapter 5), and PARETO must frequently adapt its plans as it encounters opportunities and threats. PARETO has permanent goals to make sure that the truck does not run out of fuel and to keep track of its surroundings, and receives delivery goals as it goes about its business. During a typical run of the type described in this dissertation, PARETO first fuels up the truck and loads some spare fuel, and then completes between 7 and 12 deliveries. In these runs PARETO thus has up to 14 simultaneous top level goals, between which it frequently switches its attention. A trace of one such

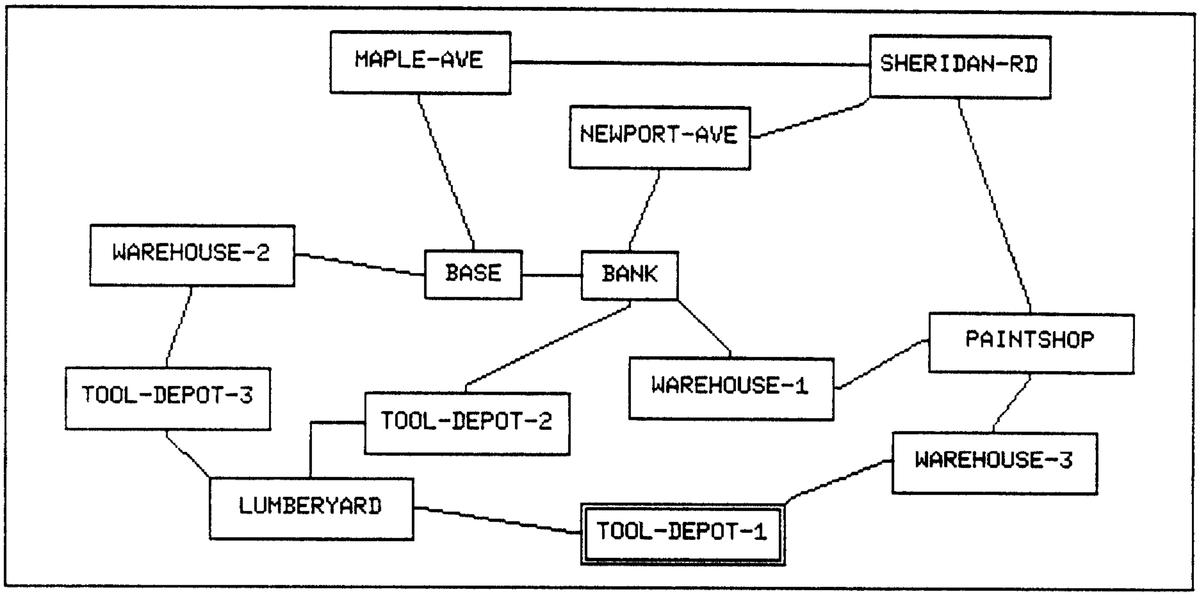


Figure 4.1 PARETO's world

run is shown in appendix C; in this run, PARETO completed 8 deliveries, and switched between top level goals about 130 times.

PARETO decides when to switch between goals by using reference features to indicate potential opportunities. It frequently takes advantage of opportunities to further goals other than the one it is actively pursuing; for example, while the truck is on its way to find one type of object, such as a bucket, it will pick up other objects that are required for delivery. In the trace shown in appendix C, PARETO starts looking for a roll of wallpaper at time 330, and doesn't find one until time 1156. Meanwhile, all the while looking for wallpaper, it has encountered, picked up, and delivered 6 other objects, and has refueled by taking advantage of a fuel drum it happened to pass.

This chapter describes the basic architecture that allows PARETO to exhibit this behavior. Chapters 6, 7 and 8 describe in more detail how PARETO uses reference features and how it decides whether to take advantage of the opportunities that it recognizes.

4.1.2 PARETO's architecture

In order to perform an action an agent must specify the details of its execution. For example, in order to grasp an object the agent must know how wide its hand must be opened and where the hand should be positioned so as to grip the required object when closed. Unfortunately, in an unpredictable world it is impossible to determine at the time at which a plan is constructed exactly what the situation will be when the actions in it come to be executed. To return to a previous example, suppose you have a goal to make a bowl of cereal. Part of this plan will include actions to take a cereal bowl out of the cupboard, which involves grasping the bowl. At the time you decide to make a bowl of cereal you cannot possibly tell exactly where the bowl will be in the cupboard and exactly where you will be standing when you try to grasp the bowl. This information about the current state of the world can only be acquired as the plan is executed.

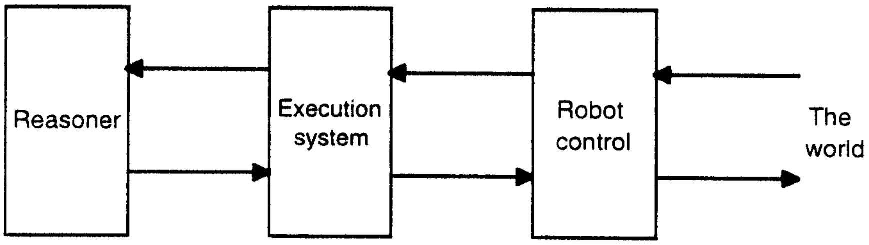


Figure 4.2 A robot control system

An agent in an unpredictable world must thus reconcile the impossibility of constructing plans that specify every detail of the actions to be performed with the need to specify the details of actions so that they may actually be executed. It is the job of the plan execution system to perform this reconciliation by turning incompletely specified plans into specific instructions that can be used to control the hardware that actually performs actions in the world. As described by Firby, the RAPs' plan execution system is intended to serve as the middle level of the three level control architecture shown in figure 4.2. The three levels of the RAPs system are:

- A *reasoner,* a deliberation layer that produces sketchy plans to achieve particular goals.
- A *plan execution* system, that fills in the details of the sketchy plans based on the situations it encounters.
- A *hardware control* system, that translates the discrete instructions from the execution system into continuous control processes on the robot hardware.

The plans executed by the RAPs system are *sketchy plans.* A sketchy plan gives only an outline of how its goals are to be accomplished, and does not specify the full details that are required in order to actually perform the necessary actions. For example, a sketchy plan to make a bowl of cereal might consist of the following steps:

- Find the cereal;
- Find the milk;
- Find a bowl;
- Pour some cereal into the bowl;
- Add milk.

Notice that this plan does not specify how to go about finding the cereal, milk, or bowl, where to put them once you have found them, how much cereal and milk to put in the bowl, and so on.[2] It is the responsibility of the plan execution system to fill in these details based on the circumstances that it encounters. There may be many different ways of accomplishing each step; for example, you might find a bowl in the cupboard, in the dishwasher, on the draining board, or in the sink. The plan execution system must choose the appropriate method.

The steps in a sketchy plan specify subgoals that the system must achieve in order to execute the plan successfully. PARETO achieves these subgoals in the same way that it achieves its top-level goals—by finding an appropriate sketchy plan. In other words, PARETO recursively expands sketchy plans by selecting methods for achieving the subgoals in the plans. Eventually, a subgoal in a sketchy plan will be achievable by a simple action, and PARETO can send the appropriate instruction to the hardware control system.

[2] The level of abstraction at which the sketchy plans should be expressed is a difficult problem in its own right, but is not one that I address in this work.

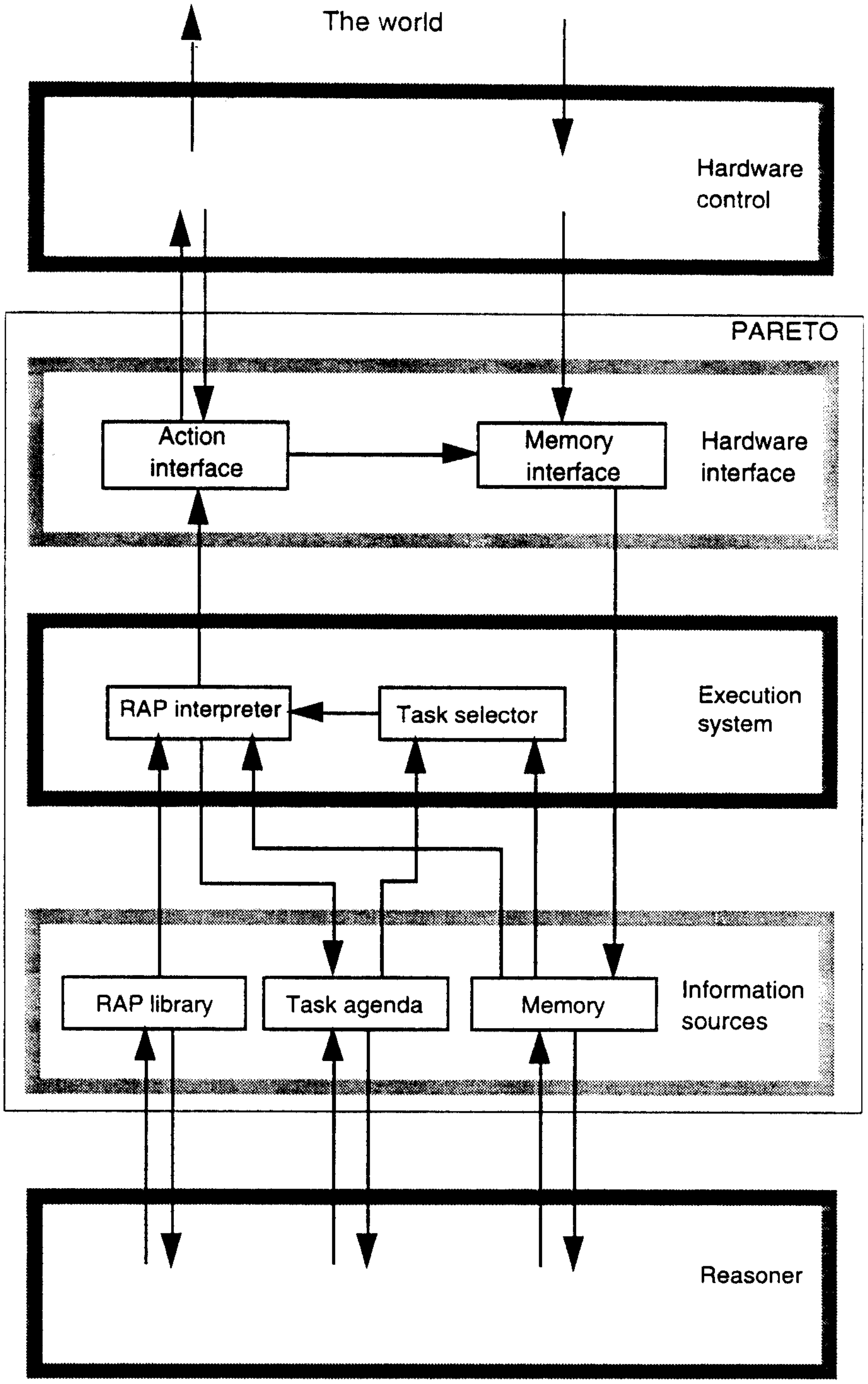


Figure 4.3 PARETO's architecture

PARETO adapts its plans to the current circumstances by choosing appropriate sketchy plans for each of its subgoals. It switches its attention from one goal to another simply by choosing to operate on a different sketchy plan. For PARETO, recognizing an opportunity for a goal is simply recognizing a good moment to execute a sketchy plan that furthers that goal.

4.2 Executing a plan

Section 4.1.2 described the three-level control architecture of which PARETO is the middle layer. This section describes the structure of PARETO in more detail, and discusses how PARETO communicates with the other layers of the architecture.

Figure 4.3 shows PARETO's architecture. The three levels described earlier, i.e. the analyzer, execution system, and hardware controller, are enclosed in dark gray boxes (■). The components through which the three levels communicate with each other are enclosed in light gray boxes (▒). PARETO is surrounded by a thin black box (|). PARETO itself consists of three major components, the execution system itself and the interfaces between the execution system and the other two levels.

4.2.1 The task agenda

When PARETO receives a goal, its first action is to place a task aimed at achieving that goal on the task agenda. When PARETO acquires a new goal, it looks in its library of *RAPs* (sketchy plans) for one that will achieve the goal. The task that is placed on the task agenda consists of the goal and the RAP that has been chosen to achieve it. PARETO's execution cycle consists of choosing a task from the agenda, processing it, and placing any new tasks that have been generated in the course of processing back on the task agenda.

The first step in the execution cycle is for the *task selector* module to choose a task from the agenda. The task selector chooses the task that will be the most productive in furthering PARETO's goals. To make this choice, it must use information obtained from the task agenda about how each task relates to the others on the agenda. For example, a task that is waiting for another task to finish will not usually be the most productive task to choose. The task selector must also use information about the current state of the world, such as the objects that are present at the truck's current location. This information is stored in PARETO's *memory*. PARETO's task selector operates by looking for opportunities to further its various goals, using a filtering mechanism based on reference features. The ability to recognize unforeseen opportunities is a significant change from the method of task selection used by Firby's RAPs system. Later chapters describe the operation of the task selector in more detail.

4.2.2 Processing a RAP

The second step in PARETO's execution cycle is the processing of the chosen task by the *RAP interpreter*. The essential function of the RAP interpreter is to fill in the details of the incompletely specified plan that describes the task. A RAP specifies all the different plans that might be used to achieve its goal. For example, a goal to find fuel might be achieved by going to the location of a fuel drum that PARETO knows about, by going to the base which is a source location for fuel, or by wandering around the world until a fuel drum is found. Each such plan is a *method* of achieving the goal. The RAP interpreter chooses one of the methods of the task that is being processed. To make this choice the RAP interpreter requires information about the current state of the world; for example, it is no use choosing the method of going to the location of a known fuel drum if PARETO doesn't know where any fuel drums are. It also requires information about the past actions of the truck, so that it can avoid methods that have proved fruitless in the past.

The processing of a task results in new goals that must be achieved. Each method in a RAP consists of a series of subgoals that must be achieved in order to achieve the RAPS' goal. By choosing a method for the task being processed, the RAP interpreter thus gives PARETO some new goals. A new task is placed on the task agenda for each of these new goals by choosing a RAP for each one.

The original task will be processed again when each of its subtasks has been completed successfully and achieved its goal. The successful execution of the subtasks is not enough to guarantee that the original task will itself have achieved its goal, because PARETO's world is dynamic and some time may pass between the execution of a task's subtasks and the repeat processing of the original task. For example, the truck might succeed in going to the location of a known fuel drum, but the drum might have meanwhile disappeared, or the fuel in it might have been used by another agent. Each RAP therefore includes a *success criterion*, specifying the goal that the task is designed to achieve, so that PARETO can determine whether the goal has in fact been achieved. When the RAP interpreter processes a task, it first checks the success criterion. If the task has succeeded, it is removed from the task agenda.

When a task on the agenda is so simple that it can be accomplished by executing a single action, processing that task consists of instructing the hardware controller to perform that action. In PARETO the *action interface* is responsible for translating the primitive actions specified in the RAPs into commands that the hardware (in this case TRUCKWORLD) can understand (Firby 1992).

4.2.3 PARETO's execution algorithm

The execution algorithm that PARETO uses thus consists of three steps:

```
Step 1.    Choose a task from the task agenda
Step 2.    If      the task has succeeded
           Then    remove it from the agenda and return to step 1
           Else    Choose one of the methods for that task
Step 3.    If      the method consists of a primitive action
           Then    execute the primitive action
           Else    place a new task on the task agenda for each subgoal in the method
           and return to step 1.
```

This algorithm allows PARETO to handle many simultaneous goals: they all become tasks on the task agenda, and any goal may be pursued according to the circumstances in force at the time.

4.2.4 PARETO's reasoner

PARETO uses a simulated hardware controller in the form of the TRUCKWORLD simulated robot delivery truck (see chapter 5). However, there is currently no analyzer in the system. In Firby's vision of this architecture, the reasoner would construct new plans (all PARETO's RAPs have been hand coded) and make strategic decisions about how to combine the plans for the various goals, what methods should be used to accomplish goals, and which goals should be pursued. The reasoner would do this through

```
(define-rap
    (index        (find-object ?criterion => ?found-place ?object))
    (succeed      (and  (my-location ?found-place)
                        (location ?object ?found-place)
                        (meets-criterion ?object ?criterion true)))
    (method
        (context  (and  (my-location ?here)
                        (location ?possible ?here)
                        (meets-criterion ?possible ?criterion unknown))
        (task-net ... steps to check that ?possible meets the criterion...))
    (method
        (context  (and  (location ?possible ?place)
                        (meets-criterion ?possible ?criterion true)))
        (task-net ... steps to travel to ?place and check  that ?possible is still there ...))
    (method
        (context  (source-for ?criterion ?place))
        (task-net ... steps to travel to ?place and see if there is a suitable object there...))
    (method
        (task-net ... steps to wander randomly until a suitable object is found...)))
```

Figure 4.4 Methods of finding an object

deliberation, using information from the RAP library, task agenda, and memory. Presumably, it would communicate with the execution system via the same modules.

Although there is no full implementation of the reasoner in the current version of PARETO, some aspects of its functionality are provided. In particular, PARETO can find routes from one location to another in its world,[3] and its plans can specify acts of inference that should be performed, just as they can specify physical actions. I mentioned above that a task may be achievable by executing a single action, in which case the RAP interpreter sends the appropriate instructions to the hardware controller. In the same way, a task may be achievable by an act of reasoning, in which case the RAP interpreter sends the appropriate request to the reasoner. Currently these requests are handled by special purpose Lisp functions.

4.3 How to read a RAP

The plans that PARETO uses are encoded in the RAP representation language. In this section I shall briefly explain the main points of this language. Appendix A provides more detail. Figure 4.4 shows one of PARETO's RAPs. It describes how the truck can find an object meeting a specific criterion.

PARETO's delivery goals are expressed in terms of *goal-criteria.* The name of the goal-criterion is used as a parameter in the indices of the RAPs that describe how to achieve the delivery goal. The other clauses in these RAPs can then use the special *meets-criterion* formula to check whether a given object meets the goal-criterion. This mechanism allows PARETO to use a single set of RAPs to describe all its delivery tasks, both those involving simple goals and those involving more complex goals, instead of requiring a separate set for each type of delivery goal.[4] Some of the complex goal-criteria used by PARETO are shown

[3] This functionality was provided in Firby's RAPs system, but was undocumented. The mechanism in PARETO is a slightly altered version of that used by Firby.

[4] The goal-criterion mechanism did not form part of Firby's RAPs system.

Goal-criterion	Conditions of satisfaction
blue-paint	(and (class ?object paint) (color ?object blue))
carry-dirt	(or (class ?object box) (class ?object bucket))
cut-twine	(or (class ?object scissors) (class ?object knife))
open-solvent	(or (class ?object scissors) (class ?object screwdriver))
red-wood	(and (class ?object wood) (color ?object red))
temp-fastener	(or (class ?object twine) (class ?object tape))
wood-5	(and (class ?object wood) (size ?object 5))

Figure 4.5 Some of PARETO's goal-criteria

in figure 4.5. There are also many simple goal-criteria, each having the class of an object as its name, and being met by any object of that class. PARETO's goal-criteria are listed in full in appendix B.

In general, a RAP includes an *index* clause, a *success* clause, and a set of *method* clauses.[5] I shall discuss the function of each of these clauses separately.

4.3.1 RAP index

A *RAP index* consists of three parts: the *name*, the *input variables*, and the *output variables*. The RAP index is the first part of the definition of a RAP, and takes the form:

```
(index     (name in-vars => out-vars ))
```

Some examples of RAP indices are:

```
(index     (deliver-object ?criterion ?site ?order-id))
(index     (find-object ?criterion => ?found-place ?object))
(index     (load-payload-object ?object))
(index     (truck-travel-to ?place))
(index     (unload-at ?site ?place ?object))
```

The name is used to retrieve the RAP from the library. The input variables of a RAP are parameters. The find-object RAP, for example, has an input variable specifying the criterion that must be met by the object to be found. The output variables become bound on the successful completion of the task. They are used to transfer information from one task to another. The find-object RAP, for example, has output variables that specify the actual object that was found and its location. These may be used by later tasks that help achieve the delivery goal: the task that loads the object into a cargo bay, for example, will need to know which object should be loaded.

4.3.2 Success clause

Each RAP has a *success clause* that specifies its goal. The format of the success clause is:

(succeed *formula*)

Some examples of success clauses are:

```
(succeed  (and  (my-location ?place)
                (location ?object ?place)
```

[5] There are other types of clauses that are not important in describing the work presented in this dissertation. See appendix A and (Firby 1989) for a full description of the RAP representation language.

```
                    (meets-criterion ?object ?criterion true)))
(succeed   (or      (location ?object bay1)
                    (location ?object bay2)))
(succeed   (and     (my-location ?here)
                    (location ?site ?here)))
```

The formula in the success clause is checked against the information in memory; if it is true, the task that the RAP described is considered to have succeeded. The success clause is checked before a task is processed. It the clause is true, the task has succeeded and is removed from the task agenda, and another task is chosen for processing. Otherwise, the task is processed in the usual way.

4.3.3 Method clause

A RAP may contain several alternative *methods* by which the task may be accomplished, each specifying a different sketchy plan that might do the job. The RAP interpreter must be able to choose a method that is appropriate for the current circumstances, so each method also specifies the conditions under which it is likely to prove effective.

A method consists of a *context* specifying the conditions under which this particular method is appropriate, and either a *primitive*, a *reasoning request*, or a *task-net* specifying what should be done.[6] A method thus takes one the following three forms:

```
(method
    (context      formula)
    (primitive    primitive-action-instruction))
(method
    (context      formula)
    (reason       reasoning-request))
(method
    (context      formula)
    (task-net     task-net-specification))
```

Method context

The *method context* is used by the RAP interpreter in deciding whether to use this particular method. The formula in the method context is checked against the information in memory; if it is true, the method may be used. This check takes place at the time that the task containing the method is processed.

Let us consider, for example, the RAP that describes how to find an object that meets a goal-criterion. It contains several different methods, as shown in figure 4.4. The first method is applicable if there is an object at the truck's current location that is not known to meet the criterion, but might do so. The method consists of checking whether the object does meet the criterion. The second method is applicable if PARETO knows of a specific object that meets the criterion elsewhere. In this case, the truck should travel to where the object is and check that it is still there. The third method is applicable if PARETO knows of a place where objects that meet the required criterion are frequently to be found. Again the truck should go there and look for a suitable object. Finally, a method that is always possible (and hence has no specific context clause) is to simply wander around until a suitable object is found.

[6] Reasoning requests are not part of Firby's RAP representation language.

The RAP interpreter decides how to accomplish a goal by choosing one of the methods in the RAP representing the plan for that goal. The information that is required in order to make the decision is specified in the context clauses of the RAP methods. When there are several methods whose contexts are valid when a task is processed, the RAP interpreter must select one of them. The way in which this is done is described in detail by Firby; briefly, the factors that may be taken into account include how successful the method has been in the past and a preference ordering on the methods that may be included in the full RAP specification (but is not described here). In a full system the reasoner might be used to determine the best method to use.

Primitive clause

Some tasks can be accomplished by executing a single action. In this case, the method contains a *primitive clause*, specifying the action that is to be executed. When the RAP interpreter encounters one of these tasks, it sends the appropriate action instruction off to the hardware. In other words, it simply goes ahead and does the task. There are also two internal operations that are represented in primitive clauses: adding an assertion to memory and erasing an assertion from memory.

Some examples of primitive clauses are:

```
(primitive  (arm-move ?arm ?place))
(primitive  (eye-scan ?place))
(primitive  (truck-turn ?dir))
(primitive  (truck-move))
(primitive  (int$assert ?clause))
```

Reasoning request

Some tasks can be accomplished through deliberation. In this case, the method contains a *reasoning request*, specifying the deliberation that is to be performed. When the RAP interpreter encounters one of these tasks, it sends the appropriate request off to the reasoner. In the current implementation of PARETO these requests are handled by special purpose Lisp functions, as there is no fully functioning reasoner.

Reasoning requests may, for example, ask the reasoner to find a plan if there is no appropriate method in the current task, analyze potential interactions between goals (see chapter 7), or infer default values for item properties.

Some examples of reasoning requests are:

```
(reason   (solve-problem ?goal))
(reason   (analyze-potential-interaction ?interaction))
(reason   (infer-default ?item ?prop-name ?prop-specs ?answer))
```

Task-net

If the task must be accomplished by performing a series of actions, the method contains a *task-net*, describing a sketchy plan for achieving the goal. The task-net contains the steps of the sketchy plan, together with some constraints on their execution. A task-net takes the form:

```
(define-rap
    (index      (deliver-object ?criterion ?site ?order-id))
    (method
      (context  (and  (location ?site ?place)
                      (not   (= ?place unknown))))
      (task-net
        (t1     (find-object ?criterion => ?found-place ?object)
                   ((and  (my-location ?found-place)
                          (location ?object ?found-place))
                      for t2))
        (t2     (load-payload-object ?object)
                   ((location ?object cargo-bay)
                      for t3 t4))
        (t3     (truck-travel-to ?place)
                   ((my-location ?place)
                      for t4))
        (t4     (unload-at ?site ?place ?object)))))
```

Figure 4.6 The deliver-object RAP

```
(task-net
    (step-1-tag   RAP-index    step-constraints)
    (step-2-tag   RAP-index    step-constraints)
    ...)
```

The step constraints specify ordering constraints on the steps in the task-net and preconditions that one step establishes for another. These constraints are used by the task selector to help it decide which task to choose for expansion. The constraints refer to the steps in the task-net by their *tags*. The *RAP-index* in a step is used to retrieve an appropriate RAP from the library when the method is expanded.

For example, there are four steps involved in delivering an object to a building site. First, PARETO must find a suitable object, then it must load it into a cargo bay. The third step is to travel to the building site, and finally the object must be unloaded. This method is shown in figure 4.6.

The first step in this task-net is to find a suitable object. This step establishes a couple of preconditions for the second step: in order to expand the second step, loading the object into the cargo bay, the truck must be at the same location as the object that was found. Step t1 must therefore be expanded before step t2, and step t2 cannot be expanded unless the truck is at the correct location. The object must remain in the cargo bay until it is unloaded: step t2 must be expanded before both t3 and t4, neither of which can be expanded if the object has been unloaded beforehand. Finally, the third step, which tells the truck to travel to the place where the delivery is to be made, must also take place before the unloading step, which must be performed at the correct location.

The RAP interpreter processes a task by choosing one of its methods, and placing a new task on the task agenda for each step in that method's task-net, or instructing the hardware to perform the method's primitive action. If the method has a task-net, the RAP-index in each step is used to retrieve an appropriate RAP from the library and to bind the appropriate variables. The step constraints for the task-net are recorded in the task agenda, and are used by the task selector when choosing the next task to process.

```
(define-rap
    (index        (open-safe ?safe))
    (Succeed      (door ?safe open))
    (method                                             ; use this method if the combination is
       (context   (and   (combination ?safe ?comb)      ; already known
                         (not (= ?comb unknown))))
       (task-net
          (t1     (toggle-thing ?safe ?comb)            ; use the combination
                     (for t2))
          (t2     (eye-examine ?safe ?front))))         ; see if the door is open
    (method                                             ; use this method if the combination
       (context   (combination ?safe unknown))          ; must be found out
       (task-net
          (t0     (find-safe-combination ?safe => ?comb)
                     (for t1))
          (t1     (toggle-thing ?safe ?comb)            ; use the combination
                     (for t2))
          (t2     (eye-examine ?safe ?front)))))        ; see if the door is open
```

Figure 4.7 Opening a safe

4.4 Examples

To make the above discussions a bit more concrete, this section presents two examples of the plan execution system in action. First, we look at how the RAP interpreter processes tasks; then we see how PARETO copes with more than one task simultaneously.

4.4.1 Processing tasks

To illustrate how the RAP interpreter processes tasks, let us consider how PARETO opens a combination safe.

A safe in PARETO's world may be unlocked by using its combination, which is the birth date of its owner. These (admittedly non-standard) safes have their owner's name written on their tops, and a big directory kept next to the safes contains everybody's birth dates. The procedure to open a safe is therefore to look at its top to find out who owns it, and then to look up the owner in the directory. PARETO can then use the owner's birth date to work out the combination and open the safe. The open-safe RAP describing this plan is shown in figure 4.7, and the find-safe-combination RAP is shown in figure 4.8. Both these RAPs are discussed in more detail in chapter 5.

How does PARETO use these RAPs? When asked to open a particular safe, a new task is placed on the task agenda. Next, assuming that PARETO has no more pressing goals, the new task is processed:

```
--> New top level goal: OPEN-SAFE
      +++  Task placed on task agenda: OPEN-SAFE
             ... Processing task: <OPEN-SAFE SAFE-1>
```

Because the combination of the safe is unknown, the second method is chosen resulting in three new tasks:

```
      +++  Task placed on task agenda: FIND-SAFE-COMBINATION
      +++  Task placed on task agenda: TOGGLE-THING
```

```
(define-rap
    (index      (find-safe-combination ?safe => ?comb))
    (succeed    (and  (combination ?safe ?comb)        ; The task succeeds if the combination
                      (not (= ?comb unknown))))        ; is known
    (method
       (task-net
          (t1     (discover-owner ?safe => ?owner)
                     (for t2))
          (t2     (discover-birthday ?owner => ?date)
                     (for t3))
          (t3     (mem-assert (combination ?safe ?date))))))
```

Figure 4.8 Finding the combination of a safe

```
+++    Task placed on task agenda: EYE-EXAMINE
```

The ordering constraints in the open-safe RAP mean that the first of these to be executed is the task to find out the safe's combination. Three additional tasks are added to the task agenda:

```
   ... Processing task: <FIND-SAFE-COMBINATION SAFE-1 => ?COMB>
+++    Task placed on task agenda: DISCOVER-OWNER
+++    Task placed on task agenda: DISCOVER-BIRTHDAY
+++    Task placed on task agenda: MEM-ASSERT
```

The task to find out who owns the safe can be accomplished by looking at the top of the safe and reading the name:

```
       ... Processing task: <DISCOVER-OWNER SAFE-1 => ?OWNER>
+++    Task placed on task agenda: EYE-EXAMINE
```

Actually examining an object requires several low-level tasks (eye-examine, eye-examine-thing, and really-eye-examine) in order to cope with the different ways in which things may be examined and any hitches in execution (such as objects having moved unbeknownst to PARETO). Eventually, the processing grounds out in a primitive action:

```
       ... Processing task: <EYE-EXAMINE SAFE-1 TOP>
+++    Task placed on task agenda: EYE-EXAMINE-THING
       ... Processing task: <EYE-EXAMINE-THING EXTERNAL SAFE-1 TOP>
+++    Task placed on task agenda: REALLY-EYE-EXAMINE
       ... Processing task: <REALLY-EYE-EXAMINE EXTERNAL SAFE-1 TOP>
           +++    Primitive hardware operation  (EYE-EXAMINE EXTERNAL SAFE-1 TOP)
                  - result:     OKAY
                  - sensor data:     SAFE SAFE-1 MAKER CHUBB
                                     SAFE SAFE-1 OWNER MARY
                                     THING SAFE-1 CONTAINER
                                     THING SAFE-1 SAFE
                                     OBJECT-SEEN EXTERNAL SAFE-1 SAFE
```

The action is performed successfully, and results in the receipt of sensory information including the information about the owner of the safe. The really-eye-examine task has no subtasks, and is not yet known to have succeeded, so it is chosen for processing again: this time, its success criterion has been met and so it is simply removed from the task agenda. Similarly, the success criteria of its parent task and the others in the hierarchy all the way up to the discover-owner task have also been met, so the tasks are also removed from the task agenda:

```
              ... Processing task: <REALLY-EYE-EXAMINE EXTERNAL SAFE-1 TOP>
              ... Task succeeded: <REALLY-EYE-EXAMINE EXTERNAL SAFE-1 TOP>
      ---     Removing task from task agenda: REALLY-EYE-EXAMINE
              ... Task succeeded: <EYE-EXAMINE-THING EXTERNAL SAFE-1 TOP>
      ---     Removing task from task agenda: EYE-EXAMINE-THING
              ... Task succeeded: <EYE-EXAMINE SAFE-1 TOP>
      ---     Removing task from task agenda: EYE-EXAMINE
              ... Task succeeded: <DISCOVER-OWNER SAFE-1 => MARY>
      ---     Removing task from task agenda: DISCOVER-OWNER
```

The remainder of the execution proceeds in the same manner. In the rest of the example, I shall omit many of the details from the program output for brevity. First, PARETO finds out Mary's birth date:

```
              ... Processing task: <DISCOVER-BIRTHDAY MARY => ?DATE>
      +++     Task placed on task agenda: EYE-EXAMINE
   ...
                  +++   Primitive hardware operation
                        (EYE-EXAMINE EXTERNAL DIRECTORY MARY)
                        - result:    OKAY
                        - sensor data:   BOOK DIRECTORY BIRTHDAY MARY COMB-12-06-59
                                         THING DIRECTORY BOOK
                                         OBJECT-SEEN EXTERNAL DIRECTORY BOOK
   ...
           ... Task succeeded: <DISCOVER-BIRTHDAY MARY => COMB-12-06-59> :: #{Goal 9}
```

Next, it stores the safe combination in memory:

```
              ... Processing task: <MEM-ASSERT (COMBINATION SAFE-1 COMB-12-06-59)>
                  +++   Primitive internal operation
                        (INT$ASSERT COMBINATION SAFE-1 COMB-12-06-59)
   ...
              ... Task succeeded: <MEM-ASSERT (COMBINATION SAFE-1 COMB-12-06-59)>
              ... Task succeeded: <FIND-SAFE-COMBINATION SAFE-1 => COMB-12-06-59>
```

The next step is to open the safe, which involves moving an arm to it and then operating the lock. It is only at this stage that PARETO decides which arm to use. Similarly, it could not have known what combination it was going to use (and hence the exact manipulation required) until the completion of the find-safe-combination task.

```
              ... Processing task: <TOGGLE-THING SAFE-1 COMB-12-06-59>
      +++     Task placed on task agenda: ARM-MOVE-TO
      +++     Task placed on task agenda: ARM-TOGGLE-THING
              ... Processing task: <ARM-MOVE-TO ARM1 SAFE-1>
   ...
                  +++   Primitive hardware operation (ARM-MOVE ARM1 EXTERNAL)
   ...
                  +++   Primitive hardware operation (ARM-MOVE ARM1 SAFE-1)
   ...
              ... Task succeeded: <ARM-MOVE-TO ARM1 SAFE-1>
   ...
                  +++   Primitive hardware operation
                        (ARM-TOGGLE ARM1 SAFE-1 COMB-12-06-59)
   ...
              ... Task succeeded: <TOGGLE-THING SAFE-1 COMB-12-06-59>
```

Finally, PARETO must check that the door really is open:

```
              ... Processing task: <EYE-EXAMINE SAFE-1 FRONT>
   ...
```

```
                  +++   Primitive hardware operation (EYE-EXAMINE EXTERNAL SAFE-1 FRONT)
                        - result:     OKAY
                        - sensor data:    SAFE SAFE-1 DOOR OPEN
       ...
              ... Task succeeded: <EYE-EXAMINE SAFE-1 FRONT>
    --> Goal accomplished: <OPEN-SAFE SAFE-1>
        ---      Removing task from task agenda: OPEN-SAFE
```

This example demonstrates how the RAP interpreter process tasks. PARETO need not decide on the details of how a task will be accomplished until forced to do so: for example, it used arm1 to open the safe, but didn't make that commitment until it had to move an arm.

4.4.2 Multiple goals

To see how PARETO handles multiple goals, we shall look at how the truck gets to the bank so that it can open the safe in the previous example. When PARETO receives the goal to go to the bank, it is at the lumberyard. It also has a goal to find a hammer, which is not currently being pursued. PARETO assigns a code number to each task, and also keeps track of the top level goal that it furthers. These code numbers are unique as long as the task has not finished, but may be reassigned on the completion of the task that they designate. In this case the traveling goal has the code 5, and is itself a top level goal:

```
    --> New top level goal: TRUCK-TRAVEL-TO :: [5:5]
```

When PARETO receives this goal it decides to pursue it, but there is a problem. Traveling to another location may interfere with other tasks that it may wish to pursue. PARETO therefore analyzes the possible interaction (see chapter 7 for details of how it recognizes potential interactions). In this case, it decides to go ahead with the traveling task (see chapter 8 for details of how it makes these decisions):[7]

```
                 +++   Primitive reasoning operation
                       <ANALYZE-POTENTIAL-INTERACTION MOVE> [5:7]
                       Analyzing MOVE interaction  for <TRUCK-TRAVEL-TO BANK> :: [5:5]
              ... Processing task: <TRUCK-TRAVEL-TO BANK> :: [5:5]
```

As it processes the truck-travel-to task and its subtasks PARETO eventually encounters the truck-travel-known-route task. The truck has not previously traveled from the lumberyard to the bank, so PARETO has not previously had to find a route between the two. It therefore posts a request to the reasoner to find a route. The route that it finds involves first traveling north to tool-depot-2 (see figure 4.1), so PARETO turns the truck to face in the correct direction, examines the road to determine the type of terrain, sets a suitable speed, and then moves the truck along the road:

```
                 +++   Primitive reasoning operation
                       <SOLVE-PROBLEM TRUCK-TRAVEL-KNOWN-ROUTE> [5:9]
                       Finding route to BANK
                 +++   Primitive hardware operation (TRUCK-TURN N)  [5:11]
                 +++   Primitive hardware operation
                       (EYE-EXAMINE LUMBERYARD ROAD-32 ALL) [5:16]
                       - result:     OKAY
                       - sensor data: ROAD-TYPE ROAD-32 STANDARD-ROAD
                 +++   Primitive hardware operation (TRUCK-SET-SPEED FAST)  [5:16]
                 +++   Primitive hardware operation (TRUCK-MOVE)  [5:15]
    *** Arrived at TOOL-DEPOT-2
```

[7] The program output in this section is edited for brevity.

The truck has now arrived at a new location. PARETO's goal to keep track of its environment is now relevant, and as this goal take priority over all others PARETO decides to pursue it (see chapter 8). It therefore scans the truck's surroundings:

```
    *** Changing goals from 5 to 0
            ... Processing task: <MONITOR-CURRENT-LOCATION> :: [0:0]
              +++   Primitive hardware operation (EYE-SCAN TOOL-DEPOT-2) [0:11]
                    - result:         OKAY
                    - sensor data:    ROAD-SEEN W ROAD-32
                                      ROAD-SEEN N ROAD-25
                                      OBJECT-SEEN EXTERNAL OBJ-20 FUEL-DRUM
                                      OBJECT-SEEN EXTERNAL OBJ-21 ROAD-SIGN
                                      OBJECT-SEEN EXTERNAL OBJ-4 HAMMER
                                      OBJECT-SEEN EXTERNAL OBJ-12 STREET-LIGHT
                                      OBJECT-SEEN EXTERNAL OBJ-5 WHEELBARROW
                                      OBJECT-SEEN EXTERNAL OBJ-11 SAW
                                      OBJECT-SEEN EXTERNAL OBJ-15 SCREWDRIVER
                                      OBJECT-SEEN EXTERNAL OBJ-2 KNIFE
                                      OBJECT-SEEN EXTERNAL OBJ-18 SAPLING
                                      OBJECT-SEEN EXTERNAL OBJ-23 SCISSORS
                                      OBJECT-SEEN EXTERNAL OBJ-16 SCISSORS
                                      OBJECT-SEEN EXTERNAL OBJ-26 CHISEL
```

Among the objects that are present is a hammer. The presence of this hammer constitutes an opportunity for PARETO to achieve its goal of finding one (see chapter 6 for a discussion of how PARETO recognizes opportunities). PARETO therefore decides to pursue this goal:

```
    *** Changing goals from 0 to 6
            ... Processing task: <FIND-OBJECT HAMMER => ?PLACE-NAME ?OBJECT> :: [6:6]
            ... Task succeeded: <FIND-OBJECT HAMMER => TOOL-DEPOT-2 ITEM-40> :: [6:6]
```

PARETO now returns to its goal of traveling to the bank. It checks that changing locations won't interfere with any other goals, turns the truck to face the correct direction, sets the speed, and moves off down the road to the bank:

```
    *** Changing goals from 6 to 5
              +++   Primitive reasoning operation
                    <ANALYZE-POTENTIAL-INTERACTION MOVE> [5:6]
                    Analyzing MOVE interaction
                    for <TRUCK-MOVE-DOWN-ONE-ROAD ROAD-25 NODE-18> :: [5:10]
              +++   Primitive hardware operation  (TRUCK-TURN N)  [5:6]
              +++   Primitive hardware operation
                    (EYE-EXAMINE TOOL-DEPOT-2 ROAD-25 ALL)  [5:14]
                    - result:         OKAY
                    - sensor data:    ROAD-TYPE ROAD-25 BUMPY
                                      ROAD-TYPE ROAD-25 WINDY
                                      ROAD-TYPE ROAD-25 WINDING-ROAD
              +++   Primitive hardware operation (TRUCK-SET-SPEED SLOW)  [5:6]
              +++   Primitive hardware operation (TRUCK-MOVE)  [5:13]
    *** Arrived at BANK
```

When the truck arrives at the bank, it first scans its surroundings so that it can keep track of its environment. PARETO then returns to its traveling goal, and finds that it has succeeded:

```
*** Changing goals from 5 to 0
        ... Processing task: <MONITOR-CURRENT-LOCATION> :: [0:0]
            +++ Primitive hardware operation  (EYE-SCAN BANK)  [0:6]
*** Changing goals from 0 to 5
        ... Processing task: <TRUCK-TRAVEL-TO BANK> :: [5:5]
        ... Task succeeded: <TRUCK-TRAVEL-TO BANK> :: [5:5]
```

This example demonstrates that PARETO handles multiple goals effectively. It interleaves steps in pursuit of its various goals according to the circumstances in which it finds itself. By doing this, it is able to behave flexibly in its unpredictable world.

4.5 Summary

In this chapter I have described the basic operation of PARETO, a plan execution system based on Firby's RAPs system. By using simple sketchy plans to achieve its goals, and recognizing and taking advantage of opportunities, PARETO can operate successfully in a complex, dynamic, and uncertain world. The next chapter describes the world in which PARETO operates and how PARETO acquires information about it.

CHAPTER 5

INFORMATION ACQUISITION IN PARETO

5.1 Introduction

One of the chief problems facing an agent operating in an unpredictable world is the need to acquire enough information about the world so that it can behave appropriately. The design of such an agent must account for when and how the agent should acquire information and how that information can be used to formulate appropriate responses to changes in the world. In chapter 1, I argued that information acquisition is a goal-driven activity, discussed the reasons motivating the acquisition of information, and described the possible methods of doing so. We saw in chapter 2 how it is possible to anticipate goals to acquire information and to plan in advance to achieve them. In this chapter I discuss in detail how information goals might be achieved. For this we need to look at the information acquisition process more closely: examining the methods of information acquisition that are available to an agent, and how they are incorporated into its plans. The other major use of information is to keep track of an unpredictable world, so that the agent can recognize and respond appropriately to threats and opportunities. In chapter 3 I discussed how the consideration of opportunities can provide a unifying framework within which to consider the problems of planning ahead and responding to a changing environment, and presented a mechanism based on reference features that will enable an agent to recognize a large and useful class of routine opportunities.

There are two reasons for an agent to acquire information during plan execution:

- So that it can recognize the need for decisions.
- So that it has the information necessary to make informed decisions.

The first type of information acquisition, checking to see whether a decision must be made, will necessarily be performed frequently and on a regular basis. This type of information acquisition is independent of the agent's other goals. An agent whose actions are driven by goals must therefore have a specific goal to acquire the necessary information, as it will not be acquired as a subgoal of any of its other goals. The second type of information acquisition is driven by the agent's need to make decisions, and is performed in response to the agent's need to decide exactly how it should achieve its goals. Both types of information acquisition are thus goal-driven activities and, like other goal-driven activities, can be planned in advance. Sections 5.4 and 5.6 discuss how information gathering activities are represented in PARETO's plans.

There are three basic methods of acquiring information: perception, inference, and memory retrieval. Chapter 1 argued that these three methods must be treated as genuine alternatives, and that all of them can be planned for in the same way as any other actions. Section 5.3 describes how these three methods are implemented in PARETO. First, however, the next section describes the ways in which PARETO's world is unpredictable, thus giving rise to the need for the acquisition of information.

5.2 PARETO's world—TRUCKWORLD

For PARETO to be useful in investigating the issues of operating in an unpredictable world, it must operate in a world in which those issues arise. As it is impractical to use the real world, PARETO operates in a simulated world. As I discussed in chapter 1, there are three important properties that make the real world unpredictable:

- The world is *complex*: the agent cannot have fully detailed knowledge of the world.
- The world is *dynamic*: changes unconnected with the agent's actions can and do occur.
- The world is *uncertain*: the agent's actions may fail to have the desired results.

The world in which PARETO operates has these properties, as described later in this section.

PARETO operates a simulated robot delivery truck, making deliveries of various objects requested by the hypothetical workers at construction sites. The simulator was built using TRUCKWORLD (Firby and Hanks 1987), which consists of a simulator platform and a set of tools for building specific simulations. In a TRUCKWORLD world, a robot delivery truck travels between locations on a network of roads, encountering and manipulating various objects as it goes. The truck is under external control—in this case, under the control of PARETO. The controller can only interact with the world through the truck. TRUCKWORLD was also used to build the simulators used by Firby for his plan execution system (Firby 1989) and by Hanks for his plan projection system (Hanks 1990).[1]

The objects in PARETO's world were built using TRUCKWORLD's tools. Most of these objects are used regularly by the construction workers whose errands the truck runs: hammers, saws, ladders, paint, and so on. There are over 30 different types of object in PARETO's world, of which 20 are used for deliveries.[2] Some objects affect each other when placed in close proximity. For example, a plank of wood whose paint is wet may smear other objects, and a roll of wallpaper may be ripped by a saw (see appendix B). At any moment, there are typically well over 100 different objects at the various locations around the world.

The TRUCKWORLD display is shown in figure 5.1. The truck that PARETO operates has two different cargo bays, and two arms that operate independently. Information about the truck is shown in an area on the left hand side of the display. The top of the truck area shows information connected with the state of the truck. From left to right, the display shows the truck's tires, how much fuel it has, the direction it is facing, the time, and its status ("happy" unless it has either had an accident or run out of fuel). The center of the truck area shows the two cargo bays and their contents. In this case, there are two fuel drums in bay1 and a hammer in bay2. The bottom of the truck area shows the two arms, with whatever they are holding. The position of the arms is shown by the position of the grippers extending from the right of each. In figure 5.1, arm1 is empty, and is positioned at the hammer which it has just loaded into bay2. Arm2 is holding a knife, which it has just picked up from the truck's surroundings.

[1] In fact, PARETO's world uses an enhanced version of TRUCKWORLD.

[2] Firby's RAPs system delivered just one type of object, different colored rocks.

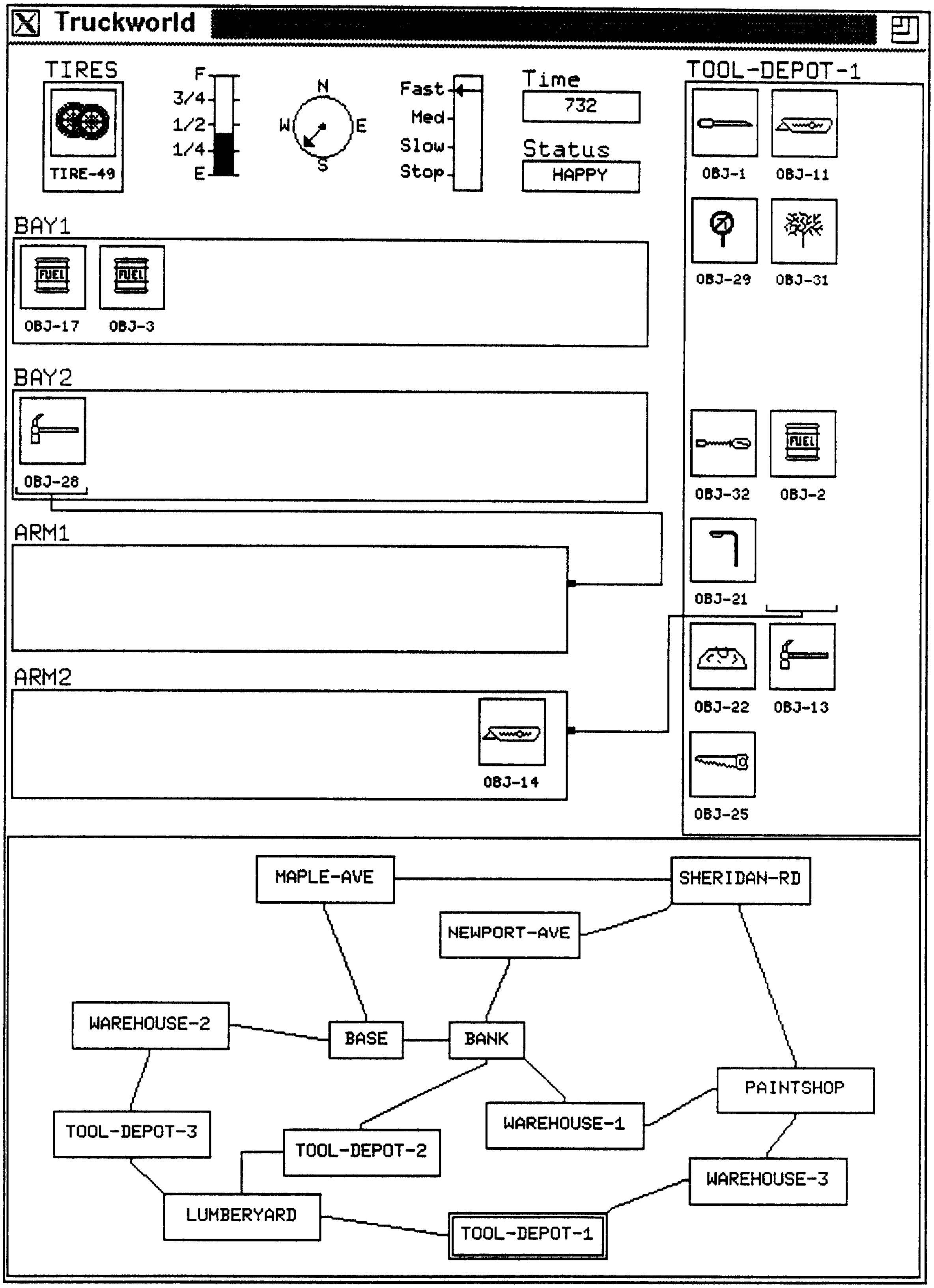


Figure 5.1 The TRUCKWORLD display

5.2.1 Complexity

The size of PARETO's world and the limited information that PARETO can obtain about it means that the world is, from PARETO's perspective, complex. At any given moment PARETO can know for certain what the situation is at only one of the many locations in the world.

Everything at the same location as the truck is assumed to be within reach, meaning that it can be picked up or examined. The truck has no access to objects at other locations, either for manipulation or observation, so PARETO's information about the current state of the world is limited. The objects at the same location as the truck are shown to the right of the TRUCKWORLD display. The truck's location (tool-depot-1 in figure 5.1) is shown at the top. In figure 5.1, the objects currently at tool-depot-1 are a chisel, a knife, a road sign, a sapling, a screwdriver, a fuel drum, a street light, a bag, a hammer, and a saw.

PARETO can affect the world only through the commands that it issues directing the truck to perform certain actions. Similarly, PARETO's only source of information regarding the state of the world comes from the feedback resulting from the execution of actions by the truck. There are two general classes of action that can be performed by the truck. Physical actions affect the state of the world; these include, for example, turning the truck to face down a road, extending an arm, or grasping an object. Sensory actions result in the receipt of information by the truck about the world state; these include scanning the truck's surroundings or examining an object. The possible actions that the truck may perform are described in appendix B.

PARETO starts off with some general knowledge about where things are likely to be found: there are usually fuel drums at the base, for example, and tool-depot-1 is a good source for hammers, knives, and other tools. It also knows the world map, so that it can travel to specific locations. However, it starts off with no specific knowledge: although buckets are often found at warehouse-1, there is not necessarily one there now. PARETO must acquire specific knowledge about the world as the truck goes about its business.

5.2.2 Dynamism

PARETO's world is *dynamic*, meaning that changes can take place in the world that are unconnected with the actions of the truck. There are five sources of dynamism in the simulated world.

First, the actions of other agents are partially simulated by randomly shuffling objects between locations in the world (Firby 1989). Every so often an object will simply disappear from one location, only to reappear later somewhere else. This means that PARETO can never rely on objects staying put for any length of time.

Second, there are changes in the world that are peculiar to specific locations. For example, empty fuel drums at the base may be refilled, and the supply of wood at the lumberyard may be replenished. The changes of this type are listed in detail in appendix A.

Third, TRUCKWORLD also simulates smaller scale changes in the location of objects. It attaches a temporary designator, or *sensor-name*, to each object encountered by the truck (Firby 1989). This sensor-name can be thought of as describing the object's detailed position with respect to the truck. PARETO must use sensor-names to refer to objects when directing the hardware to manipulate them. When the truck leaves a location, all sensor names are changed. This corresponds to the notion that the truck can keep track of objects while they are at the same location, but that they may be moved around in its absence.

Fourth, objects in the same location can interact with each other and change their physical characteristics. For example, wet paint can smear on another object, and something sharp may rip a bag with which it comes into contact. These interactions are described in detail in appendix B.

Finally, further dynamism is provided by exogenous events such as changing weather conditions and darkness. These events can affect the truck's actions: roads get muddy in the rain, for example, slowing the truck down or even causing an accident.

5.2.3 Uncertainty

Many of the truck's actions have uncertain outcomes, and may occasionally fail completely. The action of traveling along a road, for example, results in the passing of time and the consumption of fuel, but neither result is predictable even if the distance to be traveled and the speed of traveling are both known. If the truck runs out of fuel while traveling, it will not reach its intended destination but will be stranded part of the way along the road. Grasping actions, too, may fail causing the object that was to be grasped to be dropped. These failures cannot be predicted.

After executing an action, the truck receives sensory information telling it the current time, the fuel level and whether the action succeeded or, if it failed, what form the failure took. For instance, if the action succeeded the result is okay, but if the arm fails to grasp an object and drops it the result is arm-dropped.

5.3 Methods of information acquisition

An agent operating in an unpredictable world must acquire information so that it can make appropriate decisions. In acquiring the necessary information, PARETO uses each of the three methods described in chapter 1: perception, memory retrieval, and inference. The implementation of each of these methods in PARETO is described in this section.

5.3.1 Perception

PARETO makes extensive use of perception to gather information from the world. Most of this use is deliberate; TRUCKWORLD provides little information to the truck without explicit instructions to do so by way of special perceptory actions. The execution of a perceptory action by the truck results in the receipt of a number of *sensor data*. A sensor datum is a package of information from TRUCKWORLD. For example, information about the color of obj-1 would be represented as the sensor datum (color obj-1 yellow), and the information that safe-1 was seen in the truck's neighborhood would be represented as (object-seen external safe-1 safe).

PARETO has two perceptory actions: it can *scan* an area to obtain general information about the objects that are present, and it can *examine* an object to obtain more specific information about that object. PARETO's perceptory actions take exactly the same form as its physical actions such as grasping an object or setting the truck's speed. They also have preconditions like those of the physical actions. For example, the truck cannot grasp an object unless one of the arms is already positioned at the object to

```
+++ Primitive hardware operation (EYE-SCAN BANK)
     - result:     OKAY
     - sensor data:     ROAD-SEEN SE ROAD-8
                        ROAD-SEEN W ROAD-6
                        ROAD-SEEN S ROAD-9
                        ROAD-SEEN N ROAD-4
                        OBJECT-SEEN EXTERNAL SAFE-1 SAFE
                        OBJECT-SEEN EXTERNAL SAFE-2 SAFE
                        OBJECT-SEEN EXTERNAL SAFE-3 SAFE
                        OBJECT-SEEN EXTERNAL OBJ-9 SAPLING
                        OBJECT-SEEN EXTERNAL OBJ-8 GOLD-BAR
                        OBJECT-SEEN EXTERNAL OBJ-4 GOLD-BAR
                        OBJECT-SEEN EXTERNAL DIRECTORY BOOK
                        OBJECT-SEEN EXTERNAL OBJ-6 FUEL-DRUM
                        OBJECT-SEEN EXTERNAL OBJ-10 GOLD-BAR
                        OBJECT-SEEN EXTERNAL OBJ-11 ROAD-SIGN
                        OBJECT-SEEN EXTERNAL OBJ-12 GOLD-BAR
+++ Primitive hardware operation (EYE-SCAN BAY1)
     - result:     OKAY
     - sensor data:     OBJECT-SEEN BAY1 OBJ-13 FUEL-DRUM
                        OBJECT-SEEN BAY1 OBJ-14 FUEL-DRUM
```

Figure 5.2 Scanning a location

be grasped. Similarly, the truck cannot scan a location unless that location is inside the truck, or happens to be the truck's current location. The truck cannot examine an object unless that object is at the same location as the truck or is inside the truck. Moreover, some aspects of an object may not be examinable under certain conditions: the inside of a safe can only be examined, for example, if the safe's door is open.

The scanning action tells PARETO what objects are present in the area that is scanned, and (if it's the truck's neighborhood that's being scanned) what roads lead out of it. Figure 5.2 shows the results of two scanning actions. The truck is at the bank, and when it scans its neighborhood PARETO finds that four roads leave this location in various directions. In addition there are a number of objects here of various types. When the truck scans one of its cargo bays, PARETO finds that it contains two fuel drums. The information that PARETO acquires as a result of a scanning action is limited to the class of each object that is present, and the directions of the roads that are seen.

The truck can only examine those objects that would be visible if the relevant scanning action were performed. It cannot, for example, examine an object that is inside a closed safe, or one that is at another location. Examining an object allows PARETO to find out more details about it than are supplied by the scanning action. Details that might be useful include, for example, the color or size of a piece of wood, how much fuel a fuel drum contains, or the name of a safe's owner. However, examination must be focused: examining different aspects of an object provides different information about that object.[3] To determine whether a safe is open, for example, its front must be examined, while the names of its owner and maker are available from the top (see figure 5.3). PARETO must therefore ensure that the correct aspect of an object is examined when a particular type of information is required.

PARETO receives some perceptory data in the form of proprioceptive feedback. Some actions result in the automatic receipt of sensor data; for example, whenever the truck changes location a sensor

[3] This is an extension of the original TRUCKWORLD, in which examining an object always results in the receipt of all possible information about it.

```
+++ Primitive hardware operation
    (eye-examine external safe-1 front)              ; examine the front of the safe
    - result:       okay
    - sensor data:  safe safe-1 door locked
                    thing safe-1 container           ; some sensor data does not
                    thing safe-1 safe                ; depend on the aspect of the
                    object-seen external safe-1 safe ; object that is examined
+++ Primitive hardware operation
    (eye-examine external safe-1 top)                ; examine the top of the safe
    - result:       okay
    - sensor data:  safe safe-1 maker chubb          ; some sensor data is only
                    safe safe-1 owner mary           ; available from a single aspect
                    thing safe-1 container
                    thing safe-1 safe
                    object-seen external safe-1 safe
```

Figure 5.3 Examining a safe

datum is received giving details of the new location. Performing an action also gives an *action result*, such as okay or arm-cant-find. These are listed in appendix B.

5.3.2 Inference

PARETO's inference consists entirely of various types of plausible (i.e., non-deductive) reasoning. Many implicit inferences are made through property inheritance. If an object is known to be of a particular class, such as a safe, it can be assumed to have all the properties that safes usually have. For example, PARETO assumes that examining the front of a safe will inform it whether the safe is open or locked; that the safe will have a combination; that it can be opened using the combination; that it can contain other objects; and so on.

RAPs also sanction certain inferences. For example, consider the RAP for finding out a safe's combination, shown in figure 5.4. It makes use of the fact that a safe's combination is always the birth date of the owner, but there is no explicit inference step.

PARETO can also perform explicit inference. A RAP method can consist of a reasoning request, signifying that a request should be sent to the reasoning level for the appropriate reasoning to be performed. This is analogous to a primitive action, which signifies that an instruction should be sent to the hardware to perform some physical action. As PARETO lacks a full reasoning level, reasoning

```
(define-rap
    (index      (find-safe-combination ?safe => ?comb))
    (succeed    (and  (combination ?safe ?comb)        ; The task succeeds if the
                      (not (= ?comb unknown))))        ; combination is known
    (method
      (task-net
        (t1     (discover-owner ?safe => ?owner)
                   (for t2))
        (t2     (discover-birthday ?owner => ?date)
                   (for t3))
        (t3     (mem-assert (combination ?safe ?date))))))
```

Figure 5.4 Finding the combination of a safe

```
(define-rap
   (index       (examine-then-default ?item ?prop-name ?prop-specs))
   (succeed     (property-of ?item ?prop-name ?prop-specs))
   (method
      (context  (and   (examine-aspect ?prop-name ?aspect)
                       (not (examined-since-move ?item ?aspect))))
      (task-net
         (t1 (examine-for-prop ?item ?prop-name))))
   (method
      (context  (and   (examine-aspect ?prop-name ?aspect)
                       (examined-since-move ?item ?aspect)))
      (task-net
         (t1 (infer-default ?item ?prop-name ?prop-specs)))))
```

Figure 5.5   A RAP that uses inference

requests are handled by special-purpose Lisp code.

Figure 5.5 shows a RAP that makes use of inference. If examining an item fails to discover the value of one of its properties, PARETO can use default inference to obtain a value.[4] This mechanism of explicitly inferring a default value differs from that used in Firby's RAPs system, in which default inference is performed automatically whenever memory retrieval fails to find a value for a property.

Infer-default is one of PARETO's special purpose reasoning mechanisms, and uses the type of the item for which the property value is required. The defaults that PARETO uses for the objects in its world are listed in appendix B.

5.3.3 Memory retrieval

PARETO relies heavily on retrieval from memory as a method of acquiring information. It is able to do this because it has a very simple memory with an efficient retrieval system: its memory is a database of assertions and information is retrieved through matching queries directly against assertions in the database. PARETO does not use inference during memory retrieval. This means that PARETO's retrieval mechanism is efficient, but lacks flexibility.

Storage of information into PARETO's memory is mostly automatic. When PARETO receives a sensor datum, it automatically stores the information in its memory by adding an assertion to the database. PARETO can also store information explicitly.[5] We have seen that it has primitive sensing actions such as eye-examine and eye-scan. It also has primitive cognitive actions, mem-assert and mem-erase. The mem-assert action simply asserts a proposition, for example:

```
(mem-assert (combination safe-1 comb-12-06-59))
```

The mem-erase action removes an assertion from memory.

The structure of PARETO's memory, and the retrieval and storage mechanisms are based on those of Firby's RAPs system. They are described in detail in (Firby 1989). The memory mechanisms described in this chapter are the same as those used in the RAPs system except where noted otherwise.

[4] See appendix A. The syntax of item properties is described in detail by Firby.
[5] This extension to the RAPs system was suggested by Firby.

```
(define-rap
   (index        (open-safe ?safe))
   (Succeed      (door ?safe open))
   (method                                        ; use this method if the combination is
      (context  (and  (combination ?safe ?comb); already known
                      (not (= ?comb unknown))))
      (task-net
         (t1    (toggle-thing ?safe ?comb)        ; use the combination
                   (for t2))
         (t2    (eye-examine ?safe ?front))))     ; see if the door is open
   (method                                        ; use this method if the combination
      (context  (combination ?safe unknown))      ; must be found out
      (task-net
         (t0    (find-safe-combination ?safe => ?comb)
                   (for t1))
         (t1    (toggle-thing ?safe ?comb)        ; use the combination
                   (for t2))
         (t2    (eye-examine ?safe ?front)))))    ; see if the door is open
```

Figure 5.6 Opening a safe

5.4 Information acquisition in PARETO's plans

An agent in an unpredictable world cannot make all the necessary decisions about how to achieve its goals in advance. Some of these decisions must be deferred until execution time, when the information that is needed to make them will be available. An agent's plans can thus provide for acquiring the information that will support the making of these deferred decisions. All PARETO's plans provide for information acquisition in support of such decisions, primarily in the form of the memory retrieval required to check the *success* clauses, which must be matched against the information in memory. Of course, memory retrieval can only be used if the relevant information has already been acquired and stored; many RAPs therefore include explicit steps to acquire the information that will confirm or deny the success of the task, or allow the interpreter to choose a method by which to accomplish it.

Consider, for example, the RAP to open a safe, part of which is shown in figure 5.6. The task succeeds if the safe's door is open. After performing the action that should unlock the safe, it checks to see that the door really is open by examining the front of the safe.

The RAP for loading a payload object is shown in figure 5.7. The truck's cargo bays cannot contain more than eight objects. When loading an object, PARETO must therefore take the number of objects already in each bay into account when deciding which bay should be used. The RAP shown in the figure has four methods. The first one is applicable if bay 2 has not been scanned recently, thus updating the number of objects believed to be in it. This method consists solely of a step to scan the bay. Choosing this method will not achieve the task of loading the object into the truck, but it will provide the information that will allow the RAP interpreter to determine whether the second method is appropriate. The second method is applicable if there is room in bay 2, and consists of a step to load the object into bay 2. Methods three and four are the same as these two methods, except that bay 1 is used instead of bay 2.

```
(define-rap
    (index        (load-payload-object ?object))
    (succeed      (or    (location ?object bay1)
                         (location ?object bay2)))
    (method                                              ; method 1
       (context  (not (scanned-since-move bay2)))
       (task-net
          (t1     (really-eye-scan bay2))))              ; check bay2
    (method                                              ; method 2
       (context  (and   (scanned-since-move bay2)
                        (< (number-in-bay bay2) 8)))
       (task-net
          (t1     (move-thing-to ?object bay2))))        ; use bay2
    (method                                              ; method 3
       (context  (not (scanned-since-move bay1)))
       (task-net
          (t1     (really-eye-scan bay1))))              ; check bay2
    (method                                              ; method 3
       (context  (and   (scanned-since-move bay1)
                        (< (number-in-bay bay1) 8)))
       (task-net
          (t1     (move-thing-to ?object bay1)))))       ; use bay1
```

Figure 5.7 Methods for loading an object into the truck

PARETO's plans thus specify the information that is required to make deferred decisions, and also specify how it should be acquired. This is done through the success and method context clauses, which specify the information that should be retrieved from memory, and through explicit perceptory and inferential actions in method task nets.

5.5 Keeping track of the environment

PARETO keeps track of its environment by using the RAP shown in figure 5.8. The task represented by the RAP never succeeds, because the success clause can never be satisfied. The monitor clause checks to see whether the truck has scanned its surroundings recently[6] (within the last 15 time units), and whether it has scanned its surroundings since it changed locations. If not, the scanning action is performed.

In the current implementation of PARETO, the time interval between scans is fixed at 15 time units. An area of future research is to investigate how this can be varied according to how unpredictable the environment appears to be.

```
(define-rap
    (index           (monitor-current-location))
    (succeed            (not T))
    (monitor-state   (or     (not (believe (eye-scanned external) 15))
                             (not (scanned-since-move external))))
    (method
       (primitive (eye-scan external))))
```

Figure 5.8 A RAP for keeping track of the environment

[6] See appendix A. The believe memory retrieval construct is described in (Firby 1989).

5.6 Summary

This chapter has described PARETO's world, which is complex, dynamic, and uncertain. Because PARETO operates in this world, it must acquire information in order to keep track of its surroundings and make appropriate decisions. PARETO plans for all three basic methods of information acquisition: perception, memory retrieval, and inference.

In chapter 4, I presented two examples of PARETO in action. Both these examples demonstrate aspects of PARETO's information acquisition capabilities. In the first example, PARETO opens a combination safe. McCarthy and Hayes (1969) discuss many issues that arise in this situation from the point of view of what it means to know the combination of the safe. The RAPs that represent the plan used by PARETO to carry out the task demonstrate that plans to achieve information goals can be treated in exactly the same way as any other plans. The example also shows how PARETO uses plans to acquire information. For instance, in order to find out the safe's combination it used a plan involving two methods of information acquisition: perception (examining the top of the safe to find the owner's name and examining the directory to find out the birthday) and inference. The acts of perception were used to establish the preconditions for making the inference.

In the second example, we saw how PARETO interleaves steps aimed at keeping track of its environment with steps activated by the pursuit of its other goals. In both examples we saw PARETO following its plans to acquire information in support of deferred decisions, and its ability to combine different methods of information acquisition in its plans. As we saw, PARETO can make unforeseen decisions by taking advantage of unexpected opportunities. In the next few chapters, I shall explain how PARETO recognizes and makes these decisions.

CHAPTER 6

RECOGNIZING POTENTIAL OPPORTUNITIES

6.1 A heuristic filter

An agent in an unpredictable world will frequently find itself in unforeseen situations. When this happens, it must decide whether it should change its plans in the light of the new circumstances, which depends on how the unexpected situations affect its goals. Goals can be affected in two ways: either opportunities to achieve them are presented, or threats to their achievement are posed. If an unexpected turn of events has no effect on the agent's goals, then the agent can simply continue with its existing plans. In order to operate effectively in an unpredictable world, an agent must therefore be able to detect opportunities and threats to its goals, and then to change its plans appropriately in response. In what follows, I will refer only to the process of detecting opportunities. There is no need to consider threats separately, because the existence of a threat automatically implies the existence of a corresponding opportunity to avoid the undesirable consequences of that threat.

In order to respond appropriately to unexpected circumstances, an agent must be able to respond in a timely manner. In many cases, the situation may change rapidly and the window of opportunity may be narrow. An agent cannot afford to perform a detailed analysis of the effects of every nuance of the unexpected situation on each of its many goals, but must instead have a quick and easy method of detecting potential opportunities. It can then limit its reasoning to the analysis of the effects of the new circumstances on those goals that are likely to be affected by them.

In chapter 3, I argued that the agent should use a filtering process to recognize those goals that are easily achievable in a given circumstance (figure 6.1). The viability of a such a filtering process depends on the fact that the world, despite its unpredictablity in general, exhibits many regularities. In particular, it is possible to identify many situation elements that have consistent causal effects on each other, and hence on the agent's goals. For example, *sharp* objects tend to be useful in achieving goals to cut things, and tend to threaten goals that involve things that must not be cut. These functionally interesting tendencies can be labeled with *reference features*—features that are both cheap to compute and relevant to the agent's goals. Moreover, there is often a single *critical factor*, the presence of which determines whether a goal is easily achievable. Reference features associated with critical factors can thus be used as the basis of a heuristic filter indicating those goals that are potentially easy to achieve. The agent can then deliberate more extensively about the goals that pass the filter to determine whether they are indeed easy to achieve and whether they have adverse interactions with other goals (in the latter case the apparent opportunity to achieve them is not genuine). By using a filter in this way the agent can concentrate its reasoning on those goals that are most likely to repay the effort.

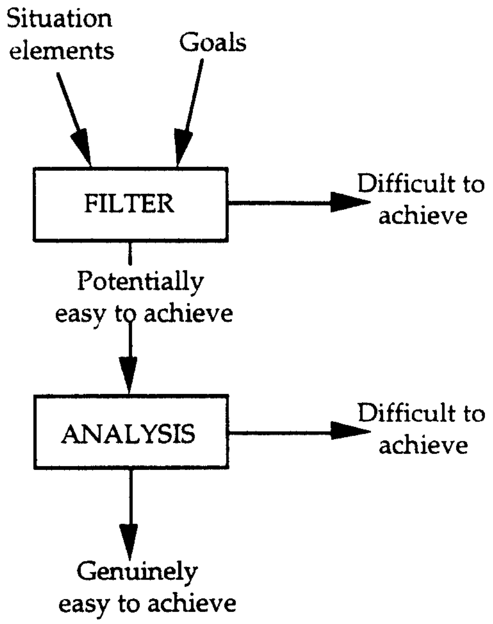


Figure 6.1 A filter for potential opportunities

In this chapter I discuss the principles behind such a filter and describe its implementation in PARETO. Section 6.2 describes PARETO's reference features, section 6.3 describes how they are used in a filtering mechanism, and section 6.4 shows how the filter fits into PARETO's decision mechanisms.

6.1.1 A simple filtering process

Reference features label collections of related functional effects, as described in section 3.5, and are attached to the objects or other situation elements that tend to cause those effects. To use reference features to indicate situation elements that are likely to be useful, an agent must also attach reference features to goals. This can be done by associating reference features with goals that are likely to be achieved by the related functional effects. By doing this, if there is an object in the current situation that shares a reference feature with one of the agent's goals, that goal is likely to be easily achievable through the use of the object.

For example, if an agent has the goal of cutting something (slicing a cake, for example), then the presence of a knife indicates that the goal is likely to be easily achievable. The reference feature *sharp* labels a collection of effects involving the destruction of an object's structural integrity, such as cutting soft objects, scratching hard ones, snapping taut strings, bursting taut membranes, piercing surfaces and splitting brittle objects. Knives are considered *sharp* because they are commonly involved in producing these effects. The goal of slicing a cake, on the other hand, requires for its achievement the destruction of the cake's structural integrity. The functional effects labeled by the reference feature *sharp* may help in achieving the goal. The reference feature sharp is thus attached to both the knife and the goal, and the presence of a sharp object, the knife, indicates that the goal of slicing a cake is likely to be easily achievable.

Reference features can thus form the basis of the following filtering process:

1. Find the reference features of the elements in the current situation.
   These reference features indicate the functional effects that may be easily achievable.
2. Find the reference features of the agent's current goals.
   These reference features indicate the functional effects that may help in achieving the current goals.
3. Compare the two sets of reference features.
   Those goals whose reference features are also attached to elements of the current situation are likely to be easily achievable.

The agent can then check to see whether the indicated goals are genuinely easy to achieve.

The use of this simple filter allows an agent to ignore all goals except those indicated as being likely to be easily achievable. The filter uses the regularity of the world as expressed in the reference features to reduce the amount of reasoning that need be done to recognize opportunities.

6.1.2 Issues in filtering

To be effective a filtering process must be quick and inexpensive to apply. The filtering process must therefore use only the information that the agent already possesses; it cannot involve any significant amount of explicit information acquisition. It must also involve minimal reasoning. These requirements mean that the agent must have ready access to the reference features of its goals and of the elements of its current situation. The filtering algorithm itself is very simple, and once the reference features have been found matching them is trivial.

An important characteristic of reference features is thus their *availability*—the agent does not require elaborate inference to find the reference features attached to a situation element. This characteristic is crucial for their use in an effective filtering mechanism. They must be easily inferable in most situations in which the associated causal effects are present. If they are hard to infer, they cease to be useful in the filtering process. If there are many situations in which the associated causal effects are present, but the reference feature itself is not inferred by the agent, the filtering process becomes unreliable, failing to spot potential opportunities. Similarly, reference features must be inferable in few of the situations in which they are not applicable, so that the agent does not waste its resources analyzing goals for which there are no potential opportunities. The next section describes the reference features that PARETO uses.

6.2 PARETO's reference features

6.2.1 Reference features attached to situation elements

The functionally interesting properties of situation elements are those that affect the achievement of goals. The reference features of situation elements are therefore directly related to the goals they help achieve.

PARETO's goals involve the operation of a truck that makes deliveries to building sites. Hypothetical workers at these sites request delivery of objects needed in their work (see appendix B for a full description of delivery goals). Requests are often specified in terms of the type of object that is

Object	Goals achieved by delivering this object	Reference features
bag	something to carry tools a bag	carrier, bag, fabric
bucket	something to carry water something to carry dirt a bucket	carrier, bucket
ladder	a ladder	ladder, large
paintbrush	a paintbrush	paintbrush
scissors	something to cut twine something to open a can of solvent a pair of scissors	sharp, prying, scissors
screwdriver	something to open a can of paint something to open a can of solvent a screwdriver	prying, screwdriver
screws	a permanent fastener some screws	fastener, screws

Figure 6.2 Some of PARETO's reference features—delivery objects

required, so every object has its type as a reference feature: for example, all screwdrivers have the reference feature screwdriver.. Other goals are more loosely specified: for example, PARETO may be asked to deliver something that can be used to carry other objects. In PARETO's world, boxes, bags, buckets and wheelbarrows can all be used to carry things, and have the reference feature *carrier*. Similarly, scissors, screwdrivers, chisels and hammers can all be used to pry the lids of cans, and have the reference feature *prying*. Things can be fastened together temporarily with string or duct tape or permanently with nails or screws, which all have the reference feature *fastener*. Some of the reference features attached to delivery objects are shown in figure 6.2, and appendix B gives a full listing.

Note that PARETO does not use these objects to perform the functions specified. It merely delivers suitable objects to the workers in the building sites, who state their requirements in these functional terms. The functional aspects are interesting to PARETO only insofar as they affect its goals. Moreover, PARETO's reference features refer only to these delivery goals: figure 6.2 does not list reference features that we might attach to the real-life counterparts of the objects in question.

It is important to note that reference features in general, and PARETO's reference features in particular, do not form a complete functional classification of situation elements. For example, not all objects that have the reference feature *sharp* help with the goal of finding something to cut twine, and not all objects that have the reference feature *carrier* help with the goal of finding something to carry dirt, or water, or tools. Reference features are useful only because they form the basis of a heuristic filter that indicates some goals as being potentially easy to achieve. Goals that pass the filter can be analyzed further: although the filtering mechanism must reduce the amount of detailed reasoning required, it need not obviate the need for all reasoning. Reference features form a *simplified* classification of objects by their functionally interesting properties.

An important subset of reference features are those that indicate possible threats to goals. For example, a *sharp* object, such as a knife, may cut a *soft* object, such as a ball of twine thus rendering it

Goal criterion	Conditions of satisfaction	Reference features
carry-tools	(or (class ?object box) (class ?object bag))	carrier
cut-twine	(or (class ?object scissors) (class ?object knife))	sharp
open-paint	(or (class ?object hammer) (class ?object screwdriver) (class ?object chisel))	prying
red-wood	(and (class ?object wood) (color ?object red))	wood, red, colored
temp-fastener	(or (class ?object twine) (class ?object tape))	fastener
wood-8	(and (class ?object wood) (size ?object8))	wood
yellow-paint	(and (class ?object paint) (color ?object yellow))	paint, yellow, colored

Figure 6.3 Some of PARETO's goal-criterion reference features

unsuitable for its purpose (tying things together) and making it unacceptable to the worker who has requested it. The way in which these reference features are used is discussed in chapter 7.

6.2.2 Reference features attached to goals

Reference features are used to indicate the functional properties that facilitate the achievement of goals. Most of PARETO's goals involve delivering objects, and the reference features attached to these goals reflect the properties of the objects to be delivered. Delivery goals specify a *goal-criterion* that must be met by the object to be delivered (see chapter 4). For example, the carry-tools goal criterion is met if the object in question is a box or a bag. There are two types of goal-criterion: those that specify only the class of object to be delivered, and more complex criteria. Every goal-criterion that specifies only the class of object to be delivered has the reference features of that class of object. For example, the hammer goal-criterion is met by any object that is a hammer, and has the reference features *hammer* and *prying* attached to it. Some complex goal-criteria are shown in figure 6.3, together with their reference features, and appendix B gives a full listing.

Goal-criteria are used to specify the objects required in PARETO's tasks to deliver an object, find an object, and load an object into a cargo-bay. Every task involving a goal-criterion has the reference features of that goal-criterion. For example, a task to deliver something with which to carry tools has the reference feature *carrier*, and a task to load a can of red paint into the cargo bay has the reference features *paint*, *red*, and *colored*.

Other tasks that have reference features include those to travel to the location of a building site, which has the reference feature *site*, the task to unload an object at a building site, which also has the reference feature *site*, and the task to refill the fuel tank, which has the reference features *fuel* and *fuel-drum*.

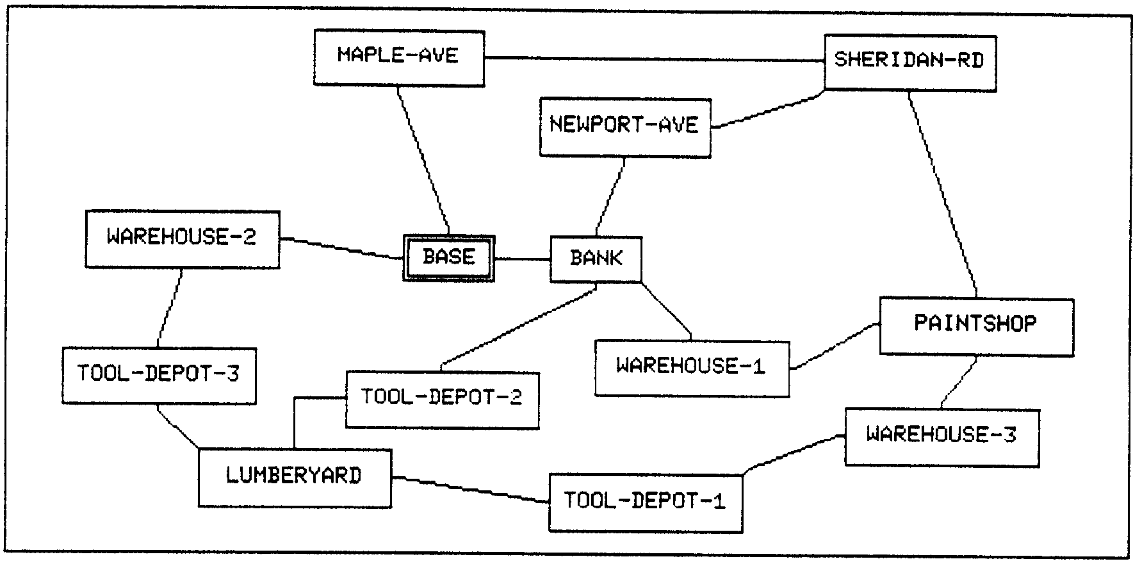


Figure 6.4 PARETO's world map

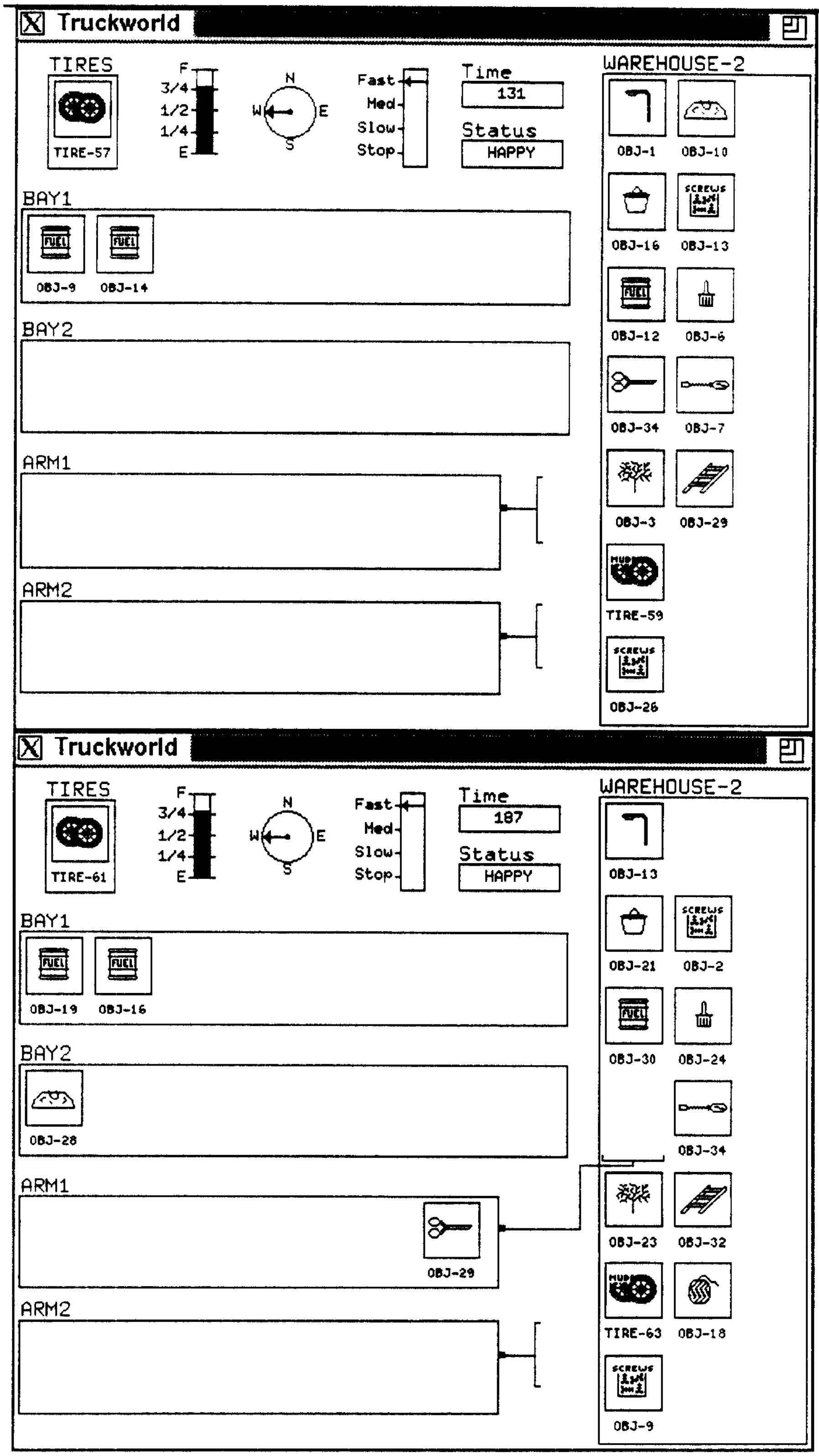
6.3 PARETO's use of reference features

In this section I shall describe PARETO's filtering process in some detail. To set the discussion in context, consider the following example. The truck is at its base, and PARETO has just received two requests: one from a worker at ils, a building site on maple-ave, for something with which to carry tools, and one from a worker at tech, a site on sheridan-rd, for something with which to cut twine. PARETO decides to pursue the carry-tools goal first, and sets off for the lumberyard, where boxes are generally to be found (see the map in figure 6.4).

The first location the truck passes through on its way to the lumberyard is warehouse-2. There are many objects at this location (figure 6.5a), among them a bag which constitutes an opportunity for the carry-tools goal, and PARETO responds by loading it into a cargo bay. PARETO also recognizes and takes advantage of the opportunity for its cut-twine goal (which is not currently being pursued) offered by the presence of a pair of scissors (figure 6.5b).

Since there is now no need to go to the lumberyard, PARETO abandons its plan to go there and instead makes its way to maple-ave, where it delivers the bag, thus achieving the carry-tools goal (figure 6.6A). Finally the truck goes to sheridan-ave, delivering the scissors to achieve the cut-twine goal (figure 6.6B).

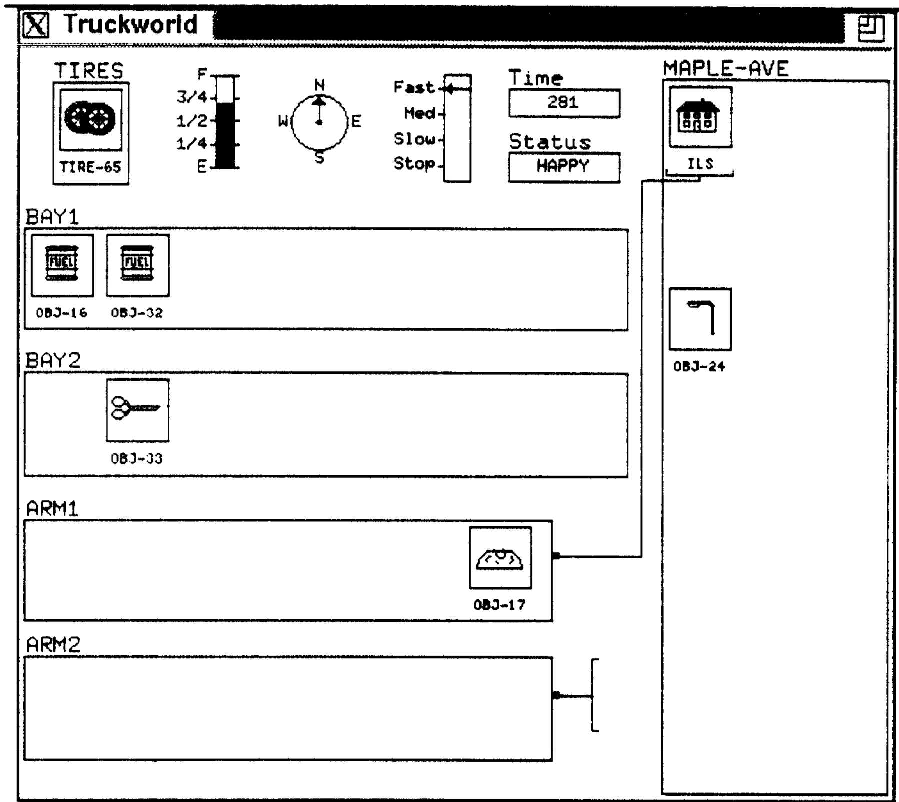
PARETO recognizes the opportunities that it encounters in the warehouse through the use of reference features. A bag has the reference feature *carrier*, signifying that it is often used to carry things, and the carry-tools goal also has the reference feature *carrier*, signifying that objects with that reference feature are often involved in its achievement. Similarly, the cut-twine goal has the reference feature *sharp*, as do the scissors. By using these reference features, PARETO is able to recognize the two opportunities and to take advantage of them. As we shall see, this requires minimal reasoning and no explicit information acquisition.



A: The truck arrives at warehouse-2, where there are many different objects, including a bag and a pair of scissors.

B: The truck has loaded the bag into BAY2 and is loading the scissors.

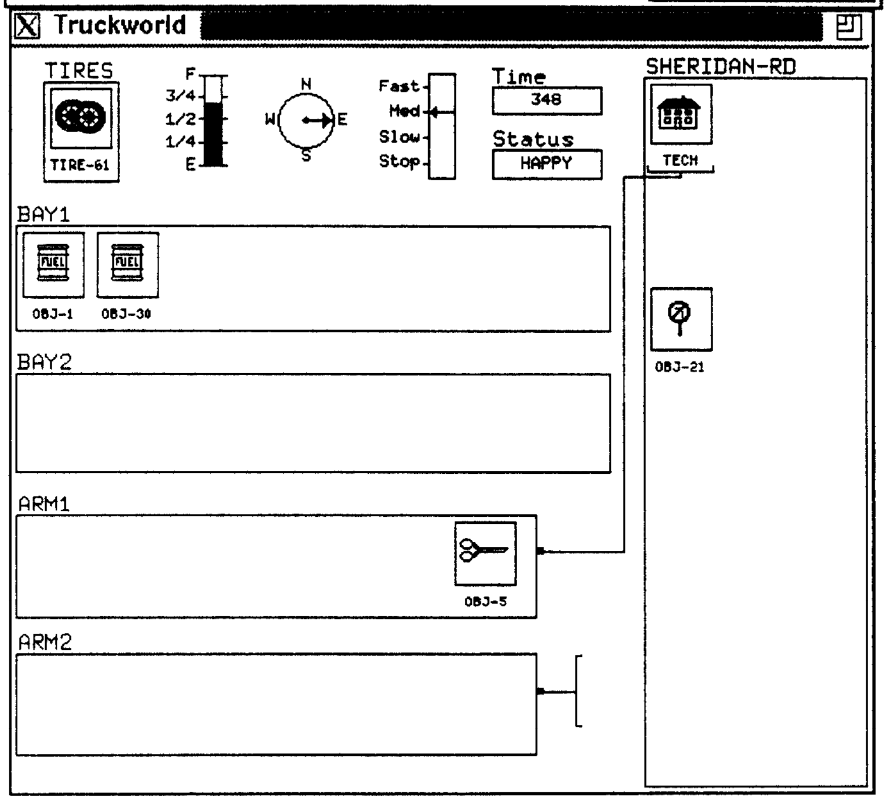
Figure 6.5 PARETO takes two opportunities



Time 281

Status HAPPY

A: The truck delivers the bag at ils.



Time 348

Status HAPPY

B: The truck delivers the scissors at tech.

Figure 6.6 PARETO takes two opportunities (contd.)

```
(define-rap
   (index        (deliver-object ?criterion ?destination ?order-id))
...
   (method
      (context  (location ?destination ?place))
      (task-net
         (t1      (find-object ?criterion => ?oplace ?object)
                     (for t2))
         (t2      (load-payload-object ?object ?criterion)
                     (for t3))
         (t3      (truck-travel-to ?place)
                     (for t4))
         (t4      (unload-at ?destination ?place ?object)))))
```

Figure 6.7 The deliver-object RAP's subtasks

In this section, I shall use this example to illustrate PARETO's use of reference features to recognize easily achievable goals. Let's look at PARETO's actions in more detail, using program output (edited for brevity). First, PARETO receives two new delivery goals:

```
--> New order ORDER-5: something to CARRY TOOLS for ILS
--> New top level goal: DELIVER-OBJECT :: [6:6]
--> New order ORDER-4: something to CUT TWINE for TECH
--> New top level goal: DELIVER-OBJECT :: [7:7]
```

PARETO decides to pursue the goal for order-5, something to carry tools (goal 6 in the output). The deliver-object task has four subtasks (see figure 6.7), which must be performed in order. The first step is to find a suitable object. In this case, objects that are suitable for carrying tools are boxes and bags. PARETO knows that boxes are often found at the lumberyard, so PARETO decides to look for one there and heads off in that direction, planning to go via warehouse-2 and tool-depot-3 (see figure 6.4).

```
... Processing task: <DELIVER-OBJECT CARRY-TOOLS ILS ORDER-5> :: [6:6]
... Processing task: <FIND-OBJECT CARRY-TOOLS => ?PLACE-NAME ?OBJECT> :: [6:5]
... Processing task: <TRUCK-TRAVEL-TO LUMBERYARD> :: [6:11]
        +++ Primitive hardware operation (TRUCK-MOVE) [6:22]
*** Arrived at WAREHOUSE-2
```

The first location the truck passes through is warehouse-2. When it first arrives there, it scans its neighborhood as it always does when it arrives at a new place. There are quite a number of objects there (figure 6.6A), most of them irrelevant to PARETO's current goals. However, two of the objects, a pair of scissors and a bag, are directly relevant to its cut-twine and carry-tools goals. There are thus two potential opportunities that PARETO should recognize.

PARETO recognizes potential opportunities in two stages. First, it uses a filtering process based on reference features to indicate those of its goals that are likely to be easily achievable in the current situation. Next, it analyzes the indicated goals in more detail. The filtering process itself consists of three steps:

1. Find the reference features of the current situation;
2. Find the reference features of PARETO's goals;
3. Match the two sets of reference features.

In the remainder of this section I describe each of these steps individually.

6.3.1 Reference features of the current situation elements

The first step in the filtering process is to find the reference features of the current situation. In PARETO's world, these comprise the reference features of all objects at the truck's current location.

As discussed above, it is critical that the determination of the reference features in the current situation involves minimal cognitive effort. In general, it may be a complex matter to determine exactly what the elements of the current situation are. Spreading the net too wide by including all elements that might possibly be present would severely reduce the efficiency of the filtering process: determining a complete list of such elements is in general not trivial, and in an unpredictable world many of them might not actually be present anyway. In any case, using elements that are not actually present would reduce the reliability of the filtering process by indicating too many potential opportunities that turn out to be false leads. On the other hand omitting elements that are in fact present would again reduce the reliability of the filtering process, this time by leading to a failure to recognize genuine opportunities. The agent must therefore compromise, and use the reference features of only those elements of its current situation that it can *easily* determine to be present.

PARETO uses the objects that it has actually seen at its current location on its current visit. As we saw in chapter 3, it has a goal to keep track of its environment which it pursues by scanning the truck's neighborhood at regular intervals and whenever the truck arrives at a new location. PARETO uses the information acquired through these scanning actions to keep track of a *focus group*, to which it adds every object that it observes to be present at its current location. The focus group is reset every time the truck changes location, and includes only those objects that the truck has seen at the current location. This means, for example, that if PARETO believes there is a gold bar in a safe at the current location the gold bar will only be in the focus group if the safe is opened and the gold bar actually observed. The use of focus groups means that PARETO does not search memory for objects that it believes to be at the current location. This has two advantages. First, possibly costly memory search is avoided. PARETO can rely on the results of the perception that it is performing in the course of pursuing its tasks. It need acquire no information solely for the purpose of using reference features to spot potential opportunities. Second, only those objects that PARETO is reasonably certain are at the current location, and are therefore accessible to the truck, are included.

PARETO derives the reference features of the objects in the focus group from the properties of the objects. Most reference features are associated with the class of an object: for example, a bag has the reference features *bag*, *soft*, and *carrier* (see appendix B). Other reference features may be associated with other properties: for example, any object whose color is known has that color as a reference feature. PARETO never gathers new information specifically for the purpose of finding reference features, but simply uses the information that it already has. For example, a can of paint always has the reference feature *colored* (attached to the class paint) but if PARETO does not know its color it will not have any specific color as a reference feature.

PARETO thus finds the reference features of the elements in its current situation with minimal reasoning and with no explicit information acquisition. In our example, it finds the following reference features:

```
... Reference features for ITEM-20 : (BAG CARRIER)
... Reference features for ITEM-19 : (LADDER LARGE)
... Reference features for ITEM-18 : (SCREWDRIVER PRYING)
... Reference features for ITEM-17 :  (PAINTBRUSH)
... Reference features for ITEM-16 :  (SCREWS FASTENER)
... Reference features for TIRE-23 :  (MUD-TIRES)
... Reference features for ITEM-14 :  (SAPLING)
... Reference features for ITEM-13 :  (SCISSORS SHARP)
... Reference features for ITEM-12 :  (FUEL-DRUM)
... Reference features for ITEM-11 :  (BUCKET CARRIER)
... Reference features for ITEM-10 :  (STREET-LIGHT)
```

6.3.2 Reference features of goals

The second stage in the filtering process is to find the reference features of the agent's goals. The agent may have an enormous number of goals of varying types. For example, a student's goals first thing one morning might include: eat breakfast, feed the cat, attend a class, write a dissertation, read a book, see a movie, and buy a birthday present for a friend, not to mention numerous more long term goals such as staying alive, being happy, and so on. Each of these goals is likely to have subgoals: in order to eat breakfast, for example, you must get cereal from the cupboard, a bowl from another cupboard, milk from the fridge, a spoon from the drawer, and make coffee (which itself has several subgoals). Each of these goals is also a subgoal of other goals: eating breakfast in order to stay alive, for example, and attending classes and writing a dissertation in order to obtain a degree, and eventually to obtain a job.

The processing required to come up with a complete set of goals, including all subgoals, would be prohibitively expensive: the inference involved in determining all possible subgoals of any given goal would in itself be a major planning exercise. However, an agent cannot possibly attend to all its goals. Many of them must be left implicit, and ignored until they are specifically made active. The filtering process for goals that are potentially easy to achieve need consider only those goals that are currently active. The agent's many implicit goals can be ignored. This policy can easily lead to missed opportunities: for example, suppose you are planning a trip abroad, and forget that you will have to obtain some foreign currency while you are still in this country. You may have several opportunities to get foreign currency, but will fail to take advantage of them because the goal to get foreign currency remains implicit. However, this is a problem with the attention mechanism, and presumably could be addressed through learning from one's mistakes.

PARETO has only a very rudimentary attention mechanism: with a few exceptions (explained in chapter 8), all the tasks on the task agenda, and hence all PARETO's goals, are active.

PARETO attaches reference features directly to tasks on the task agenda. Each RAP specifies reference features for its task. For example, the RAP for the unload-at task specifies the reference feature *site*. This task directs the truck to complete a delivery by unloading the object to be delivered at the relevant building site. The presence of a building site thus indicates that the task is possibly easy to achieve.

PARETO's delivery goals are expressed in terms of *goal-criteria* (see chapter 4). Each goal-criterion has reference features associated with it; for example, the goal-criterion carry-tools has the reference feature *carrier* associated with it (see appendix B for a complete list of the reference features attached to goal-criteria). The deliver-object RAP specifies that the reference features associated with the goal-criterion should be attached to the deliver-object task (see figure 6.8). When reference features are expressed in terms of input variables, they are instantiated when the task is placed on the task agenda.

```
(define-rap
    (index         (deliver-object ?criterion ?destination ?order-id))
    (reference-features ((rf-from-crit ?criterion)))
    (method
       ... tasks to find and deliver an object...  ))
```

Figure 6.8 The deliver-object RAP's reference features

For example, when a deliver-object task is placed on the agenda for the carry-tools goal in our example, it has the reference feature *carrier* attached to it, while the deliver-object task for the cut-twine goal has the reference feature *sharp.*

6.3.3 Matching reference features

The third and final stage in the filtering process is to compare the reference features found from the current situation with those found from the active goals. In our example, there are now five tasks on the agenda that serve the carry-tools goal: the top level deliver-object task, and its four children, find-object, load-payload-object, truck-travel-to, and unload-at (see figure 6.8). Three of these have the reference features associated with the carry-tools goal-criterion. This means that they have the *carrier* reference feature, which is also attached to the bag that is at the current location. PARETO's filtering process therefore indicates that these three tasks are potentially easy to achieve:

```
+++    Potential opportunity for <DELIVER-OBJECT CARRY-TOOLS ILS ORDER-9>
       ITEM-20 (BAG) has reference feature CARRIER
+++    Potential opportunity for <FIND-OBJECT CARRY-TOOLS => ?PLACE-NAME ?OBJECT>
       ITEM-20 (BAG) has reference feature CARRIER
+++    Potential opportunity for <LOAD-PAYLOAD-OBJECT ?OBJECT CARRY-TOOLS>
       ITEM-20 (BAG) has reference feature CARRIER
```

There is also a bucket, which also has the *carrier* reference feature, at the current location. The same three tasks are indicated as being potentially easy to achieve by virtue of the presence of the bucket:

```
+++    Potential opportunity for <DELIVER-OBJECT CARRY-TOOLS ILS ORDER-9>
       ITEM-11 (BUCKET) has reference feature CARRIER
+++    Potential opportunity for <FIND-OBJECT CARRY-TOOLS => ?PLACE-NAME ?OBJECT>
       ITEM-11 (BUCKET) has reference feature CARRIER
+++    Potential opportunity for <LOAD-PAYLOAD-OBJECT ?OBJECT CARRY-TOOLS>
       ITEM-11 (BUCKET) has reference feature CARRIER
```

The top level deliver-object task for the cut-twine goal has not yet been processed by the RAP interpreter, and is therefore the only task for that goal on the task agenda. This task has the reference feature *sharp,* as do the scissors that are present at the current location. The filtering process therefore indicates that this task, too, is potentially easy to achieve:

```
+++    Potential opportunity for <DELIVER-OBJECT CUT-TWINE TECH ORDER-8>
       ITEM-13 (SCISSORS) has reference feature SHARP
```

6.3.4 The filtering process in PARETO

The filtering process that PARETO uses thus consists of three steps: gathering the reference features of the objects in the current focus group, gathering the reference features from the active tasks on the task agenda, and comparing the two sets of reference features to determine those tasks that are likely to be easily achievable. None of these three stages requires the explicit acquisition of information, and all involve minimal reasoning. This filtering process thus allows PARETO to recognize potential opportunities quickly and easily.

In our example the use of reference features has indicated that there are four tasks that may be easy to achieve in the current situation. Before PARETO can decide to go ahead and perform any of them, it must determine whether they are indeed genuinely easy to achieve. Otherwise, there are no opportunities for them in the current situation and PARETO should continue with its current plan, to go to the tool shed.

6.4 Easily achievable goals

The filtering process described above only indicates those goals that are *likely* to be easily achievable. The planner must therefore follow up with a more detailed analysis of the indicated goals to determine whether they are actually easy to achieve.

There are two reasons that a goal may be indicated as being potentially easy to achieve when it is not. First, because the reference features associated with a goal are necessarily not as specific as the actual goal, it is possible that a situation element with the relevant reference feature will not in fact facilitate the achievement of the goal. For example, the goal of slicing a cake has the reference feature *sharp*, but not all sharp objects can actually be used in achieving it: a pair of scissors, for instance, is of little help. This situation commonly arises in PARETO. The reference feature *carrier*, for instance, is attached to the carry-tools goal-criterion, but there are some objects having the reference feature *carrier* that cannot be used to carry tools, *e.g.* buckets and wheelbarrows. The presence of a bucket or wheelbarrow will thus result in the carry-tools goal being indicated as likely to be easily achievable when it is not.

The second reason for a false indication that a goal is potentially easy to achieve is a result of the standard "conjunctive goal problem." The use of reference features indicates only that one subgoal of the indicated goal is particularly easy to achieve: in general, any goal has many subgoals, all of which must be achieved in order to achieve the goal itself. The use of reference features assumes that there is one dominant subgoal that is harder to achieve than the others. For example, in order to cut a piece of string we need to have something sharp to cut it with, the piece of string must be immobilized, the blade of the cutting tool must be safely manipulable, and so on. The assumption is that the majority of these subgoals can be achieved whatever the circumstances and that the procurement of a cutting tool is thus the primary issue. If this turns out to be false, the indicated goal will not in fact be easily achievable.

As these examples show, PARETO must confirm that a task is genuinely easy to achieve in order to determine whether there is an opportunity for it in the current situation. In chapter 3, I described two types of opportunities: predicted opportunities, that have been foreseen in plan construction, and unexpected opportunities. A predicted opportunity is simply a situation in which the goal is easily

```
(define-rap
   (index        (find-object ?criterion => ?place ?object))
   (succeed      (and  (my-location ?place)
                       (location ?object ?place)
                       (meets-criterion ?object ?criterion true))))
   (method
      (context  ...there's an object here that might be suitable...)
      (task-net    ...check to see whether it will do...))
   (method
      (context  (and  (my-location ?here)              ; there's a suitable object
                      (location ?thing ?aplace)        ; somewhere else
                      (not (= ?aplace  ?here))
                      (meets-criterion ?thing ?criterion true)))
      (task-net
         (t1    (truck-travel-to ?aplace)
                   ((my-location ?aplace) for t2))
         (t2    (check-object-ok ?thing ?criterion))))
   ...other methods...)
```

Figure 6.9 The find-object RAP

achievable because the next step in the plan can be carried out: in PARETO's terms, the next task in the plan is ready for processing by the RAP interpreter—*i.e.*, it is not waiting for any others to be completed (see chapter 4). An unexpected opportunity is one that has not been foreseen in plan construction. In PARETO's terms, a task is unexpectedly easy to achieve if it has already succeeded, regardless of whether it is the next task to be processed for the goal in question.

For instance, in our example the find-object task for the carry-tools goal has already succeeded, because there is an object meeting the goal-criterion (the bag) at the truck's current location. The load-payload-object task for the same goal is ready to be processed: the truck can now load the bag into its cargo bay. The finding of a bag at the warehouse is unexpected: PARETO had planned to go to the lumberyard and find a box. It would also clearly be productive to process the deliver-object task for the cut-twine goal: there is a pair of scissors at the truck's current location which could be used to achieve the goal. If the deliver-object task for the cut-twine goal is processed it will result in a find-object task which has already succeeded and a load-payload-object task which will itself be ready for processing. Although the finding of a pair of scissors at the warehouse is not in itself expected, the deliver-object task is the next step in the plan for the cut-twine goal, and were it not for the fact that PARETO chose to pursue the carry-tools goal first, would be processed next.

6.4.1 Unexpectedly easy to achieve

PARETO reasons about unexpected opportunities by looking at the success clauses of the tasks in question. In our example, there is an unexpected opportunity, due to the presence of the bag, for the carry-tools goal. In fact, there are three tasks for this goal that are potentially easy to achieve: the deliver-object, find-object, and load-payload-object tasks. The deliver-object task has not yet succeeded (it succeeds when the delivery is successfully completed) and is not ready to be processed, as it is waiting for the completion of its subtasks (find-object, load-payload-object, truck-travel-to, and unload-at). The load-payload-object task has likewise not succeeded, and is waiting for the find-object task to complete. These two tasks are therefore rejected.

The find-object task, on the other hand, has succeeded: its success criterion is met by the presence of the bag, which is at the same location as the truck and meets the goal-criterion for the carry-tools goal (see figure 6.9). This task is therefore classified as easy to achieve, and in fact there is a genuine opportunity for it:[1]

```
+++ Has already succeeded <FIND-OBJECT CARRY-TOOLS => ?PLACE-NAME ?OBJECT>
*** Taking unexpected opportunity: [6:5]
    <FIND-OBJECT CARRY-TOOLS => ?PLACE-NAME ?OBJECT>
... Processing task: <FIND-OBJECT CARRY-TOOLS => ?PLACE-NAME ?OBJECT> :: [6:5]
... Task succeeded: <FIND-OBJECT CARRY-TOOLS => WAREHOUSE-2 ITEM-20> :: [6:5]
```

Because this task has already succeeded, there is now no need to perform the subtask of going to the lumberyard to look for a box. This task is therefore abandoned—it is simply removed from the task agenda. The next task to be performed in pursuit of the top-level goal is therefore to load the bag into the truck:

```
... Processing task: <LOAD-PAYLOAD-OBJECT ITEM-20 CARRY-TOOLS> :: [6:8]
...
... Task succeeded: <LOAD-PAYLOAD-OBJECT ITEM-20 CARRY-TOOLS> :: [6:8]
```

It is important to note that even if the bag were not present, the find-object task for the carry-tools goal would still have been indicated as being potentially easy to achieve, due to the presence of the bucket. However, in this case further analysis would have shown that it was not genuinely easy to achieve, as neither the bucket nor any other object present at the warehouse meet the carry-tools goal-criterion and thus enable the success criterion of the find-object task to be met.

6.4.2 Predictably easy to achieve

In our example, there is a predicted opportunity for the cut-twine goal due to the presence of the scissors. Let us consider how PARETO reasons about the deliver-object task for the cut-twine goal, which was indicated as being potentially easy to achieve. This task has clearly not yet succeeded: it succeeds only when the delivery is made and accepted. However, it is ready to be processed, as it is not waiting for any other tasks. It can therefore be counted as being easy to achieve, and in fact it turns out that there is a genuine opportunity to pursue this task, which PARETO takes:

```
*** Taking expected opportunity: [7:7]
    <DELIVER-OBJECT CUT-TWINE TECH ORDER-8>
*** Changing goals from 6 to 7
... Processing task: <DELIVER-OBJECT CUT-TWINE TECH ORDER-8> :: [7:7]
```

As before, four new tasks are placed on the task agenda. The presence of the scissors constitutes an opportunity for two of them:

```
+++ Potential opportunity for <FIND-OBJECT CUT-TWINE => ?PLACE-NAME ?OBJECT>
    ITEM-13 (SCISSORS) has reference feature SHARP
+++ Potential opportunity for <LOAD-PAYLOAD-OBJECT ?OBJECT CUT-TWINE>
    ITEM-13 (SCISSORS) has reference feature SHARP
```

Again, the find-object task has already succeeded, and the next step is to load the scissors into the truck:

```
... Processing task: <FIND-OBJECT CUT-TWINE => ?PLACE-NAME ?OBJECT> :: [7:5]
```

[1] Recall that there is only an opportunity for a task if it is easy to achieve and does not interfere with other goals. PARETO's method of analyzing goal interactions is described in chapter 7.

```
...  Task succeeded: <FIND-OBJECT CUT-TWINE => WAREHOUSE-2 ITEM-13> :: [7:5]
...  Processing task: <LOAD-PAYLOAD-OBJECT ITEM-13 CUT-TWINE> :: [7:8]
...  Task succeeded: <LOAD-PAYLOAD-OBJECT ITEM-13 CUT-TWINE> :: [7:8]
```

The next step for this goal would be to head off to sheridan-rd to deliver the scissors. However, this would conflict with PARETO's previous goal of going to maple-ave to deliver the bag.[2] PARETO decides to return to its original carry-tools goal, and goes to maple-ave first:

```
***  Changing goals from 7 to 6
...  Processing task: <TRUCK-TRAVEL-TO MAPLE-AVE> :: [6:5]
***  Arrived at MAPLE-AVE
...  Task succeeded: <TRUCK-TRAVEL-TO MAPLE-AVE> :: [6:5]
...  Processing task: <UNLOAD-AT ILS ITEM-20> :: [6:10]
...  Task succeeded: <UNLOAD-AT ILS ITEM-20> :: [6:10]
...  Task succeeded: <DELIVER-OBJECT CARRY-TOOLS ILS ORDER-9> :: [6:6]
-->  Order ORDER-7 for something to CARRY TOOLS at ILS filled at time 283
```

Finally PARETO delivers the scissors at sheridan-rd:

```
***  Changing goals from 6 to 7
...  Processing task: <TRUCK-TRAVEL-TO SHERIDAN-RD> :: [7:5]
***  Arrived at SHERIDAN-RD
...  Task succeeded: <TRUCK-TRAVEL-TO SHERIDAN-RD> :: [7:5]
...  Processing task: <UNLOAD-AT TECH ITEM-13> :: [7:12]
...  Task succeeded: <UNLOAD-AT TECH ITEM-13> :: [7:12]
...  Task succeeded: <DELIVER-OBJECT CUT-TWINE TECH ORDER-8> :: [7:7]
-->  Order ORDER-6 for something to CUT TWINE at TECH filled at time 351
```

PARETO's method of analyzing potential opportunities to see if they are genuinely easy to achieve is thus very simple. It involves acquiring information by retrieval from memory (the success criterion) and from the task agenda (whether a task is waiting for others).

6.5 Summary

In this chapter I have described a filtering process based on reference features that can be used to indicate goals that are potentially easy to achieve. As implemented in PARETO, the process uses a very simple algorithm, and involves minimal reasoning and no explicit acquisition of information. The agent can then go on to analyze the indicated goals in more detail to determine whether they are genuinely easy to achieve. The analysis that PARETO performs involves information acquisition by retrieval from internal knowledge sources (memory and task agenda), but involves no inference. The entire process by which PARETO recognizes which of its tasks are easy to achieve in the current situation thus involves little information acquisition in either the filtering or analysis stages.

Recognizing those tasks that are easy to achieve is the first stage in determining the tasks for which the current situation constitutes a good opportunity. PARETO's ability to perform this recognition quickly and easily enables it to respond quickly and appropriately to the opportunities that it encounters, as we shall see in the following two chapters.

[2] The method by which PARETO determines this is discussed in chapter 7. How PARETO chooses between the available opportunities is described in chapter 8.

The current situation constitutes an opportunity for a task if that task is easy to achieve and has no adverse effects on the agent's other goals. Chapter 7 discusses the issues involved in determining whether a task interacts adversely with others.

As we saw in the example described in this chapter, an agent may have several opportunities at once. When the truck arrived at the warehouse, there were opportunities for both the carry-tools goal and the cut-twine goal. The agent must be able to decide which of the available opportunities to take; PARETO's choice mechanism is described in chapter 8.

CHAPTER 7

REFERENCE FEATURES AND GOAL INTERACTIONS

7.1 Goal interactions

The last chapter described a filtering mechanism for potential opportunities that works by indicating those goals that are easily achievable. However, even if a goal is easily achievable there may be no opportunity for it: pursuing it might have adverse effects on the agent's other goals. An agent must therefore perform further analysis to determine whether potential opportunities identified by the filtering process are indeed opportunities for the indicated goals. In general, this analysis is complex, but it must be performed quickly enough that opportunities can be seized while they still exist.

The same considerations apply to this part of the task of recognizing opportunities as apply to recognizing easily achievable goals. There is a large number of factors involved in the achievement of each of the agent's many goals; each factor may be implicated in a problematic interaction with another goal, and no factor can be ruled out *a priori*. The analysis required to determine whether any given set of factors do lead to an interaction may be arbitrarily complex, and the number of possible sets of factors increases exponentially with the number of goals the agent has. An exhaustive analysis would be prohibitively time-consuming and expensive, and would prevent a timely and appropriate response to opportunities that are presented. These similarities to the earlier problem suggest the use of a similar approach, and PARETO does in fact use a filtering mechanism based on reference features.

This chapter describes a simple mechanism that allows an agent to detect problematic interactions without performing extensive projection and analysis of all possible interactions between its goals. First, I describe the notion of *effective independence*, a simplifying assumption that captures the intuition that most of the time most goals have no significant interactions with each other. In the remainder of section 7.1 I describe the outline of a heuristic filtering process, very similar to the one described in the last chapter, that uses reference features to indicate potentially problematic interactions. Section 7.2 consists of a description of this heuristic filtering process as implemented in PARETO. The potentially adverse interactions indicated by the filtering process must be analyzed in more detail to determine whether or not they are genuine, and this process is discussed in section 7.3. Finally, the whole procedure by which interactions are detected is summarized in section 7.4, which also discusses how this procedure fits in to the overall process of opportunity detection.

7.1.1 Effective independence

In deciding whether or not to pursue an opportunity, an agent might in principle consider any number of possible interactions between its existing plans and the pursuit of the opportunity. Suppose, for example, you are running some errands and have the opportunity to collect some books that you had lent to a friend. You would have to stop by the side of the road to pick up the books which might involve problems with passing traffic; the other objects by the side of the road might obstruct you during the loading process; the extra weight of the books might make the load exceed your car's weight capacity; the books take up space in your car, which might not be available; they might damage other objects already in the car; other objects might damage the books; if the books are loose they might move around in transit and get damaged; the time taken to pick them up might make you late for another appointment; the books' weight might affect the fuel consumption and speed of your car, and hence the your ability to run other errands; and so on. The number of possible interactions is enormous.

Unfortunately, any one of these interactions could actually constitute a serious threat. For example, you might already have a very heavy load in your car, and the extra weight of ten boxes of books would be too much for the suspension; if you are also planning to collect some furniture there may be no room for the books as well; boxes of books may crush some dry-cleaning you have just picked up; if the books are not in boxes, garden tools may damage them; if you are very short of gas, the extra weight of the books might make all the difference between making it to the nearest gas station or not; and so on. However, most of these interactions are insignificant in practice. Usually, the load in your car does not approach its carrying capacity; there is plenty of room for a few boxes of books; the books will not damage or be damaged by anything else in your car; their weight will not affect your fuel consumption; and so on.

A major simplification that would significantly reduce the complexity of the reasoning required to recognize opportunities would be to assume that the agent's various goals are *independent*, i.e., the pursuit of one goal does not in any way interact with the pursuit of any other goals. Unfortunately, this assumption would deny the possibility of recognizing those circumstances in which there are adverse interactions between goals. A weaker version of the independence assumption is to assume that an agent's goals are *effectively independent* of each other, i.e., that there are no *significant* interactions between them, unless there is evidence to the contrary (Pryor and Collins 1991). In our example, in the absence of any other indications you could assume that picking up the books would have no adverse effects on any other errands you might plan to run. In general, the assumption of effective independence enables the agent to ignore most interactions between goals and to concentrate on interactions that violate the assumption.

7.1.2 Heuristic filtering

The assumption of effective independence makes the decision about whether to pursue an opportunity much simpler, since all insignificant interactions with other goals can simply be ignored. The agent need reason only about those interactions that are likely to be significant: in particular, it need only reason about possible adverse interactions between goals. In order to use the assumption of effective independence to reduce the amount of reasoning about interactions, an agent must have a quick and easy method of spotting potential violations of effective independence. Having recognized that a goal is potentially easy to achieve, the next stage in determining whether there is a genuine opportunity for this goal is thus detecting any potentially adverse interactions between this goal and the agent's other plans.

The process of finding problematic interactions between goals typically involves projecting the future course of goal achievement and to analyze each interaction that is found to determine whether it is problematic (Ferguson 1992; McDermott 1992). Unfortunately, projection in an unpredictable world is no

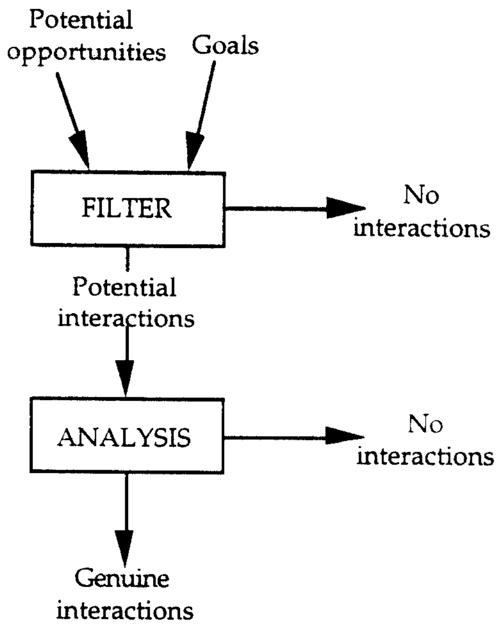


Figure 7.1 A filter for potential interactions

simple matter. In particular, it involves determining in advance the methods to be used to achieve goals in various possible circumstances and computing the probabilities of those circumstances arising. In other words, using projection is equivalent to planning in full detail for all possible contingencies. Projection is therefore not suitable as a method of recognizing adverse interactions for the purpose of recognizing potential opportunities.

An agent must therefore be able to recognize potential violations of effective independence without resorting to extensive analysis of all possible ways of achieving its goals. An effective approach to this problem is to use the regularities in the world to indicate likely interactions. This is possible because many objects consistently tend to interact with other objects in particular ways. For example, knives often cut other objects, wet paint leaves marks on anything it comes in contact with, and heavy objects tend to crease clothes. In general, an agent can use the functional regularities in the world to tag situation elements that are frequently involved in significant interactions. These tags are the *reference features* that were described in chapter 3. For instance, if some objects frequently cut other objects, the agent can take note of that fact and mark the cutting objects as *sharp* and the objects that are cut as *soft*. When it encounters a sharp object in conjunction with a soft object, the agent can recognize that cutting interactions are likely to take place.

We have already seen how reference features can be used in a heuristic filtering process to recognize goals that are potentially easy to achieve; a similar process can be used to recognize interactions that may constitute potential violations of effective independence.

The filtering process to detect potentially adverse interactions consists of three steps:

1. Find the situation elements involved in the potential opportunity and the other plans that the agent is pursuing;
2. Find the reference features of these situation elements;
3. Compare the reference features.

The situation elements that are relevant are those that are involved in a goal for which there is a potential opportunity and the goals with which it might interact. Returning to our example of collecting books from a friend, the books themselves are relevant, as are the other contents of the car, and anything else that you may be planning to collects as you run your errands. Note that the agent's goals themselves count as situation elements and may have reference features associate with them; for example, an *urgent* goal is likely to interact adversely with goals whose achievement is accomplished through actions that take a long time. If the filtering process is to be quick and easy, the agent must be able to determine which situation elements are relevant without using projection and without committing to a specific method of achieving its goals. PARETO does this by keeping track of the specific objects that are involved in the achievement of each goal as described in section 7.2.1.

In general, reference features label collections of functional effects; the functional effects being considered here are interactions. A reference feature indicates the type of interaction that the element to which it is attached may take part in, and the role that the element plays in the interaction. For example, a *sharp* object may be involved in an interaction in which it cuts something else; a *soft* object may be involved in a cutting interaction in which it is cut by something else. However, the presence of a *sharp* object is not on its own sufficient to indicate that a cutting interaction is likely, and nor is the presence of a *soft* object. Both types of object must be present for the interaction to occur. PARETO uses a simple characterization of interactions (described in section 7.2.2) in terms of the roles played by the interacting elements. This characterization ensures that only those interactions that might actually occur are flagged as likely violations of effective independence.

PARETO uses this filtering process to spot potentially problematic interactions. It then analyzes the indicated interactions in detail, and, if they turn out to be genuine, takes action to avoid them. For example, suppose PARETO has an order for something to carry tools in and another for a chisel. The truck is currently at tool-depot-2 where it observes one of each (see figure 7.2). However, there is a potential problem: if the bag is made of fabric, the chisel may puncture it and render it useless. PARETO recognizes this potential interaction through the use of reference features: a sharp object together with a fibrous object indicates a possible puncture interaction:

```
*** Potential PUNCTURE interaction:
    INSTRUMENT:     CHISEL (ITEM-25) is POINTED in DELIVER-OBJECT  [6:6]
    OBJECT:         BAG (ITEM-26) is FIBROUS in DELIVER-OBJECT  [7:7]
```

PARETO next adds a new task to analyze this potential interaction (see section 7.3.1 below):

```
+++ Adding decision task for PUNCTURE interaction  [7:14]
```

When this task comes up for execution, PARETO analyzes the situation and realizes that it needs to know what the bag is made of (section 7.3.2):

```
Analyzing PUNCTURE interaction for <DELIVER-OBJECT BAG ILS ORDER-1> ::  [7:7]
   Potential PUNCTURE interaction:
   INSTRUMENT:     CHISEL (ITEM-26) is POINTED in DELIVER-OBJECT  [6:6]
   OBJECT:         BAG (ITEM-27) is FIBROUS in DELIVER-OBJECT  [7:7]
   - answer:    NEED MORE INFO: WHAT IS THE BAG MADE OF?
```

The truck therefore inspects the bag, and discovers that it is in fact made of fabric:

```
+++ Executing action (EYE-EXAMINE TOOL-DEPOT-2 ITEM-27 TOP)
```

When the decision task next comes up for processing, PARETO realizes that there is a genuinely problematic interaction:

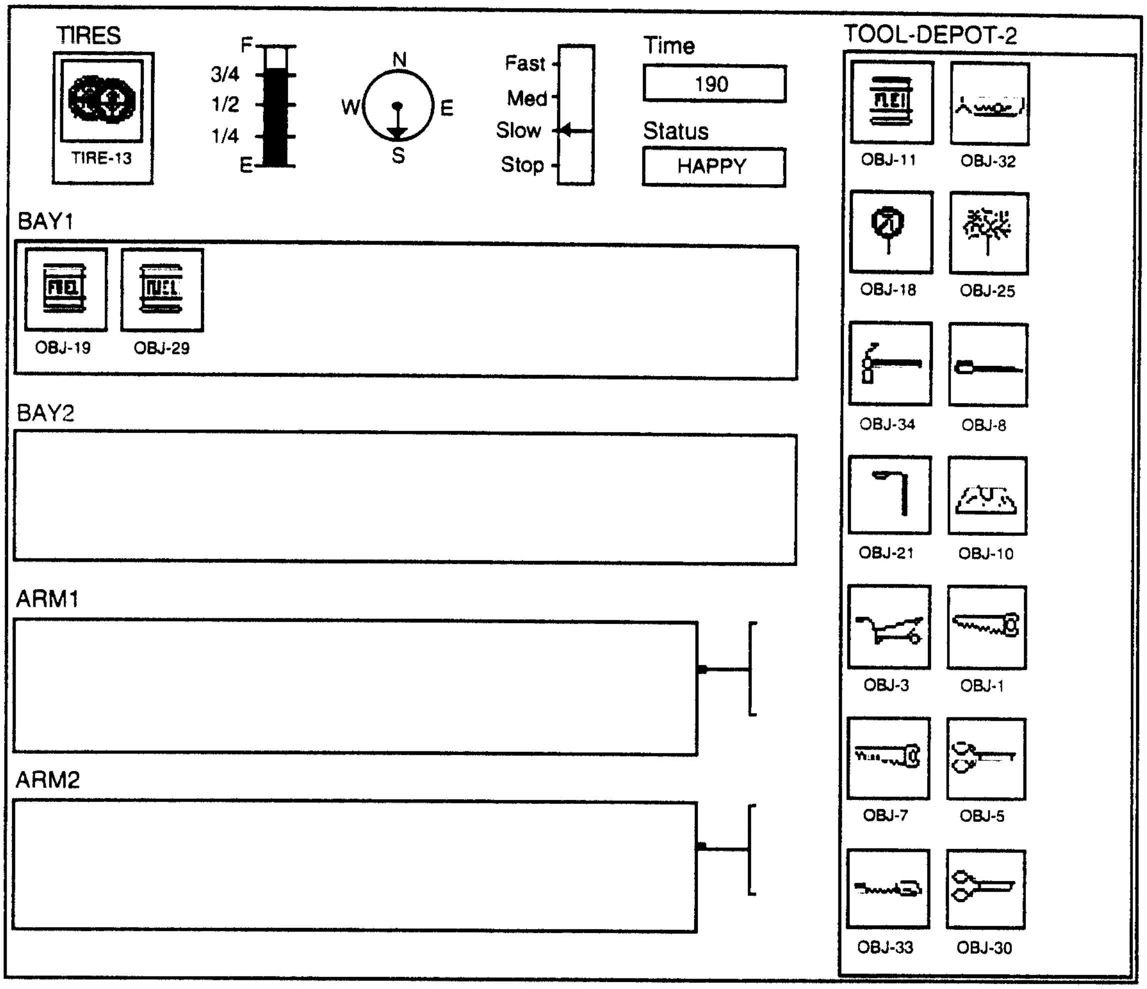


Figure 7.2 The truck finds a bag and a chisel

```
Analyzing PUNCTURE interaction for <DELIVER-OBJECT BAG ILS ORDER-1> :: [7:7]
   Potential PUNCTURE interaction:
   INSTRUMENT:     CHISEL (ITEM-26) is POINTED in DELIVER-OBJECT [6:6]
   OBJECT:         BAG (ITEM-27) is FIBROUS in DELIVER-OBJECT [7:7]
   - answer:   GENUINE INTERACTION: FABRIC BAG
```

It therefore postpones the task to deliver the bag until after the chisel has been delivered, in order to avoid the interaction (section 7.3.3):

```
      --> Postponing DELIVER-OBJECT [7:7] at 186 to avoid PUNCTURE interaction
```

In this example we have seen PARETO use a filtering process based on reference features, followed by a more detailed analysis to determine whether there is indeed a problematic interaction. Having determined that the interaction is problematic, PARETO takes action to avoid it. The next section will examine the issues that arise in the filtering process; section 7.3 examines the other stages.

```
(define-rap
    (index        (deliver-object ?criterion ?site ?order-id))
    ...
    (method
      (task-net
        (t1       (find-object ?type => ?place ?object)
                    (for t2))
        (t2       (load-payload-object ?object)
                    (for t3))
        (t3       (travel-to-site ?site)
                    (for t4))
        (t4       (unload-at ?site ?object)))))
```

Figure 7.3 Part of a RAP to deliver an object

7.2 Spotting interactions in PARETO

PARETO spots potentially problematic interactions using the filtering process outlined in the previous section. There are three issues that arise in the filtering process, one for each stage:

- Determining the situation elements involved in the goals under consideration;
- The reference features that indicate potential interactions;
- Using the reference features.

In this section, each of these issues is considered in turn.

7.2.1 Situation elements

PARETO detects interactions between tasks by examining the reference features of the situation elements involved in the tasks under consideration. The situation elements involved in the tasks are the equivalent of the *focus group* used in the recognition of goals that are potentially easy to achieve (see chapter 5). The question thus arises as to which situation elements are involved in a given task. As I shall explain in this section, there are three types of situation elements that may possibly be involved in a task: those in terms of which the task is specified, those that are involved in the task's parent task, and those (if any) whose presence constitutes a potential opportunity for the task.

First, the situation elements involved in a task must include those that are used to specify the task: for example, a task to collect some books from a friend clearly involves the books. However, even after you have started to pursue the task you don't necessarily know exactly which books you will be collecting. In order to allow for interactions involving the specific books in question (*e.g.*, one might turn out to be too big for the box you have brought to put them in) you must know which books they are. Some situation elements are thus identified in the task description (for example, a task to pick up a specific book), while others become identified as the task proceeds (for example, a task to collect whatever books your friend gives you).

These relationships between tasks and the situation elements that are used to specify them are represented in different ways in the RAP language that describes PARETO's tasks. Situation elements that are identified in the task description are those that are included in the task's input variables (see chapter 4). For example, the object to be loaded into a cargo bay appears as an input variable in the load-payload-object task. Situation elements that become identified as the task proceeds are included in the output variables of a subtask. For example, the deliver-object task shown in figure 7.3 does not specify the particular object to be delivered, but instead specifies the criterion that it must meet. The find-object subtask of the deliver-object task finds an object—a pair of scissors, say—that meets the criterion, and specifies the object in its output variables. That pair of scissors is the object that is to be delivered, and is specified in the input-variables of the load-payload-object and unload-at-site tasks.

The second way in which a situation element may be involved in a task is if it is involved in that task's parent. A situation element may be involved in the child task even when it is not directly specified. For example, suppose you are cooking some vegetables. You wash and prepare the vegetables, bring some water to the boil, and add the vegetables. The vegetables are involved in all these tasks, even though they do not explicitly appear in the specification of the task to bring some water to the boil. However, adverse interactions involving the vegetables can arise during the performance of that task: you should not add salt to the water, for instance, if you are cooking dried beans. The vegetables are involved in the boiling water task by virtue of their involvement in the overall task to cook vegetables.

A similar situation arises in PARETO's tasks to deliver objects. The object to be delivered forms part of the specification of the deliver-object task, as discussed above. One of the subtasks of the deliver-object task is a task to travel to the building site where the delivery is to be made. The actual object to be delivered is not specified in the travel-to-site task: the procedure for getting to the appropriate location does not explicitly depend on the object that is to be delivered. However, the delivery object is involved in the task, and adverse interactions can arise during the performance of the task. For example, suppose the truck is delivering a knife and has another goal to deliver a roll of wallpaper. If these two objects are carried in the same cargo bay, the knife might rip the roll of wallpaper, rendering the latter unusable.

The third way in which a situation element may be involved in a task is if it helps create a potential opportunity for the task. For example, suppose you are looking for something with which to lever off the lid of a can of paint. You see a fork which might do, but you are not sure whether its end is slim enough to fit between the can and the lid. At this stage the fork does not form part of the specification of the opening task, but if the opportunity is taken it will do so, and should certainly be considered as a source of possible interactions. For example, if it is the fork that you intend to use to eat your lunch, using it to open the can of paint would result in the adverse interaction of covering it with paint. However, not all situation elements that indicate an opportunity for a task will necessarily be involved in the achievement of that task even if the opportunity is taken. If a situation element indicates that a goal is potentially easy to achieve, but the analysis of the achievability of the goal does not involve that situation element, then the element is also unlikely to be involved in any interactions between that goal and any others.

PARETO's goals, as we have seen, are primarily involved with delivering physical objects. For PARETO the situation elements involved in a task are therefore the physical objects involved in that task. PARETO must have a method of determining what objects are involved in a given task (its *goal-objects*). This mechanism must be very efficient, or the advantage of using reference features is lost: it cannot, for instance, consist of reasoning about how the task will be achieved. The goal-objects of PARETO's tasks are therefore derived from the task specifications without considering how each subtask will be achieved. The goal-objects of a task reflect the three ways in which situation elements may be involved in a task, and comprise the following elements:

- Objects specified in the task's input variables;

Possible interaction	Conditions
A chisel PUNCTUREs a bag	The bag is made of cloth
A saw RIPs a roll of wallpaper	
A knife CUTs a ball of twine	The knife's blade is extended
A wheelbarrow FILLs a cargo bay, preventing the truck from carrying any other objects	The bay is already partly filled
A plank of wood SMEARs any other object	The wood is newly painted and is still wet
The truck MOVEs	

Figure 7.4 Some problematic interactions in PARETO's world

- Objects specified in the output variables of a subtask;
- The goal-objects of the task's parent;
- The object (if any) whose presence constitutes a potential opportunity for the task.

Consider, for example, the deliver-object task shown in figure 7.3. The input variables include no objects, so the task has no goal-objects from that source. The find-object subtask has an output-variable that specifies the object that is to be delivered—a pair of scissors, say. Once that subtask has been performed successfully, the scissors are included in the goal-objects of the deliver-object task. The scissors are also included in the goal-objects of the load-payload-object and unload-at tasks, in whose input-variables they appear. They are also included in the goal-objects of the travel-to-site task by inheritance from the deliver-object task.

Finally, suppose the goal-criterion of the deliver-object task is to find something to cut twine, which is met by either a pair of scissors or a knife. If the deliver-object task is not waiting for the completion of any other tasks, the presence of any *sharp* object would constitute an opportunity for the task. These objects (if any) are also included in goal-objects of the deliver-object task.

Whenever PARETO processes a task, it updates the goal-objects of that task and of its parent. A task's goal-objects are thus always readily available, allowing the first stage of the filtering process to proceed quickly and easily.

7.2.2 Reference features

The second stage in the filtering process is to find the reference features of the situation elements involved in the goals under consideration. The reference features that PARETO uses are tags for the types of problematic interaction that occur in its world. An interaction is problematic if it hinders the achievement of a goal. In PARETO's world these include interactions that destroy an object that is being delivered, interactions that make an object fail to meet delivery requirements, and interactions that make it impossible to achieve other tasks (see figure 7.4). For example, if a saw is carried in the same cargo bay as a roll of wallpaper, the saw may cut the wallpaper, which is thereby rendered useless. A newly-painted plank of wood may smear any other object with which it comes into contact. Any task that involves the truck's changing location may render other tasks unachievable because many tasks such as loading and unloading objects rely on the truck's being at the same location as the object being manipulated. The interactions in PARETO's world are described fully in appendix B.

Reference feature	Interaction	Role
colored	SMEAR	instrument
large	FILL	instrument
sharp	RIP	instrument
fabric	RIP	object

Figure 7.5 Reference features for interactions

PARETO's reference features tag situation elements that may be involved in interactions. For example, *sharp* objects may RIP or CUT *soft* objects, and *colored* objects may have wet paint and SMEAR other objects. However, a soft object always plays the same role in a RIP interaction: it is always the object that is damaged, rather than the one that does the damaging. Similarly, a can of solvent always STAINs a plank of wood, rather than the other way round. Reference features are tags for the role that the object plays in the interaction as well as the interaction itself. Some of the reference features that PARETO uses to indicate potential interactions are shown in figure 7.5; appendix B gives a full listing.

The way in which PARETO associates reference features with objects was described in chapter 5. Briefly, reference features are associated with properties of the object under consideration. However, PARETO uses only those properties of which it is aware. The only type of information acquisition that it performs in order to determine reference features is retrieval from memory. Reference features attached to goals are also readily available. The second stage of the filtering process thus involves minimal information acquisition.

7.2.3 Comparing reference features

The third stage in the filtering process is to compare the reference features of the task for which there is a potential opportunity (the *opportunity task*) with those of the other tasks the agent is pursuing. The purpose of the filtering process is to recognize potential violations of effective independence for a particular task for which there may be an opportunity. It is only interactions involving the opportunity task that are relevant to the determination of whether there is indeed an opportunity for it. The filtering process therefore indicates only the potential interactions involving the opportunity task.

In general, an interaction between goals may involve a number of situation elements. For example, suppose you put a box of books in your car which crushes the dry cleaning you have just collected. The books and the cleaning are clearly involved in the interaction, but so are other elements including the following: the car seat on which the cleaning and books were placed and which is one of the two surfaces between which the cleaning is crushed; the bend in the road that made the books shift on the seat; the bumps in the road that ; and so on. However, many of these situation elements are not by themselves sufficient to indicate the presence of a likely interaction. For example, the car seat is present whenever something is carried in the car, but there is not always an interaction involving crushing. However, the close proximity of a box of books and clothes often does result in the clothes being crushed: the clothes and books are usually supported by a surface against which the crushing can take place, and there is often some motion that brings the two into contact. This is an example of the *critical factor hypothesis* applied to interactions.

Interaction	Critical roles	Reference features
CUT	instrument object	sharp soft
RIP	instrument object	sharp fabric
PUNCTURE	instrument object	pointed fabric
SMEAR	instrument	colored
STAIN	instrument object	solvent colored
MOVE	mover	change-loc

Figure 7.6 Potential interactions used by PARETO

As it was described in chapter 3, the critical factor hypothesis states that the presence of a single factor is often crucial to the existence of an opportunity for a given goal, although there may many other factors that are involved in the achievement of that goal. Similarly, for any interaction between goals there are often only one or two factors whose involvement is crucial in the interaction. In our example, the books and the clothes supply the critical factors.

Many interactions have two critical factors. For example, a cutting interaction involves the object that cuts and that which is cut, a crushing interaction, as we saw, involves the object that crushes and that which is crushed, and so on. Some interactions, however, involve only one critical factor. Wet paint smears anything with which it comes into contact: the presence of wet paint is on its own sufficient to indicate that the interaction may occur. The critical factor hypothesis simplifies the filtering process for interactions enormously, just as it simplifies the filtering process for easily achievable goals. Instead of requiring all possible elements that might play a role in an interaction to be present, a potential interaction is indicated by the presence of only those elements that supply critical factors, or *critical elements*. In general, it appears that very few interactions have more than two critical elements, however many elements might be involved in the actual interaction.

Figure 7.6 shows some of the interactions that take place in PARETO's world. Each interaction is shown with the roles that must be filled by critical elements. For example, the *cut* interaction is indicated by two critical elements, one that will perform the cutting and one that will be cut. The move interaction, on the other hand, has only one critical element: the task in which a change of location arises. There are so many tasks that are affected by a change of location that it would be invidious to indicate a potential move interaction for each one of them, but there are comparatively few tasks that actually involve movement. The latter are thus reliable indicators of potential interactions. Whenever a task that may result in a change of location is considered for processing, there is a potential interaction with all other tasks that PARETO might perform.

The opportunity task is likely to be involved in a problematic interaction if both the following are true:

- The potential interaction is tagged by a reference feature of the opportunity task;
- Each of the potential interaction's critical roles is indicated by a reference feature of one of the tasks under consideration.

The filtering process indicates those potential interactions that have all their critical roles filled and that have not been analyzed previously and found to be harmless.

7.3 Detailed reasoning

The filtering process may indicate that the task in question is likely to be involved in a problematic interaction, but reference features are a simplified functional description of the situation elements to which they apply and not all indicated interactions are genuinely problematic. The indicated interactions must therefore be analyzed in more detail in order to determine their status. In general the analysis required to determine interactions between goals is complex and may involve the acquisition of arbitrary amounts of information. The use of reference features simplifies the analysis by indicating which of the agent's goals are likely to be involved in interactions. The goals not so indicated can be assumed not to be involved in any adverse interactions.

Reference features also indicate the form the analysis should take, by indicating the type of interaction in which the goal in question is potentially involved. For example, suppose an agent has one goal that involves a *sharp* object and another that involves a *soft* object. The use of the filtering process described in the last two sections would indicate not only that there is a potential interaction between these two goals but also that the potential interaction will involve the sharp object *cutting* the soft one. This is an important aspect of the use of reference features. Simply knowing that a goal my be involved in an interaction does not significantly reduce the amount of reasoning that need be performed. A given goal may in theory be involved in an enormous number of interactions, most of which are insignificant. The use of reference features allows the agent to focus its attention on just those interactions that are potentially problematic.

PARETO's architecture consists of three layers: the hardware control layer, the plan execution layer, and the reasoning layer. The reasoning layer is responsible for analysis of the type that is required to determine whether a potential interaction is genuinely problematic. However, the current implementation does not include a full reasoning layer. Instead, as discussed in chapter 4, there is a collection of special-purpose reasoning procedures that can be used when required. This collection includes procedures to analyze potential interactions. When the filtering process detects a potential interaction, an analysis task is added to the task agenda. For instance, in the example we saw earlier, PARETO recognizes a potential puncture interaction and then adds a new analysis task to the agenda:

```
*** Potential PUNCTURE interaction:
    INSTRUMENT:     CHISEL (ITEM-26) is POINTED in DELIVER-OBJECT  [6:6]
    OBJECT:         BAG (ITEM-27) is FIBROUS in DELIVER-OBJECT  [7:7]
+++ Adding decision task for PUNCTURE interaction  [7:14]
```

The analysis of a potential interaction first involves determining whether the interaction is genuinely problematic. This may involve the acquisition of information by any method, including perception. Lastly, if the interaction turns out to be genuine, the reasoning layer must decide how it can be avoided, and place the appropriate constraints on the task agenda so that the execution layer will process the tasks in the correct manner to ensure that the interaction will not occur. In the remainder of this section I describe how these three issues are addressed in PARETO.

7.3.1 Genuine interactions

There are two aspects to determining whether an indicated interaction is genuinely problematic: first, whether the interaction will actually occur and second, whether it is problematic.

Many indicated interactions will not actually occur. There are two ways in which the filtering process may indicate a potential interaction that, on analysis, turns out not to be an interaction at all. First, one of the goal-objects whose reference features are used by the filtering process may turn out not to be involved in the achievement of the goal. This happens when the goal-object in question is one that simply indicates the presence of a potential opportunity (as described in chapter 6), which itself turns out not to be genuine. For instance, in the example discussed above, there is a pair of scissors whose presence may constitute an opportunity for the goal of finding a chisel (the goal and the scissors both have the reference feature *sharp*). However, when PARETO analyzes the situation it finds that the scissors will not meet the goal criterion and therefore will not be involved in achieving the goal; the interaction will not occur.

```
Analyzing RIP interaction for <DELIVER-OBJECT BAG ILS ORDER-1> :: [7:7]
    Potential RIP interaction:
       INSTRUMENT:     SCISSORS (ITEM-28) is SHARP in DELIVER-OBJECT  [6:6]
       OBJECT:         BAG (ITEM-27) is FIBROUS in DELIVER-OBJECT  [7:7]
    - answer:   ITEM-28 NOT INVOLVED: NO INTERACTION
```

Second, the objects concerned may not interact in the way indicated by their reference features.[1] For example, PARETO's reference features indicate a RIP interaction when one task involves a *sharp* object and another involves a *soft* object. However, not all sharp objects actually rip soft objects. It is only saws that rip bags or rolls of wallpaper. Moreover, a bag is ripped only if it is made of fabric—leather bags are too tough. Similarly, a knife cuts only if its blade is extended, and a painted plank of wood smears other objects only if its paint is still wet. Suppose, for instance, that PARETO has the opportunity to pick up a bag when it is already carrying a knife. There is a potential interaction, because knives are sharp and bags are soft, but when it comes to the analysis stage PARETO realizes that in fact knives do not cut bags:

```
Analyzing CUT interaction for <DELIVER-OBJECT BAG ILS ORDER-1> :: [7:7]
    Potential CUT interaction:
       INSTRUMENT:     KNIFE (ITEM-27) is SHARP in DELIVER-OBJECT  [6:6]
       OBJECT:         BAG (ITEM-30) is SOFT in DELIVER-OBJECT  [7:7]
    - answer:   NO INTERACTION
```

Even if an indicated interaction will actually occur, it is not necessarily problematic. For example, a stain interaction occurs when a can of solvent comes in contact with a plank of wood. However, this matters only if the color of the wood is specified in the goal criterion; if it doesn't matter what color wood is delivered, it doesn't matter if it is stained.

```
Analyzing STAIN interaction for <DELIVER-OBJECT SOLVENT ILS ORDER-2> :: [6:6]
    Potential STAIN interaction:
       INSTRUMENT:     SOLVENT (ITEM-33) is SOLVENT in DELIVER-OBJECT  [6:6]
       OBJECT:         WOOD (ITEM-26) is PAINTED in DELIVER-OBJECT  [7:7]
    - answer:   INTERACTION NOT PROBLEMATIC: stained wood doesn't matter
```

[1] The conditions under which the interactions in PARETO's world take place are shown in appendix B.

7.3.2 Information acquisition

PARETO may need to acquire information during the analysis of a potential interaction. In general, there is no limit on the information that may be required or the methods by which it may be desirable to acquire it. In the architecture described in chapter 4, the reasoning layer itself can acquire information through inference and can retrieve information from memory. In order to acquire information by perception, however, the plan execution layer must be used to mediate between the reasoning layer that requires the information and the hardware controller that can perform the necessary perceptual actions.

When information that must be acquired through perception is required, an appropriate task is added to the task agenda and the analysis suspended until the perception has been performed. For example, in order to determine whether a smear interaction will actually take place, PARETO must find out whether the paint on the plank of wood that is suspected of being involved in the interaction is actually wet. This information must be acquired through perception. In the following output, PARETO directs the truck to determine whether the paint is wet on the piece of wood being considered for a delivery:

```
Analyzing SMEAR interaction for <DELIVER-OBJECT WOOD ILS ORDER-1> :: [7:7]
    Potential SMEAR interaction:
        INSTRUMENT:   WOOD (ITEM-26) is PAINTED in DELIVER-OBJECT  [7:7]
    - answer:     NEED MORE INFO: IS THE PAINT WET?
```

When the truck examines the wood, it finds that in fact the paint is dry. PARETO then analyzes the potential interaction again, and decides that it will not occur:

```
+++ Executing action (EYE-EXAMINE PAINTSHOP ITEM-26 TOP)
Analyzing SMEAR interaction for <DELIVER-OBJECT WOOD ILS ORDER-1> :: [7:7]
    Potential SMEAR interaction:
        INSTRUMENT:   WOOD (ITEM-26) is PAINTED in DELIVER-OBJECT  [7:7]
    - answer:     PAINT DRY: NO INTERACTION
```

7.3.3 Avoidance strategies

When analysis indicates that an interaction is genuinely problematic, PARETO must decide what to do about it. Again, this is the job of the reasoning layer which is at present unimplemented. Instead, there is a collection of special purpose procedures. There are currently four simple strategies that PARETO can use to handle problematic interactions:

- Postpone one of the tasks involved in the interaction until the other has finished;
- Delay one of the tasks for a specific period of time ;
- Perform actions to prevent the occurrence of the interaction;
- Note that the interaction is unavoidable.

In general, strategies to handle interactions could be highly contingent and involve arbitrarily complex changes to the tasks on the task agenda. McDermott (1992) describes some of the transformations that are possible.

In the rest of this section we will look at an example of each of PARETO's four strategies. The one that arises most commonly in practice is to postpone one of the tasks involved in the interaction until the other has finished. This can be thought of as the sledgehammer strategy: it does not address the cause of the interaction in any way, but prevents it from taking place by preventing the interleaving of the actions required to achieve the tasks. The postponement is accomplished by adding the relevant constraints to the tasks on the task agenda. For example, when PARETO analyzes a potential PUNCTURE interaction between a chisel and a bag and discovers that it will indeed be problematic, it postpones the task to deliver the bag until after the chisel has been successfully delivered:

```
Analyzing PUNCTURE interaction for <DELIVER-OBJECT BAG ILS ORDER-1> :: [7:7]
    Potential PUNCTURE interaction:
       INSTRUMENT:     CHISEL (ITEM-26) is POINTED in DELIVER-OBJECT [6:6]
       OBJECT:         BAG (ITEM-27) is FIBROUS in DELIVER-OBJECT [7:7]
    - answer:    GENUINE INTERACTION: FABRIC BAG
    --> Postponing DELIVER-OBJECT [7:7] at 187 to avoid PUNCTURE interaction
        [7:7] must wait for DELIVER-OBJECT [6:6]
```

When a task has been postponed in this way, it becomes inactive. It is not reactivated until the task for which it is waiting is removed from the task agenda. An inactive task is not considered when PARETO looks for potential opportunities (see chapter 6).

PARETO's second strategy is to postpone one of the tasks for a fixed period of time. This is slightly more focused than the first strategy, but is rarely applicable. In PARETO's world, the only type of interaction that is affected by the passing of time is the SMEAR interaction, in which a newly painted piece of wood may smear other objects with which it comes into contact. However, paint only stays wet for a limited period before it dries. When PARETO encounters one of these interactions, it therefore delays the task involving the wood for 50 time units:

```
Analyzing SMEAR interaction for <DELIVER-OBJECT WOOD ILS ORDER-1> :: [7:7]
    Potential SMEAR interaction:
       INSTRUMENT:   WOOD (ITEM-18) is PAINTED in DELIVER-OBJECT [7:7]
    - answer:    GENUINE INTERACTION: PAINT IS WET
    --> Delaying DELIVER-OBJECT [7:7] at 144 for 50 time units to avoid SMEAR interaction
```

This delaying strategy also results in the inactivation of the delayed task, which is not considered when PARETO looks for potential opportunities.

The third strategy that PARETO uses is to perform some action that will prevent the interaction from taking place. For example, a knife will cut twine if ~~the~~knife's blade is extended: the interaction can be prevented by retracting the blade.

```
Analyzing CUT interaction for <DELIVER-OBJECT TWINE ILS ORDER-2> :: [6:6]
    Potential CUT interaction:
       INSTRUMENT:     KNIFE (ITEM-22) is SHARP in DELIVER-OBJECT [7:7]
       OBJECT:         TWINE (ITEM-25) is SOFT in DELIVER-OBJECT [6:6]
    - answer:    GENUINE INTERACTION: BLADE EXTENDED
                 RETRACT BLADE TO AVOID INTERACTION
    --- Retracting knife blade to avoid CUT interaction at 178
```

Next time the potential interaction is analyzed, it is found not to be genuine:

```
Analyzing CUT interaction for <DELIVER-OBJECT TWINE ILS ORDER-2> :: [6:6]
    Potential CUT interaction:
       INSTRUMENT:     KNIFE (ITEM-22) is SHARP in DELIVER-OBJECT [7:7]
       OBJECT:         TWINE (ITEM-25) is SOFT in DELIVER-OBJECT [6:6]
    - answer:    NO INTERACTION: BLADE RETRACTED
```

In general, preventing an interaction from occurring is a common strategy in everyday life. You can prevent fragile objects from getting smashed as you move them by packing them safely, for example, and can prevent a heavy box of books from crushing the cleaning you have just collected by moving one of them to a different part of the car.

The final strategy that PARETO uses is to note that the interaction is unavoidable. The task selector can then take the interaction into account when choosing the next task to be processed. The strategy is appropriate when the task will have to performed at some stage, and when its execution will undoubtedly interact with other tasks. In PARETO's world, this happens with tasks that involve a change of location:

```
Analyzing MOVE interaction for <TAKE-OBJECT-TO-SITE ITEM-26 ILS> :: [6:9]
    Potential MOVE interaction:
        INSTRUMENT:   TAKE-OBJECT-TO-SITE is CHANGE-LOC at TOOL-DEPOT-2  [6:9]
    - answer:     CHANGE LOCATION
*** Goal always interacts: TAKE-OBJECT-TO-SITE [6:9]
```

PARETO's options in dealing with problematic interactions are limited to these four simple strategies. There are many more that are possible and that we ourselves use in our everyday lives. Perhaps the most obvious omission is that PARETO is unable to rule out a particular object for the furtherance of a given goal. For example, we saw above how PARETO can delay the pursuit of a goal in order to avoid an interaction in which the wet paint of a piece of wood would be SMEARED; an obvious alternative would be to avoid the use of that particular piece of wood, and instead use one with dry paint. This is currently beyond PARETO's capabilities, and is the subject of future work.

7.4 Summary

In this chapter I have described a two stage process for determining whether the pursuit of a goal would result in adverse interactions with other goals of the agent. The first stage uses a filter based on reference features to indicate potential interactions. The filtering process involves minimal reasoning and no explicit acquisition of information. The agent using it can then go on to analyze the indicated interactions in more detail to determine whether they are genuinely problematic. The analysis may be arbitrarily complex, but is it simplified by the use of reference features. The filtering process indicates the specific situation elements that are potentially involved in interactions, and the particular type of interaction that is likely. Reference features thus focus the agent's reasoning.

The second stage in this process is to analyze the interactions indicated by the first stage as being potentially problematic. I described how PARETO performs this analysis, and the simple strategies it uses to avoid those interactions that turn out to have adverse consequences. In some cases this analysis requires the acquisition of information to support the decision of how to handle the circumstances that have led to the potential opportunity.

The last chapter described how to use reference features to recognize easily achievable goals, and this chapter has described how to use them to spot problematic interactions between goals. They thus form the basis of a mechanism for recognizing opportunities, as shown in figure 7.7. First, the filtering process described in chapter 6 is used to indicate those goals that are likely to be easy to achieve. The indicated goals are then analyzed to determine whether they are indeed potential opportunities. Next, the filtering process described in this chapter is used to indicate potentially problematic interactions. Finally, the indicated interactions are analyzed in more detail. The two filtering processes both rely on reference

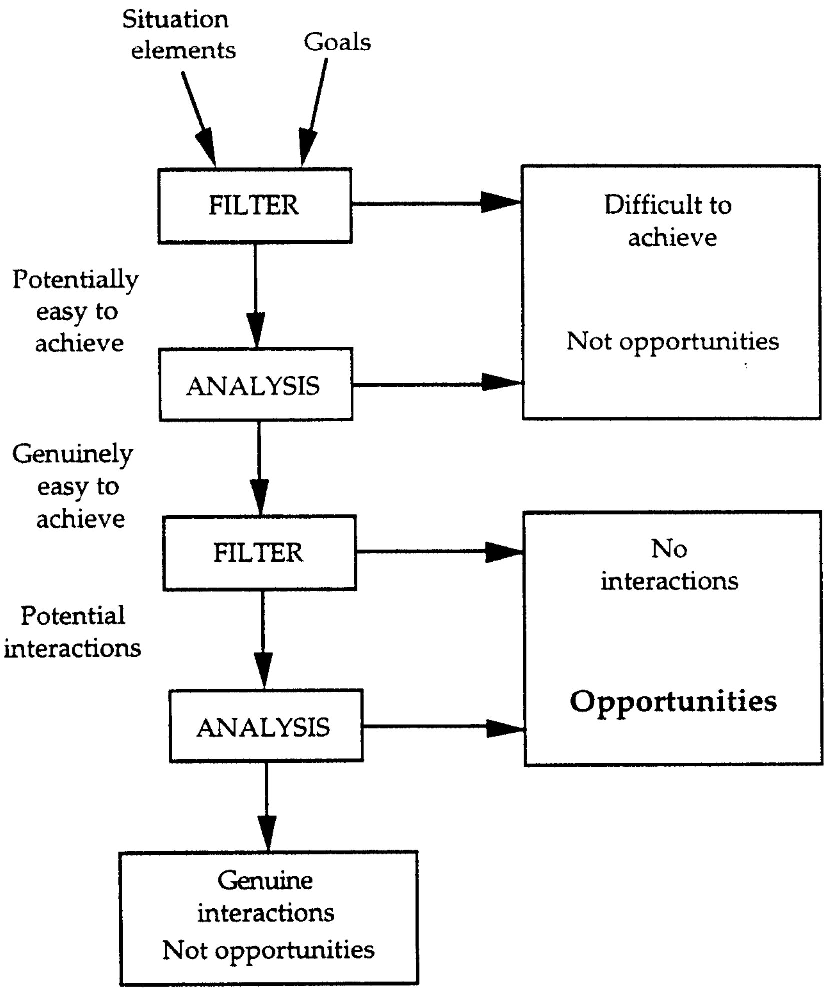


Figure 7.7 Recognizing opportunities

features, which are also to used to focus the second analysis stage. The two filters and the analysis of achievability all involve minimal reasoning and little explicit acquisition of information.

An agent operating in an unpredictable world will often encounter unforeseen situations. Some of these situations may require the agent to decide how to adapt its plans in the light of the new circumstances, while others have no effect on the agent's goals and require no decisions. If the agent is to respond in a timely manner to the changes in its dynamic and unpredictable environment it must be able to distinguish those unexpected situations that affect its goals, and in which decisions must therefore be made about how to pursue its goals, from those that have no effect on its goals and do not require decisions. In these two chapters I have described how this recognition can take place quickly and easily through the use of reference features. In the next chapter I describe how PARETO uses opportunities as the basis of its task selection mechanism.

CHAPTER 8

DECIDING WHAT TO DO NEXT

8.1 Introduction

An agent operating in an unpredictable world will frequently encounter unforeseen situations. If these situations affect its goals, it must respond appropriately by adapting its plans to the new circumstances. In other words, the agent must make unforeseen decisions. I have argued in previous chapters that an effective way of recognizing the need for these decisions is to recognize unforeseen opportunities, and that the planning process can itself be seen as a process of predicting opportunities. An agent that can recognize unforeseen opportunities and handle them in the same way that it handles the opportunities that have been predicted in its plans will thus be able to cope effectively in an unpredictable world.

Chapter 3 described a mechanism based on *reference features* that would enable an agent to recognize a large class of opportunities, and chapters 6 and 7 described how this mechanism is implemented in PARETO, and how PARETO also uses reference features to recognize problematic interactions between its goals. So far, then, we have seen how PARETO determines what opportunities there are to further its goals. In this chapter, I shall explore the issues involved in integrating the opportunity recognition mechanism into the agent's procedure for deciding what to do next, and describe how PARETO decides which of the available opportunities should be pursued.

In general, an agent will have many goals that it could pursue, and may detect opportunities for a number of them. It must constantly choose which of these goals to pursue (Brand and Birnbaum 1990; Goodwin and Simmons 1992; Hexmoor and Nute 1992). In the next section I discuss the characteristics of the behavior that should be produced by the choice mechanism. In section 8.3 I describe PARETO's task selector, showing how behavior meeting the desiderata specified in section 8.2 is produced with minimal recourse to deliberation and explicit information acquisition, and discuss the role that reasoning could play in this process.

8.2 Behavior patterns

An agent that behaves reasonably in an unpredictable world is one that responds appropriately to the situations that it encounters. Appropriate responses are those that appear to have taken the unexpected circumstances into account without ignoring the agent's existing plans. In theory, an agent could decide what to do next by choosing the option with the highest expected utility; in practice the calculation of expected utilities is extremely complex. Indeed, although there is some agreement on the types of factors that should be taken into account in performing utility calculations, there have been few proposals as to the method of accounting for these factors, many of which appear to be rather nebulous and are certainly difficult to quantify.[1]

The approach taken in the design of PARETO is therefore rather more qualitative in nature. In this section I describe four factors that should influence the decision of what to do next, and in the next section I describe the mechanisms in PARETO's task selector that produce behavior that takes these factors into account. These four factors can be expressed as desiderata for reasonable behavior:

- Concentrate on important goals;
- Don't miss opportunities;
- Don't interrupt tasks unnecessarily;
- Don't get side tracked and fail to return to the original task.

The order in which these factors are listed is not significant; they must all be taken into account when making a decision. In general, no single factor can be taken as being more crucial than the others; the balance between them varies according to the circumstances.

8.2.1 Important goals

An agent must have some method of judging which are its most important goals. These are the goals for which there is a very large difference between the utility of achieving them and the utility of failing to achieve them. A large difference can of course occur because either the consequences of achieving a goal are very beneficial, or the consequences of failure are disastrous, or a combination of the two. It is often appropriate to pursue an important goal; the opportunity cost of not doing so may be very high.

Many *preservation* goals (Schank and Abelson 1977)are very important to the agent that has them, and should be given high priority when deciding what to do next. For example, the consequences of failing to achieve the preservation goal to avoid injury may be extremely serious. In particular, if an agent is injured it may be prevented from achieving any of its other goals. Sometimes, of course, other considerations outweigh the potential for injury; if hurting yourself is the only way to stay alive, for instance, or if there is a strongly held principle that you believe you should support.

Other important goals are straightforward achievement goals. Many graduate students place a high priority on finishing their dissertations, for example, and a large part of society considers the goal to increase their net worth extremely important. Even at a more concrete level goals vary in importance. It may be more important to catch a train than to stop and talk to a friend, for instance, or to go to a movie with friends than to watch a television program.

Sometimes, however, there may be a good reason for choosing to pursue a goal other than the most important. Often, a short delay will not affect the achievement of a goal and will enable the agent to pursue some other goal for which there is a limited opportunity.

8.2.2 Seize opportunities

The second characteristic of reasonable behavior is that opportunities are seized when the cost of doing so is low. For example, unless you are rushing to catch a train you are quite likely to stop to chat with a

[1] The work of Haddawy and Hanks (1990; 1992) is the most notable step in the direction of quantification.

friend you meet on the street; if you see a five dollar bill lying on the sidewalk, you will pick it up; and so on.

In particular, it is often appropriate for an agent to change its plan for its current goal if a suitable opportunity presents itself. For example, suppose a robot is navigating through some woody terrain of which it has only an incomplete map. It planned its route on the basis that there was a belt of trees across the most direct line to its destination; as it approaches the area, it sees that there is in fact a passage through the trees. It should adapt its plan by abandoning its planned route and instead taking the short cut.

Of course, there are situations in which it is inappropriate to take advantage of an opportunity; if you see a restaurant whose menu you want to look at, but are driving through very heavy rush hour traffic, for example, it would probably be inappropriate to interrupt your current task.

8.2.3 Continuity

It is frequently undesirable to interrupt a task that is being executed. For example, if an assembly robot is in the middle of assembling one component, a widget, say, when the parts for the next component arrive, in most circumstances it would be inappropriate for it to interrupt the widget assembly to assemble the new component. In particular, given that there are opportunities for both tasks, it should probably not interleave them. In theory it would be possible for it to reach out its arm for a widget part, then for a part for the other component, and so on, without ever picking up either part. This would clearly be counterproductive. In general, it is desirable behavior to stick to the current task.

There are, of course, situations in which interruptions are desirable, especially if an opportunity arises for a very important goal, or one that is extremely time critical. For example, if you see a five dollar bill on the sidewalk on the way to the bus stop in the morning, it is most unlikely that it will still be there when you return in the evening. You should interrupt your walk in order to pick it up. Similarly, if the fire alarm rings when you are in the shower you would be wise to abandon your plans for cleaning yourself; on the other hand, if the telephone rings you may well decide not to answer it.

8.2.4 Focus

In general, it is often important to ensure that attention returns to the original task after an interruption. Clearly, it is important to distinguish between cases in which opportunities should be grasped and cases in which doing so would lead to sidetracking. For example, suppose you are in the supermarket looking for orange juice. Before you find it, you spot the bread at the end of the next aisle, and go off to get a loaf. This process continues until you have nearly all the items on your shopping list, except the orange juice. At this point you find yourself near a checkout point with nobody waiting, but you should ignore this unusual opportunity. Instead you should return for the orange juice that you have so far failed to get.

8.3 PARETO's choice mechanism

PARETO decides what to do next by deciding which of the tasks on its task agenda to process next. Some of these tasks may be ruled out of consideration because they have been deactivated, as explained in chapter 7; others may be waiting for other tasks to complete, or specific situations to obtain. However, there are usually many tasks that could be processed next, and PARETO must choose between them. The mechanism that it uses to make this choice meets the requirements described in the previous section, and is based on the notion of *opportunities;* whenever PARETO must choose a task to process, it chooses from among those tasks for which there are good opportunities. As I discussed in chapter 3, there are two kinds of opportunities: those that were predicted in the construction of the plans that PARETO executes; and those that were not predicted, which must therefore be recognized on the fly. PARETO uses the filtering mechanism described in chapter 6 to recognize potential opportunities in the form of tasks that are easily achievable, and that described in chapter 7 to recognize which of these opportunities are genuine, because they do not have problematic interactions with other tasks.

In fact, the problem is a little more complex than simply choosing from among these opportunities. As I explained in chapter 7, PARETO uses a filtering process to recognize potentially problematic interactions, which must then be analyzed further in order to determine whether they are genuinely problematic. This analysis may be complex and involve the acquisition of arbitrary amounts of information. PARETO performs this analysis by setting up a task to do so and adding this task to its task agenda, so there are often potential opportunities on the task agenda that are waiting to be analyzed. In this case the tasks to analyze them are also on the task agenda. PARETO must take this into account when deciding which opportunity to take next.

PARETO's choice mechanism is very simple, and involves little explicit reasoning about the consequences of any given choice. In particular, the reasoning layer is not used, except inasmuch as possibly problematic interactions are analyzed. The simplicity of PARETO's mechanism is both a strength and a weakness. It is a strength because it means that PARETO can respond quickly to changes in the world, without analyzing all the possible consequences in detail. It is a weakness because it means that PARETO has no mechanism for analyzing the possible consequences in detail. For instance, PARETO does not choose which delivery goal to address next on the basis of how close the truck is to a suitable object. Indeed, PARETO cannot take factors such as this, or any other factor based on explicit reasoning as to how the goal might be accomplished, into account. The investigation of ways in which such factors could be included in the choice mechanism without over-complicating it and losing the advantages of simplicity is an area of future work.

The remainder of this section describes PARETO's choice mechanism in some detail. First, there is a recapitulation of how PARETO's task agenda works, and the form that opportunities take. Section 8.3.2 explains how PARETO keeps track of the goals towards which it is working, and section 8.3.3 describes how priorities are assigned to tasks.

8.3.1 The task agenda and opportunities

PARETO decides what to do next by choosing a task from its task agenda.[2] However, not all tasks on the task agenda are eligible for processing. In fact, only those tasks for which there are opportunities are considered when it comes to making the choice. Recall from chapter 6 that PARETO uses a filtering process to recognize potential opportunities: reference features are used to indicate those tasks that are potentially easy to achieve, and the indicated tasks are then analyzed to determine whether they are

[2] See chapter 4 for a description of PARETO's execution cycle.

```
(define-rap
   (index          (monitor-current-location))
   (succeed      (not T))
   (monitor-state    (or (not (believe (eye-scanned external) 15))
                             (not (scanned-since-move external))))
   (method
      (task-net
         (t1 (eye-scan-p external)))))
```

Figure 8.1 A RAP to scan the truck's surroundings

indeed easily achievable. Tasks that are ineligible for processing are eliminated from this process at two different stages, depending on the reason for their ineligibility.

First, *inactive* tasks are not considered at all when PARETO is looking for opportunities. Tasks may become inactive in two ways, both methods of avoiding problematic interactions, as described in chapter 7: a task may be postponed until after the successful completion of another task, or it may be delayed for a specific period of time.

Second, there are tasks that should only be processed under certain circumstances; if those circumstances do not obtain, the task is ineligible for processing. The *monitor* clause in the RAP specification is used to indicate such circumstances. For example, the task to scan the truck's surroundings should only be processed if at least 15 time units have passed since the last scanning, or when the truck has just arrived at a new location. The monitor clause for this RAP is shown in figure 8.1. If the monitor clause of the RAP associated with a task is not met, then the task is found to be not easily achievable during the analysis phase.

PARETO distinguishes between the two types of opportunity described in chapter 3, *predicted* opportunities and *unexpected* opportunities, according to the way in which a task is easily achievable. From PARETO's perspective, an opportunity is *predicted* if it is for a task that is easily achievable because it is ready for processing, *i.e.*, one that is not waiting for any other tasks to complete. Such a task is the next step in one of PARETO's plans; as explained in chapter 3, planning can be seen as a process of predicting future opportunities. PARETO considers a task that has already succeeded, and that is not the next step in any of its plans, to represent an *unexpected* opportunity.[3]

8.3.2 Main and side tasks

In order to avoid being sidetracked by every opportunity that it encounters, PARETO keeps track of its *main task*, to the pursuit of which it returns after taking advantage of an opportunity. It also has a *side task*, which is often an opportunity that it has taken. PARETO will in general address its main task only if there are no feasible side tasks, but applies stricter feasibility criteria to side tasks than it does its main task. It will not address a side task if doing so would interfere with any other tasks, but is prepared to address the main task under such circumstances. The effect of this is that PARETO will take advantage of opportunities, treating them as side tasks, but as soon as pursuing them might divert attention from its main task it abandons them. This mechanism thus helps PARETO to seize opportunities while maintaining focus on the task that it has deemed to be currently most important.

[3] The details of the achievability analysis are given in section 6.4.

For instance, in the program trace shown in appendix C, at time 404 PARETO is addressing the task of finding some wallpaper with which to fulfill order-2. It arrives at warehouse-3 at time 436, and immediately switches to the monitor-current-location task. Next, at time 440, attention is switched to the task of fulfilling order-3: PARETO spots some tape, which is loaded into the truck at time 465. PARETO now spots some screws, which constitute an opportunity for order-4; these are loaded into the truck at time 482. After loading the screws, the next task to be addressed is that of fulfilling order-5, prompted by the presence of a pair of scissors. PARETO analyzes a possible cut interaction involving the scissors, decides that it is not problematic, and the scissors are loaded into the truck at time 495. Next, PARETO considers a possible opportunity to find a knife for order-6, but it turns out not to be genuine. At time 502, attention is switched to some twine, which constitutes an opportunity for order-7; it is loaded into the truck at time 512. There are no more opportunities at this location, so at time 512 PARETO returns to the original task of looking for some wallpaper. While the truck has been at warehouse-3, PARETO has addressed five delivery tasks other than the one that brought it to this location; it has also addressed the monitor-current-location task on four separate occasions (omitted from the above description for clarity). However, at the end of this episode attention is returned to the task of looking for wallpaper: PARETO does not abandon this in favor of going off to deliver any of the four objects that were loaded into the truck.

In general, the main and side tasks are changed in three circumstances: when the previous top-level task has been achieved successfully, when there is no further feasible subgoal for the previous task, and when the previous task is superseded by another that PARETO deems to be more important. There is little to say about the first: when a task has been completed, PARETO must choose another to be addressed.

The episode described above includes several instances of the second: all the tasks that were addressed, other than that of finding wallpaper, were side tasks. As soon as the next step for any of these tasks involved changing locations, and thus problematic interactions with other tasks, the side tasks were abandoned as having no further feasible subgoals. A main task can be abandoned for similar reasons: for example, if the next step is inactivated in order to avoid an interaction, PARETO must choose a new main task.

Third, a task may be superseded by another. One particular instance of this is a frequent occurrence: the monitor-current-location task is considered extremely important by PARETO, and is addressed whenever it is eligible for processing (immediately after arriving at a new location, and whenever more than 15 time units have elapsed since it was last addressed). Another task that is often considered important enough to supersede other tasks is that of refueling the truck when it runs low.

PARETO's mechanism for deciding whether a task that needs to be addressed should be the main task or merely a side task is simple. The new task is the side task except in the following cases:

- There is no current main task, because the previous one has been abandoned;
- The new task has a higher priority than the current main task;
- The new task is for a preservation goal.

The next section describes the latter two cases in more detail.

There is one case in which the new task becomes a side task even if there is no current main task: some tasks are noted as being specifically *interrupt* tasks, and they can never become main tasks. The most notable task of this type is the monitor-current-location task.

8.3.3 Task priority

Some goals are more important to PARETO than others. For example, preservation goals (Schank and Abelson 1977), such as those to ensure an adequate fuel supply and to keep track of the truck's surroundings, cannot be neglected in favor of delivery tasks. If the truck is disabled, whether through running out of fuel or any other mishap, all goals will become unachievable. PARETO's task selector always makes sure that preservation goals take precedence over other tasks.

In addition, some delivery goals may have a higher priority than others. This is handled by assigning a numeric measure of priority to each task. A task with a higher priority may supplant one with a lower priority.

One of the pitfalls of a system of priorities is that a goal with higher priority, whether or not it is a preservation goal, will interrupt another task. Sometimes such interruptions are appropriate: for example, the monitor-current-location task should always interrupt the task that is currently being addressed. At other times, however, interruptions are unnecessary and would lead to a lack of continuity. For example, even if the truck is running low on fuel it can load up objects that are at its current location. There is a danger that the supplanting of lower priority tasks will lead to a lack of continuity. PARETO avoids this danger by allowing such supplanting to take place only when the task would in any case change, except if the supplanting goal is explicitly known to be urgent, in which case the supplanting takes place immediately.

In this section and the last we have seen that PARETO must have information about general ways in which tasks interact with other tasks: this is accomplished by attaching annotations to the RAP descriptions. The possible annotations are the following:

- *Interrupt*: this task should never be PARETO's main task;
- *Preservation*: this task should take priority over all other tasks, and unless it is an interrupt task, should always be a main task rather than a side task;
- *Urgent*: this task should be addressed immediately it becomes feasible.

These annotations are, in effect, reference features. They describe *functional* aspects of the tasks, and allow PARETO to use a simple task selection mechanism instead of performing a detailed analysis of the implications of addressing a given task. Their use in this way, like their use in the filters described in earlier chapters, is not foolproof: it may not, for instance, always be the best decision to make a preservation task the main task instead of a side task, and it may well be that occasionally a brief delay before addressing an interrupt task would not be inappropriate.

8.4 Task selector

This section describes the action of PARETO's task selector in some detail. There are several functions that must be performed: PARETO must choose new main and side tasks, and must decide which task should be processed. The top-level algorithm is shown in figure 8.2. First, PARETO chooses a new main task if it does not currently have one. If it is unable to find a new main task, execution halts: there are no tasks that can be processed. Next, it looks for a task that should supplant the current main task, then looks for a task that should supplant the current side task (if any). Finally, it chooses a task to process from among the available opportunities. In the next few sections we consider each of these stages individually.

```
Select-task(main, side, opportunities)
    If there is no main task
        Choose a new main task
    Endif
    If there is no main task
        Exit
    Else
        Find a task, if there is one, that should supplant the current main task
        Find a task, if there is one, that should supplant the current side task
        Choose a task to process
        Return the chosen task
    Endif
```

Figure 8.2 Task selection algorithm

8.4.1 Choosing a main task

PARETO chooses a new main task in two circumstances: first, when it currently has no main task because the previous one either completed or had no further feasible steps; and second, when it chooses a task to supplant the current main task.

PARETO always makes sure that it has a main task. This means that when its current main task completes, it must choose a new one. It chooses according to the following order of preference:

1 The parent of the completing task;
2 Any decision task;
3 Any unexpected opportunity;
4 Any predicted opportunity;
5 The next step for any task that is involved in a problematic interaction.

The reasons behind this order of preference are as follows:

1 Choosing the parent of the completing task ensures continuity. PARETO's main task need not be a top-level task; if it is not, it makes sense to continue with the same top-level task when the current task completes.

2 Choosing a decision task helps to keep future options open. Decision tasks are placed on the task agenda in order to reason about interactions; there may therefore be two or more tasks that are waiting for a decision task to complete. Processing decision tasks thus in general increases the number of tasks that are eligible for processing.

3 Choosing an unexpected opportunity ensures that such opportunities are seized when possible. Recall that an unexpected opportunity is a task whose success criterion has been met, whether or not it has actually been processed and its subtasks completed. PARETO should take advantage of such opportunities.

4 Choosing a predicted opportunity ensures that tasks that are not involved in problematic interactions are processed in preference to those that are.

5 A next step that is involved in a problematic interaction is chosen only when none of the other options are available. Sometimes interactions cannot be avoided.

PARETO's method of choosing a new main task thus supports many of the criteria for reasonable behavior that were described earlier: in particular, it supports the seizing of opportunities and continuity of focus.

PARETO supplants its current main task if it can find a supplanting task for which the following are true:

1 The supplanting task is eligible for processing;

2 The supplanting task is not an interrupt task;

3 Either The supplanting task is a preservation task and the current main task is not;

Or Both the supplanting task and the current main task are preservation tasks and the supplanting task has a higher priority than the current main task;

Or Neither the supplanting task nor the current main task are preservation tasks and the supplanting task has a higher priority than the current main task;

4 There is no other task that should supplant the supplanting task.

This method of finding a supplanting task ensures that PARETO gives preference to addressing its most important goals.

8.4.2 Choosing a side task

PARETO uses its side task to enable it to seize opportunities without losing track of the main task that it is pursuing. It looks for a new side task when the current one has been abandoned, either because it completed successfully or because the next step for the task is not currently eligible for processing, and when there is no non-interacting step for the current main task (see next section). The procedure for finding a new side task is similar, but not identical, to that for finding a new main task:

1 The parent of the previous side task;

2 Any unexpected opportunity;

3 Any predicted opportunity;

4 Any decision task.

The differences are important. First, both unexpected and predicted opportunities are considered before decision tasks. This is because the purpose of the side task mechanism is to take advantage of opportunities: genuine opportunities are therefore preferred to the potential opportunities represented by decision tasks. Second, an interacting next step is never considered as a possible side task. Again, this is because such a task is never an opportunity.

PARETO also checks to see whether the current side task should be supplanted. The current side task is supplanted if PARETO can find a supplanting task for which the following are true:

1 The supplanting task is eligible for processing;

2 The supplanting task is an interrupt task;

3 Either The supplanting task is a preservation task and the current side task is not;

Or Both the supplanting task and the current side task are preservation tasks and the supplanting task has a higher priority than the current side task;

Or Neither the supplanting task nor the current side task are preservation tasks and the supplanting task has a higher priority than the current side task;

4 There is no other task that should supplant the supplanting task.

Except for the fact that the supplanting task must be an interrupt task, this method is the same as that for finding a task that should supplant the current main task.

8.4.3 Selecting a task

Once PARETO has found main and side tasks, it chooses a task for processing. It chooses from among the eligible tasks the task agenda in the following order:

1 An unexpected opportunity with the side task as an ancestor;
2 A predicted opportunity with the side task as an ancestor;
3 A decision task with the side task as an ancestor;
4 An unexpected opportunity with the main task as an ancestor;
5 A predicted opportunity with the main task as an ancestor;
6 A decision task with the main task as an ancestor;
7 Choose a new side task;
8 A next step for the main task that interacts problematically with other tasks.

To summarize, PARETO continues to address its main task until the next step involves a problematic interaction; it then finds opportunities for other tasks, which it also pursues until the next step involves a problematic interaction; finally, it returns to the main task and continues to address it regardless of whether there are problematic interactions. The method of choosing the main task ensures that it is the most important task for PARETO to address, so if all available tasks are implicated in problematic interactions it is the main task that is the appropriate one to pursue. In this case, the next step for this task is indeed an opportunity, despite the problematic interaction.

This choice procedure ensures that PARETO fulfills the criteria described in section 8.2. The mechanisms for choosing the main and side tasks described in the previous two sections ensure that PARETO concentrates on important goals. The preference for opportunities for the side task, and the way that side tasks are chosen, help PARETO to seize opportunities. Continuity is provided by trying to continue with the parent of the previous main or side task, and by taking all non-interacting opportunities for the main task before seizing opportunities for other tasks. Finally, the mechanism of main and side tasks keeps PARETO focused and stops it getting side-tracked.

8.4 Summary

In this chapter I have described PARETO's task selector, which is based on the notion of *opportunities*. When choosing the next task to be processed, PARETO confines its attention to tasks for which there are opportunities, whether predicted or unexpected. By treating both types of opportunity in essentially the same way, PARETO can respond effectively to the unpredicted circumstances that it encounters while making full use of its plans.

PARETO's use of opportunities, together with its mechanism of main and side tasks, ensures that its behavior meets the following important criteria:

- It concentrates on important goals;
- It seizes opportunities;
- It maintains continuity;

- It maintains focus.

These criteria allow PARETO to approximate the optimal behavior of choosing the task with maximum expected utility. Accurate calculations of utility are complex and time-consuming; performing them would prevent PARETO from responding in a timely manner to its dynamic environment. The criteria used by PARETO, while a simplification, produce reasonable behavior under most circumstances, and allow a simple, effective task selection mechanism that involves little analysis. Indeed, the mechanism described here does not use PARETO's reasoning layer at all, and requires no explicit acquisition of information.

CHAPTER 9

CONCLUSIONS AND FURTHER WORK

9.1 Summary

The world is unpredictable—it is impossible to tell in advance exactly what its state will be at any future time. An agent operating in such a world must be able to cope with this unpredictability. In this dissertation I have considered two aspects of this problem: first, the agent must be able to acquire information about the actual state of the world; and second, it must be able to adapt its plans to the actual circumstances it encounters. I have explored these issues through the design and implementation of two systems: Cassandra, a system that constructs plans for use in an unpredictable world; and PARETO, a system that executes plans and adapts them to its current circumstances.

9.1.1 Information in plan execution

An agent operating in an unpredictable world must acquire information about its current circumstances if it is to be able to adapt its plans appropriately. The following questions must be answered in the design of the agent:

- When should the agent acquire information?
- What information should it acquire?
- How should it acquire the desired information?

In chapter 1, I argue that the agent requires information in order to make decisions, and that the goal to acquire information is thus a subgoal of the goal to make a decision. The need to make a decision indicates both when information is required and what information is required. This view of information acquisition as driven by the need to make decisions differs from the traditional view of information goals being subgoals of direct actions (McCarthy and Hayes 1969; Moore 1985; Morgenstern 1987; Etzioni et al. 1992); for example, my view is that the need to know the combination of a safe arises out of the need to decide what actions to perform in order to open it, while the traditional view holds that it is simply a precondition of the action of opening a safe. Seeing information goals as arising out of the need to make decisions provides much more flexibility and allows a unified view of the problems of plan construction in an unpredictable world. The decision of what actions to perform in order to open a safe is treated in the same way as the decision of what safe to open, or even whether to open a safe or go to the bank. In an unpredictable world it may well be impossible to tell in advance what the best method of obtaining cash will be; the decision of how to achieve that particular goal must be postponed until the circumstances under which the agent will be operating are known. In principle, this decision is exactly the same as the decision of whether to turn the dial on the safe to the left or the right; until the agent knows the safe combination, it cannot decide what actions it should perform.

The decisions that drive information acquisition may be either predicted or unpredicted. In an unpredictable world, the agent cannot fully specify its plans ahead of time, but can often predict what the possible options will be and what decisions it must make about how to achieve its subgoals and perform the necessary actions. However, totally unforeseen circumstances may arise that lead to the necessity of radically changing the current plans; by their very nature, these decisions cannot be foreseen by the agent but the occasion for them must be recognized on the fly.

The making of both these types of decision may require information. I argued that the goal to acquire information is a goal like any other goal, and that it can therefore be planned for like any other goal. There are three main methods of acquiring information—perception, memory retrieval, and inference—all of which can be planned activities, and each of which is useful under different circumstances.

The design of Cassandra shows how a planner can anticipate the need for decisions, plan to make them, and plan to acquire the necessary information. Cassandra, is, I believe, the first planner to treat decision-making explicitly. It is thus the first planner to allow for the possibility of using different decision-making methods, according to the decision and the circumstances under which it must be made.

In chapter 5, I demonstrate how the need for information arises out of PARETO's need to make decisions, both predicted and unpredicted. Previous work on information acquisition in plan execution has concentrated on perception (Kaelbling 1987; Olawsky and Gini 1990; Simmons 1990), but in the design of PARETO I have shown how all three methods of information acquisition can be treated equally. Perception, though vitally important, is not the only method of information acquisition and cannot be seen in isolation. It must be combined with memory retrieval and inference, and the implementation of PARETO demonstrates how this can be done. PARETO's plans include steps to acquire information using all three methods, which are considered as genuine alternatives to each other.

Cassandra and PARETO thus demonstrate the feasibility of the approach to information acquisition that I have described in both plan construction and plan execution.

9.1.2 Opportunities

The second major strand of work described in this thesis is concerned with how an agent operating in an unpredictable world can recognize when its existing plans need to be changed to cope with the unexpected. The foundation of the work that I describe is the realization that what is needed is a unifying framework within which an agent can both plan ahead and respond to the unexpected. In chapter 3 I argue that just such a framework is provided by opportunities. Planning ahead can be viewed as predicting when opportunities will arise and how they should be handled, while responding to the unexpected can be seen as recognizing and responding to unforeseen opportunities. If an agent has an efficient method of recognizing opportunities then it can effectively treat both predicted opportunities—those that have been planned for—and unpredicted opportunities in the same way.

The design of PARETO is based on this principle. Chapters 6 and 7 describe how PARETO recognizes opportunities, and chapter 8 describes how PARETO uses opportunities as the basis of its mechanism for deciding what to do next. This mechanism allows it to take advantage of unexpected opportunities while not losing sight of its primary task.

The main contribution of this thesis is the theory of opportunity recognition that it presents. Previous approaches have been inadequate. Hayes-Roth and Hayes-Roth (1979) confined their at-

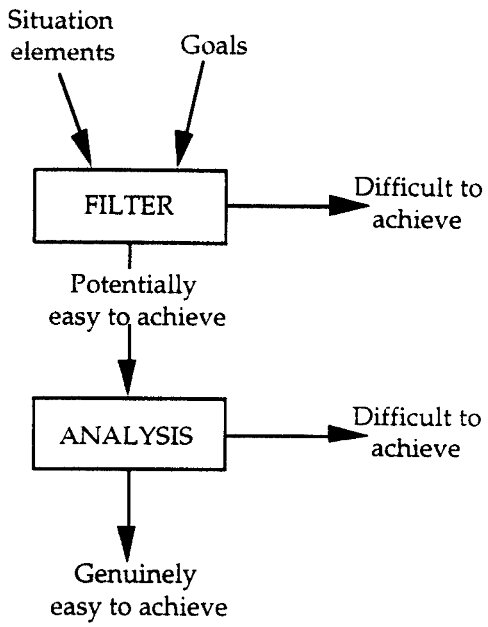


Figure 9.1 PARETO's filter for potential opportunities

tention to opportunism at the time of plan construction, and ignored opportunism during plan execution. Birnbaum and Collins (1984) fail to address the problem of how opportunities can be recognized in a dynamic world easily enough that they may be responded to rapidly. Hammond and his colleagues (1993) present a method of opportunity recognition that is effective only if the agent has decided exactly how to achieve its goals. Their method would not allow an agent to recognize an opportunity for a method of achieving a goal other than that currently being pursued. The method of opportunity recognition implemented in PARETO is much more powerful: it does not rely on any decisions having been made about how a goal should be achieved, but instead is based on a simple characterization of goals in terms of the functionality required to achieve them.

The chief problem facing an agent that wishes to respond appropriately in a dynamic and unpredictable world is that of choosing what to respond to. The number of possible factors in the world is enormous, and each one of them may in theory affect any of the agent's many goals. Reasoning about all possible implications would prevent any timely response. The solution that I propose and have implemented in PARETO is to use a filtering mechanism to indicate those of the agent's goals that are potentially affected by factors in the current situation: only the indicated goals need be analyzed in any detail (see figure 9.1). The filtering mechanism that PARETO uses has two foundations:

- The critical factor hypothesis;
- Reference features.

The critical factor hypothesis, which is introduced in chapter 3, states that the presence of a single factor is often crucial for the existence of an opportunity in a given situation. The process of recognizing opportunities thus comes down to a process of recognizing critical factors. PARETO recognizes critical factors through the use of reference features: features that are both cheap to compute and functionally significant.

The use of reference features allows an agent to recognize a large class of routine opportunities, and thus to operate flexibly and effectively in an unpredictable world. As well as indicating the potential presence of an opportunity for a goal, they also indicate the form that the opportunity is likely to take, and thus not only indicate when reasoning would be productive but also the direction the reasoning should take. They can also be used to indicate potential threats in the form of problematic interactions between goals, as described in chapter 7. Their use in PARETO supports the claim that, in order to respond in a timely fashion to unforeseen situations, an agent must have knowledge concerning the functional tendencies of the elements that it regularly encounters in its environment.

9.2 Further work

There are many directions for further work opened up by the research I have described. Some of them are enhancements to Cassandra and PARETO; others address more fundamental issues.

9.2.1 Cassandra and the classical planning paradigm

Cassandra is a contingency planner: it predicts situations in which several outcomes are possible and makes a plan for each possible outcome. The design of Cassandra ignores the question of which contingencies should be planned for. Some, for instance, might be so unlikely that the effort of planning for them is not worthwhile, while others might be such that there is no plan that will work.

As no planner can be expected to recognize the impossibility of achieving a goal in the general case (Chapman 1987), it is in general impossible to recognize the latter situation. However, there is a possible heuristic approach, as suggested by Peot and Smith (1992). We could introduce an alternative method of resolving open goal conditions: simply assume the goal in question fails. If a suitably high penalty were assigned to plans with failed goals, such a plan would be pursued only after Cassandra failed to find a plan in which all goals were achieved. Introducing this extension would, of course, result in loss of completeness. In setting the penalty to be applied to plans in which one branch is allowed to fail, an assumption must be made about the maximum possible cost of a viable plan. If there is in fact a viable plan that costs more than the penalty, Cassandra would no longer be guaranteed to find it.

Recognizing very improbable contingencies is in principle much easier. Draper, Hanks and Weld (1994b; 1994a) have recently built a probabilistic contingency planner, C-BURIDAN, which is based on Peot and Smith's CNLP, and Cassandra could be extended in the same way. A probabilistic version of Cassandra would have significant advantages over C-BURIDAN, because of its use of secondary preconditions to handle context-dependent effects and its explicit representation of decisions.

Another of Cassandra's limitations is that it does not perform branch merging: it cannot identify an action in one contingency branch with an action in another. Work is in progress to introduce this functionality; a limited version is described in (Collins and Pryor 1994).

The representation of different specific methods for making decisions is an important direction for future research. An extension of Cassandra (Collins and Pryor 1994) introduces a type of decision that directs the executing agent to perform all branches resulting from a given source of uncertainty, which allows the construction of plans that can succeed in situations in which there is no way of telling what the actual outcome is (*e.g.*, the bomb in the toilet problem described in chapter 2). It would also be relatively simple to add to Cassandra's decision representation processes that depend on information about probabilities, *e.g.*, to follow a particular course of action if the probability of a given outcome exceeds a certain value.

Cassandra, like all other planners in the classical paradigm, is designed to find a single plan that will achieve the goal in the given circumstances. There is no notion of the robustness of the resulting plan—whether it will work in a variety of circumstances. Probabilistic planners such as C-BURIDAN introduce the notion of robustness by constructing plans that will succeed with a probability greater than some lower limit. The problem with the probabilistic approach is that it requires information about the probabilities of various outcomes: C-BURIDAN could not, for instance, construct a plan for the bomb-in-a-toilet problem if it had no information about the probability of the bomb's being in each package. A extension of Cassandra (Collins and Pryor 1994) can handle this problem, but cannot handle situations in which there is no plan that is absolutely guaranteed to work. An area of future research is to investigate the notion of robustness, develop characterizations of robust plans, and develop methods of constructing them.

In general, Cassandra suffers from the same problems as all other planners in the classical paradigm: the search space blows up rapidly with the complexity of the problem (made worse in Cassandra by contingency branching);. the treatment of time is limited and actions are represented as instantaneous state changes; the planners have little notion of the efficiency of the plans they construct; and there is no effective trade-off between the efficiency of the plan produced and the effort expended in producing it. Various researchers have investigated these problems, but no adequate solutions have yet been found.

9.2.2 PARETO's reasoning layer

The chief limitation of the current implementation of PARETO is the lack of a reasoning layer (see figure 9.2). As described in chapter 4, the reasoning layer would communicate with the execution layer through the RAP library, task agenda and memory. It would be responsible for the detailed analysis of potential interactions, and would also influence the task selection process.

The existence of such a reasoning layer would provide significant advantages over the current implementation. First, the current ad hoc methods of reasoning about interactions could be replaced. This would require the development of principled methods that would remain simple enough to enable timely responses, but that would be able to reason more effectively about future possibilities. Firby originally designed the RAP representation to facilitate both plan execution and reasoning about plans, and developing a full reasoning layer would involve a full investigation of the latter aspect of the representation.

Second, the development of a full reasoning layer would enhance PARETO's capabilities of recognizing opportunities. McDermott (1992) describes how replanning in parallel can enable a system to apply transformations that allow it to combine its plans efficiently. If PARETO had such a capability it could, for example, choose its routes based on the opportunities that would be opened up; it could also use more sophisticated methods of task selection than that described in chapter 8. Even a reasoning layer that could operate in parallel with the execution layer would not be able to reason about all possibilities: an important area of research would therefore be to investigate how reasoning could be

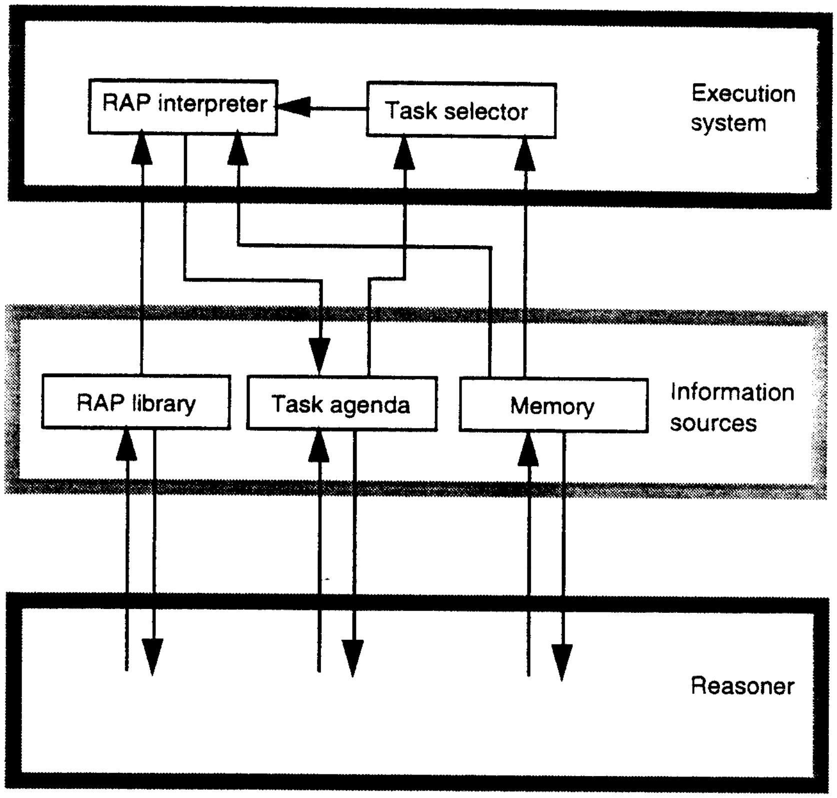


Figure 9.2 PARETO's reasoning and execution layers

focused productively. An obvious, and I believe productive, approach would be to extend the use of reference features.

Third, the mechanism for choosing between different methods of achieving a goal could be improved. Currently, the choice is based on static preferences between methods and the past success rates of methods, and, beyond feasibility issues, is fairly insensitive to the context in which the choice must be made. A full reasoning layer would allow more detailed consideration of the alternatives.

Fourth, more sophisticated methods of avoiding problematic interactions could be used. As described in chapter 7, PARETO currently has only a few simple methods of avoidance; these take little account of the details of the potential interactions. A full reasoning layer would allow the use of more elaborate plan transformations (Collins 1987; McDermott 1992) and more precise avoidance strategies.

Finally, a full reasoning layer would extend PARETO's information acquisition capabilities. Currently, it is limited to a few isolated types of inference (see chapters 4 and 7 and appendix A) that are used in restricted circumstances.

I have discussed the functionality of a full reasoning layer: it remains to consider the methods of reasoning that it would use. A range of reasoning methods would allow it to choose the most appropriate for the combination of reasoning task and circumstances. An important area of further work is to investigate and characterize possible methods. A starting point would be to adapt Cassandra for use with PARETO, and characterize those reasoning tasks for which a contingency planner in the classical paradigm would be appropriate.

9.2.3 Information acquisition in PARETO

In the previous section I considered how PARETO's inference capabilities could be extended. There are, however, other aspects of its information acquisition that would also repay further work.

Currently, the memory system severely limits PARETO's performance. There are few significant changes from the system as implemented by Firby, which has some severe drawbacks. First, as the number of propositions stored grows over time, retrieval and modification become significantly slower. Second, there is no method of discarding assertions that are out of date except through explicit retraction. It would be useful to investigate the possibility of having assertions gradually expire, and to consider what characteristics of the information represented would influence the rate of expiry. Third, the method of identifying objects currently observed with those observed previously is complex and relies on substantial oversimplifications: for example, that objects do not change location.[1] This area of research is vitally important in the design of autonomous agents, and would repay investigation.

The current design of PARETO assumes that all perception (with the exception of proprioceptive feedback) occurs deliberately, and thus ignores the problems of coping with large amounts of perceptive data. I believe that the use of reference features will prove productive in the design of filtering mechanisms to handle the data-overload problem that is common to many robots and other autonomous systems (Hayes-Roth 1992). The notion of directed perception is, however, an important one and would benefit from some refinement. For example, the frequency with which PARETO scans its surroundings in order to keep up to date is currently constant: an obvious extension would be to vary this according to the perceived dynamism of the environment. A potentially productive approach to this problem is to adapt the credibility theory results used in the world of insurance (Hossack et al. 1983; Beard et al. 1984).

9.2.4 Decisions

A common theme emerging from previous sections is the detailed consideration of decisions. Cassandra, by allowing the explicit representation of decisions, provides scope for the use of a variety of decision-making methods; the enhancement of PARETO's reasoning layer would allow it to make more use of analysis and projection in the making of decisions; and the way in which information is acquired may have significant effects on the viability of decision-making methods. The methods currently used in PARETO are *ad hoc* and limited. An important area of further work is to investigate possible methods of decision making and to determine the circumstances under which they are practicable. In particular, a characterization of the types of information required and the sensitivity of the decision making process to the quantity and quality of the available information are important.

Much of the existing work on decision making takes a theoretical, utility-based view (Watson and Buede 1987); the use of utility theory requires a large amount of information about probabilities and utilities that may not be available, and the analysis that is required may be prohibitive for an agent

[1] Because of its complexity and the fact that it is irrelevant to most of the issues I have considered, I have not described it in this dissertation.

operating in a dynamic world. There is evidence that people do not exclusively use these methods (Tversky and Kahneman 1988); and there are convincing arguments that optimality should not necessarily be the goal (Simon 1955). I believe that a productive area of research is in the field of qualitative representation of the factors that should influence decisions (Cohen 1985; Wellman 1988; Fox 1991). It is important that an agent operating in a dynamic and unpredictable world should be able to make adequate decisions on the basis of very little information, as there will be occasions on which information is not readily available. It would be useful to investigate whether qualitative approaches are more likely to be successful in such circumstances, or whether qualitative techniques are adequately robust.

9.2.5 Reference features

Perhaps the most important area of research suggested by the work described in this dissertation is the further investigation of reference features. In the implementation of PARETO I have demonstrated that a filtering mechanism for potential opportunities based on reference features is feasible, but I have not yet demonstrated that it is effective. In order to do this, a system that operates in a real world domain must be built. This will involve the construction of a set of reference features for the chosen domain. I believe that a set of reference features can be found for any domain, but this requires empirical confirmation.

Connected with this endeavor is the investigation of whether reference features are a genuine psychological phenomenon or whether they are simply the basis of an effective mechanism that is totally unlike anything actually used by people. There are two principal methods I would like to use to investigate this hypothesis. First, the use of reference features in opportunity recognition can be investigated. Recently some interesting experiments on the use of objects in opportunity recognition have been performed (Patalano et al. 1993); these experiments were designed to test a theory which relies on the agent having already committed to a specific method of goal achievement (Hammond et al. 1993). I believe that similar experiments would help to elucidate the psychological foundations of reference features.

Second, an important corollary of the use of reference features that I have described is their role in the development of *expertise*. An expert in a domain is one who can solve problems in that domain effectively; a domain expert is certainly expected to be able to recognize routine opportunities in that domain. This compares with a domain novice, who can solve problems only with difficulty, and may not recognize even routine opportunities. The difference between the two is due to their relative familiarities with the domain; in particular, a novice frequently has to work from first principles. I believe that a useful characterization of experts and novices is in terms of the reference features they have for the domain. An expert is one who has a comprehensive set of reference features; a novice is one who has a limited or nonexistent set. I would like to investigate this hypothesis during the construction of the set of reference features for a real world domain described above. Through interviewing experts and investigating their methods of problem solving (through techniques such as protocol analysis) I hope to discover whether they do in fact use reference features and whether they are aware of doing so and can generate sets of them.

This leads on to the third major area of investigation connected with reference features, and that is how an agent acquires them. PARETO is simply provided with a set attached to the objects in its world and to its tasks and goals; however, an important aspect of intelligent behavior is the ability to learn from experience. This learning should certainly include learning about the functional aspects of the agent's environment, and learning to associate perceptual cues with them. If reference features are to form the basis of a satisfactory theory of behavior for a goal-based agent, then a theory of their acquisition must be developed.

9.3 Conclusion

One of the most fundamental problems facing an agent operating in the real world is how to respond appropriately to its dynamic environment by balancing action and deliberation (Hanks and Firby 1990). Too much action and not enough deliberation and its responses are unlikely to be appropriate; too much deliberation and not enough action and they are unlikely to be timely. To operate effectively, the agent must be able to tell when to deliberate and what to deliberate about.

This problem is a special case of the more general problem of when to gather information about the world and what information to gather. In a complex world it is impossible to know everything there is to know: the agent must be selective, and must focus its information-gathering activities on the information that most affects the carrying out of its tasks. Deliberation is a form of information gathering (together with perception and memory retrieval), and an agent's deliberation must likewise be focused.

Luckily, the world is not entirely unpredictable. Sometimes an agent can tell in advance that it is going to require further information and the form that information is likely to take. Cassandra is a planning system for use in such circumstances, and the plans that PARETO uses also represent information gathering activities, whether deliberation, perception, or memory retrieval, explicitly. However, there are occasions when the need for information cannot be predicted, and the chief contribution of this thesis addresses the problem of recognizing such situations.

In this dissertation I have argued that, in order for an agent to operate effectively in an unpredictable world, it must have a comprehensive set of *reference features*: features that are both cheap to compute and functionally relevant. The existence of such features is possible because of the basic regularity and functional consistency of the world, and the use of reference features allows the agent to make full use of these world properties. Reference features provide the necessary focusing by indicating both when information-gathering should take place and the form that it should take. I have shown that a mechanism based on reference features is feasible through the implementation of PARETO; the next stage is to show that it works in the real world.

The approach of treating deliberation as a special case of the more general problem of information has thus proved effective. The problem facing an agent in the real world is not just when to think and when to do: it is better expressed as when to find out and when to do. One of the chief characteristics of humans is their curiosity—we are now working towards building inquisitive computer systems.

REFERENCES

Agre, P. E. 1988. "The Dynamic Structure of Everyday Life." PhD, Massachusetts Institute of Technology.

Agre, P. E. and D. Chapman. 1987. "Pengi: An implementation of a theory of activity." In *Proceedings of the Sixth National Conference on Artificial Intelligence,* AAAI.

Allen, J., J. Hendler, and A. Tate, ed. 1990. *Readings in Planning*. San Mateo, CA: Morgan Kaufmann.

Alterman, R. 1986. "An adaptive planner." In *Proceedings of the Fifth National Conference on Artificial Intelligence,* AAAI, 65-69. Also in (Allen, et al., 1990).

Barrett, A., S. Soderland, and D. S. Weld. 1991. *Effect of Step-Order Representations on Planning*. Department of Computer Science and Engineering, University of Washington, Seattle. Technical Report 91-05-06.

Barrett, A. and D. S. Weld. 1993. Partial-Order Planning: Evaluating Possible Efficiency Gains. *Artificial Intelligence* (Forthcoming):

Beard, R. E., T. Pentikäinen, and E. Pesonen. 1984. *Risk theory: The stochastic basis of insurance*. 3rd ed., London: Chapman and Hall.

Birnbaum, L. 1986. *Integrated processing in planning and understanding*. Department of Computer Science, Yale University. Technical Report YALEU/CSD/RR #489.

Birnbaum, L. and G. Collins. 1984. "Opportunistic planning and Freudian slips." In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society, Boulder, CO,* Lawrence Erlbaum Associates.

Boddy, M. and T. Dean. 1989. "Solving time-dependent planning problems." In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, MI,* AAAI.

Boddy, M. and K. Kanazawa. 1990. "Controlling decision-theoretic inference." In *Working notes of the AAAI Spring Symposium: Planning in Uncertain, Unpredictable, or Changing Environments, Stanford, CA,* AAAI.

Brand, M. and L. Birnbaum. 1990. "Noticing Opportunities in a Rich Environment." In *Twelfth Annual Conference of the Cognitive Science Society, Cambridge, MA,* Lawrence Erlbaum Associates.

Bresina, J. and M. Drummond. 1990. "Integrating Planning and Reaction: A preliminary report." In *Working notes of the AAAI Spring Symposium: Planning in Uncertain, Unpredictable, or Changing Environments, Stanford, CA,* AAAI.

Brooks, R. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* 2 (1): 14-23.

Brooks, R. A. 1990. Elephants don't play chess. *Robotics and Autonomous Systems* 6 : 3-15.

Chapman, D. 1987. Planning for Conjunctive Goals. *Artificial Intelligence* 32 : 333-337. Also in (Allen, et al., 1990).

Chapman, D. 1990. *Vision, instruction and action*. MIT Artificial Intelligence Laboratory. Technical Report AI-TR 1204.

Chrisman, L. and R. Simmons. 1991. "Sensible Planning: Focusing Perceptual Attention." In *Proceedings of the Ninth National Conference on Artificial Intelligence, Anaheim, CA,* AAAI.

Cohen, P. R. 1985. *Heuristic reasoning about uncertainty: an artificial intelligence approach*. London: Pitman.

Collins, G. and L. Pryor. 1992a. "Achieving the functionality of filter conditions in a partial order planner." In *Proceedings of the Tenth National Conference on Artificial Intelligence, San Jose, CA,* AAAI.

Collins, G. and L. Pryor. 1992b. *Representation and performance in a partial order planner*. The Institute for the Learning Sciences. Technical Report #35.

Collins, G. and L. Pryor. 1993. "What are filter conditions for?" In *Working notes of the Spring Symposium on Foundations of Automatic Planning: The Classical Approach and Beyond., Stanford, CA,* AAAI.

Collins, G. and L. Pryor. 1994. "Keys and boxes: Planning in an uncertain world." In *Twelfth National Conference on Artificial Intelligence, Seattle, WA,* , (submitted).

Collins, G. C. 1987. *Plan creation: Using strategies as blueprints.* Department of Computer Science, Yale University. Technical Report YALEU/CSD/RR #599.

Davis, E. 1990. *Representations of Commonsense Knowledge.* San Mateo, CA: Morgan Kaufmann.

Dean, T. and M. Boddy. 1988. "An analysis of time-dependent planning." In *Proceedings of the Seventh National Conference on Artificial Intelligence, St Paul, MN,* AAAI.

Dehn, N. J. 1989. *Computer story-writing: The role of reconstructive and dynamic memory.* Yale University Department of Computer Science. YALEU/DCS/TR792.

Draper, D., S. Hanks, and D. Weld. 1994a. "A probabilistic model of action for least-commitment planning with information gathering." In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence, Seattle, WA,* Morgan Kaufmann.

Draper, D., S. Hanks, and D. Weld. 1994b. "Probabilistic Planning with Information Gathering and Contingent Execution." In *Second International Conference on Artificial Intelligence Planning Systems, Chicago, IL,* Morgan Kaufmann.

Etzioni, O., S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. 1992. "An approach to planning with incomplete information." In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning, Boston, MA,* Morgan Kaufmann.

Feldman, J. A. and R. F. Sproull. 1977. Decision Theory and Artificial Intelligence II: The Hungry Monkey. *Cognitive Science* 1 : 158-192. Also in (Allen, et al., 1990).

Ferguson, I. A. 1992. *TouringMachines: An architecture for dynamic, rational, mobile agents.* Computer Laboratory, University of Cambridge. Technical Report No. 273.

Fikes, R. E. and N. J. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2 : 189-208. Also in (Allen, et al., 1990).

Firby, R. J. 1987. "An investigation into reactive planning in complex domains." In *Proceedings of the Sixth National Conference on Artificial Intelligence, Seattle, WA,* AAAI, 202 - 206.

Firby, R. J. 1989. *Adaptive execution in complex dynamic worlds.* Department of Computer Science, Yale University. Technical Report YALEU/CSD/RR #672.

Firby, R. J. 1992. "Building symbolic primitives with continuous control routines." In *First International Conference on Artificial Intelligence Planning Systems, College Park, Maryland,* Morgan Kaufmann, 62 - 69.

Firby, R. J. and S. Hanks. 1987. *The simulator manual.* Department of Computer Science, Yale University. Technical Report YALEU/CSD/RR #563.

Fox, J. 1991. "Decision theory and autonomous systems." In *IMACS workshop on qualitative reasoning and decision aids,* Elsevier.

Georgeff, M. P. and F. F. Ingrand. 1989. "Decision-making in an embedded reasoning system." In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, MI,* Morgan Kaufmann, 972-978.

Gibson, J. J. 1979. *The ecological approach to visual perception.* Boston, MA: Houghton Mifflin.

Goodwin, R. and R. Simmons. 1992. "Rational handling of multiple goals for mobile robots." In *First International Conference on Artificial Intelligence Planning Systems, College Park, Maryland,* Morgan Kaufmann, 70 - 77.

Haas, A. R. 1986. A Syntactic theory of Belief and Action. *Artificial Intelligence* 28 : 245-292.

Haddawy, P. and S. Hanks. 1990. "Issues in Decision-Theoretic Planning: Symbolic Goals and Numeric Utilities." In *Proceedings of a Workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, CA,* DARPA.

Haddawy, P. and S. Hanks. 1992. "Representations for decision-theoretic planning: Utility functions for deadline goals." In *Third International Conference of Principles of Knowledge Representation and Reasoning, Boston, MA,* Morgan Kaufmann.

Hammond, K. 1984. *Indexing and Causality: The organization of plans and strategies in memory.* Department of Computer Science, Yale University. Technical Report YALEU/CSD/RR #351.

Hammond, K. 1989. "Opportunistic Memory." In *Eleventh International Joint Conference on Artificial Intelligence, Detroit, MI,* , 504-510.

Hammond, K., T. Converse, M. Marks, and C. Seifert. 1993. Opportunism and learning. *Machine Learning* 10 : 279-309.

Hanks, S. and R. J. Firby. 1990. "Issues and architectures for planning and execution." In *Proceedings of a Workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, CA,* DARPA.

Hanks, S. and D. S. Weld. 1992. "Systematic adaptation for case-based planning." In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems, College Park, Maryland,* Morgan Kaufmann, 96-105.

Hanks, S. J. 1990. *Projecting plans for uncertain worlds.* Department of Computer Science, Yale University. Technical Report YALEU/CSD/RR #756.

Hansson, O., A. Mayer, and S. Russell. 1990. "Decision-Theoretic Planning in BPS." In *Working notes of the AAAI Spring Symposium: Planning in Uncertain, Unpredictable, or Changing Environments, Stanford, CA,* AAAI.

Hartman, L. 1990. *Decision Theory and the Cost of Planning.* Computer Science Department, University of Rochester. Technical Report 355.

Hayes-Roth, B. 1990a. Architectural foundations for real-time performance in intelligent agents. *The Journal of Real-Time Systems* 2 : 99 - 125.

Hayes-Roth, B. 1990b. "Dynamic control planning in intelligent agents." In *Working notes of the AAAI Spring Symposium: Planning in Uncertain, Unpredictable or Changing Environments, Stanford University,* AAAI.

Hayes-Roth, B. 1992. "Selective perception in an intelligent monitoring agent." In *Working notes of the AAAI Spring Symposium: Control of Selective Perception, Stanford, CA,* AAAI.

Hayes-Roth, B. and F. Hayes-Roth. 1979. A cognitive model of planning. *Cognitive Science* 3 (4): 275-310. Also in (Allen, et al., 1990).

Hexmoor, H. and D. Nute. 1992. *Methods for deciding what to next and learning.* University of Georgia. TR AI-1992-01.

Hintikka, J. 1962. *Knowledge and Belief.* Ithaca, NY: Cornell University Press.

Horvitz, E. J., J. S. Breese, and M. Henrion. 1988. Decision Theory in Expert Systems and Artificial Intelligence. *International Journal of Approximate Reasoning* 2 : 247-302.

Horvitz, E. J., G. F. Cooper, and D. E. Heckerman. 1989. "Reflection and Action Under Scarce Resources: Theoretical Principles and Empirical Study." In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, MI,* IJCAI.

Hossack, I. B., J. H. Pollard, and B. Zehnwirth. 1983. *Introductory statistics with applications in general insurance.* Cambridge: Cambridge University Press.

Kaelbling, L. P. 1987. "An Architecture for Intelligent Reactive Systems." In *Reasoning about Actions and Plans,* ed. Georgeff and Lansky. Also in (Allen, et al., 1990).

Langley, P. 1992. "Systematic and Nonsystematic Search Strategies." In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems, College Park, Maryland,* Morgan Kaufmann.

Lyons, D. M. and A. J. Hendriks. 1992. "A practical approach to integrating reaction and deliberation." In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems, College Park, Maryland,* Morgan Kaufmann, 153-162.

Maes, P. 1991. "Adaptive action selection." In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society, Chicago, IL,* Lawrence Erlbaum Associates.

McAllester, D. and D. Rosenblitt. 1991. "Systematic nonlinear planning." In *Proceedings of the Ninth National Conference on Artificial Intelligence, Anaheim, CA,* AAAI, 634-639.

McCarthy, J. and P. J. Hayes. 1969. "Some philosophical problems from the standpoint of artificial intelligence." In *Machine Intelligence 4,* ed. B Meltzer and D. Michie. 463-502. Edinburgh University Press. Also in (Allen, et al., 1990).

McDermott, D. 1992. *Transformational planning of reactive behavior*. Yale University, Department of Computer Science. YALEU/CSD/RR #941.

McDermott, D. V. 1987. A Critique of Pure Reason. *Computational Intelligence* 3 : 151-160.

Moore, R. C. 1985. " A Formal Theory of Knowledge and Action." In *Formal Theories of the Commonsense World*, ed. J. R. Hobbs and R. C. Moore. Norwood, NJ: Ablex. Also in (Allen, et al., 1990).

Morgenstern, L. 1987. "Knowledge Preconditions for Actions and Plans." In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Milan*, IJCAI.

Olawsky, D. and M. Gini. 1990. "Deferred planning and sensor use." In *Proceedings of a Workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, CA*, DARPA, 166 - 174.

Owens, C. 1990. "Indexing and Retrieving Abstract Planning Knowledge." Ph.D. Thesis, Yale University.

Patalano, A. L., C. M. Seifert, and K. J. Hammond. 1993. "Predictive encoding: Planning for opportunities." In *Fifteenth Annual Conference of the Cognitive Science Society, Boulder, CO*, Lawrence Erlbaum Associates, 800-805.

Pednault, E. P. D. 1988. Synthesizing plans that contain actions with context-dependent effects. *Computational Intelligence* 4 : 356-372.

Pednault, E. P. D. 1989. "ADL: Exploring the middle ground between STRIPS and the situation calculus." In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann.

Pednault, E. P. D. 1991. "Generalizing nonlinear planning to handle complex goals and actions with context-dependent effects." In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, Sydney, Australia*, IJCAI.

Penberthy, J. S. and D. S. Weld. 1992. "UCPOP: A sound, complete, partial order planner for ADL." In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning, Boston, MA*, Morgan Kaufmann.

Peot, M. A. and D. E. Smith. 1992. "Conditional Nonlinear Planning." In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems, College Park, Maryland*, Morgan Kaufmann.

Pryor, L. and G. Collins. 1991. "Information gathering as a planning task." In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society, Chicago, IL*, Lawrence Erlbaum Associates.

Pryor, L. and G. Collins. 1992. "Planning to perceive: A utilitarian approach." In *Workshop notes of the AAAI Spring Symposium: Control of Selective Perception, Stanford, CA*, AAAI.

Pryor, L. and G. Collins. 1993. *Cassandra: Planning with contingencies*. The Institute for the Learning Sciences, Northwestern University. Technical Report #41.

Russell, S. and E. Wefald. 1991. *Do the Right Thing: Studies in Limited Rationality*. Cambridge, MA: MIT Press.

Sacerdoti, E. 1977. *A structure for plans and behavior*. New York: American Elsevier.

Schank, R. C. 1982. *Dynamic memory*. Cambridge: Cambridge University Press.

Schank, R. C. 1986. *Explanation Patterns*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Schank, R. C. and R. P. Abelson. 1977. *Scripts, plans, goals and understanding*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Schoppers, M. J. 1987. "Universal Plans for Reactive Robots in Unpredictable Environments." In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Milan*, IJCAI.

Shafer, G. and J. Pearl, ed. 1990. *Readings in Uncertain Reasoning*. San Mateo, CA: Morgan Kaufmann.

Simmons, R. 1990. "An architecture for coordinating planning, sensing, and action." In *Proceedings of a Workshop on Innovative Approaches to Planning, Scheduling and Control, San Diego, CA*, DARPA.

Simon, H. A. 1955. A behavioral model of rational choice. *Quarterly Journal of Economics* 69 : 99 - 118.

Sloman, A. 1989. On designing a visual system: Towards a Gibsonian computational theory of vision. *Journal of Experimental and Theoretical Artificial Intelligence* 1 (4): 189-337.

Steel, S. 1993. "A connection between decision theory and program logic." In *Prospects for Artificial Intelligence: Proceedings of the Ninth biennial conference of the Society for the Study of Artificial*

Intelligence and the Simulation of Behaviour, ed. A. Sloman, D. Hogg, G. Humphreys, A. Ramsay, and D. Partridge. 281-290. Amsterdam: IOS Press.

Stefik, M. 1981. Planning with constraints (MOLGEN: Part 2). *Artificial Intelligence* 16 : 141-170.

Sussman, G. J. 1975. *A computer model of skill acquisition.* New York: American Elsevier.

Tate, A. 1977. "Generating project networks." In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge, MA,* IJCAI. Also in (Allen, et al., 1990).

Tversky, A. and D. Kahneman. 1988. "Rational choice and the framing of decisions." In *Decision Making: Descriptive, Normative and Prescriptive Interactions,* ed. D. E. Bell, H. Raiffa, and A. Tversky. Cambridge: Cambridge University Press.

Warren, D. 1976. "Generating conditional plans and programs." In *Proceedings of the AISB Summer Conference, University of Edinburgh,* .

Watson, S. R. and D. M. Buede. 1987. *Decision synthesis.* Cambridge: Cambridge University Press.

Wellman, M. P. 1988. "Qualitative Probabilistic Networks for Planning under Uncertainty." In *Uncertainty in Artificial Intelligence,* ed. J. F. Lemmer and L. N. Kanal. 2. Amsterdam: Elsevier. Also in (Schafer & Pearl 1990).

Wellman, M. P. 1990. *Formulation of Tradeoffs in Planning under Uncertainty.* London: Pitman.

Wilkins, D. E. 1988. *Practical Planning: Extending the Classical AI Planning Paradigm.* San Mateo, CA: Morgan Kaufmann.

Zito-Wolf, R. and R. Alterman. 1993. "A framework and analysis of current proposals for the case-based organization and representation of procedural knowledge." In *Proceedings of the Eleventh National Conference on Artificial Intelligence, Washington, DC,* AAAI, 73-78.

APPENDIX A

THE RAP LANGUAGE

This appendix gives the full syntax and semantics of the RAP language as used by PARETO. Much of it is adapted from (Firby 1989); differences between the language used by PARETO and that used in the original RAPs system are noted.

An extended BNF is used for the definitions. Non-terminals are *italicized*. Terminals include quoted strings and base programming language types (e.g., `symbol`). The order in which terms enclosed in double square brackets ⟦ ⟧ appear is not significant. The Kleene star * has its usual significance, meaning zero or more occurrences. The suffix $^+$ means one or more occurrences, and the suffix ? means zero or one occurrence.

A.1 RAP clauses

rap-definition ::= '(define-rap' *index-clause* ⟦ *success-clause*? *monitor-clause*? *reference-feature-clause*? *preconditions-clause*? *constraints-clause*? *repeat-clause*? *resources-clause*? ⟧ '(methods' *method-description*$^+$ '))'

A.1.1 Index clause

index-clause ::= '(index (' *rap-name in-vars*? *out-vars*? '))'
rap-name ::= `symbol`
in-vars ::= *variable**
out-vars ::= '=>' *variable**
variable ::= `variable`

The *in-vars* must have current bindings when the task described by the RAP is processed. The *out-vars* become bound during task execution and the bindings are exported to the parent task. The index clause is described in more detail in section 4.3.1.

A.1.2 Success clause

success-clause ::= '(succeed' *memory-formula* ')'

The *memory-formula* (see section A.4) in the success clause describes the criteria applied to see whether the task represented by the RAP has succeeded. It may contain free variables (variables other than those in the input variables of the index). These may be used to bind output variables in the index, but they will not bind variables in the remainder of the RAP definition. If the success clause is not specified in the RAP definition, the task is considered to have succeeded when it has successfully executed any of its methods. The success clause is described in more detail in section 4.3.2.

A.1.3 Monitor clause

monitor-clause ::= '(monitor' *memory-formula* ')'

The *memory-formula* (see section A.4) in the monitor clause describes the criteria applied to see whether the task represented by the RAP is eligible for processing. If the monitor clause is not specified in the RAP definition, the task is always eligible for processing. Free variables in the monitor clause cannot be used to bind variables in the remainder of the RAP definition. The monitor clause is described in more detail in section 8.3.1.

A.1.4 Reference feature clause

reference-feature-clause ::= '(reference-features' *reference-feature*+ ')'
reference-feature ::= `symbol` | '(criterion' *criterion* ')' | '(loc' *location* ')'
criterion ::= *variable* | `symbol`

The reference features used to determine task priority are described in more detail in section 8.3.3. The *criterion* and *location* cannot be free variables. The values to which they are bound are used to find the appropriate reference features, as listed in figures B.8 and B.9.

A.1.5 Preconditions clause

preconditions-clause ::= '(preconditions' *memory-formula* ')'

The *memory-formula* (see section A.4) in the preconditions clause describes conditions that must be true in order for the task to be processed. If they are not true, the task fails when it is processed. Free variables in the preconditions clause cannot be used to bind variables in the remainder of the RAP definition.

A.1.6 Constraints clause

constraints-clause ::= '(constraints' *memory-formula* ')'

The *memory-formula* (see section A.4) in the constraints clause describes conditions that must be true throughout the processing of the task. They are inherited by subtasks. If they are not true, the task fails when it is processed. Free variables in the preconditions clause cannot be used to bind variables in the remainder of the RAP definition.

A.1.7 Repeat clause

repeat-clause ::= '(repeat' *memory-formula* ')'

The *memory-formula* (see section A.4) in the repeat clause describes conditions that indicate that the task should be repeated. The semantics of this clause are "repeat-while," not "repeat-until." If both the success clause and the repeat clause are true, the success clause takes precedence. If no repeat clause is specified, the task is not allowed to repeat indefinitely, but fails after a small fixed number (currently two) attempts. Free variables in the repeat clause can be used to bind variables in the remainder of the RAP definition.

A.1.8 Resources clause

resources-clause ::= '(resources' *resource-definition*$^+$ ')'
resource-definition ::= *set-resource-definition* | *counter-resource-definition*
set-resource-definition ::= '(set' *variable set-value*? ')'
counter-resource-definition ::= '(counter' *variable counter-value* ')'
set-value ::= `set-resource`
counter-value ::= `counter-resource` | `integer`

Resources are used for short term memory. A resource is created and bound to a variable in the resources clause. The values supplied are used to initialize them; if no initial value is supplied for a set resource, it is initially empty. Resources are updated at execution time as specified by annotations in the task-net specification (see section A.3.2). Set resources are updated in two ways: things may be added or removed. Similarly, counter resources may be incremented or decremented.

A.2 Method descriptions

method-description ::= '(' *method-name*? *preference-ranking*? *clone-method-specification*? *method-context*
(*task-net-description* | *primitive-action-specification* | *reasoning-request-specification*) ')'
method-name ::= `symbol`
preference-ranking ::= `integer`

If no name is specified for a method, then a unique one is generated. If no preference ranking is given, then zero is assumed. The preference rankings are used to choose between methods when more than one is applicable.

A.2.1 Clone specification

clone-method-specification ::= '(clone' *clone-value* ')'
clone-value ::= `integer` | `variable`
If the *clone-value* is a variable, that variable must be bound to a counter resource or an integer at the time the method is selected. If the effective value is greater than one, then the method is performed that number of times. For example, if there is a task net the requisite number of families of subtasks are placed on the task agenda. This way of duplicating methods is not in Firby's original language specification.

A.2.2 Method context

method-context ::= '(context' *memory-formula* ')'

The *memory-formula* (see section A.4) in the method context describes conditions that indicate that the method is appropriate. If no context is specified, the method is always appropriate. Free variables in the method can be used to bind variables in the remainder of the method description. Method contexts are discussed in more detail in section 4.3.3.

A.2.3 Primitive actions

primitive-action-specification ::= '(primitive' *action-specification* ')'
action-specification ::= `hardware-action` | *memory-action*
memory-action ::= '(int$assert' *variable* ')' | '(int$erase' *variable* ')'

Methods containing primitive actions generate primitive action requests. These requests are passed either to the hardware interface or to the memory system. Hardware actions are passed to the TRUCKWORLD simulator, and are described further in appendix B. Memory actions either add an assertion to memory or erase an assertion from memory. The variable in a memory action must be bound to an appropriate clause when the method is executed.

A.2.4 Reasoning requests

reasoning-request-specification ::= '(reason' *reasoning-request* ')'
reasoning-request ::= *problem-request* | *interaction-request* | *default-request*
problem-request ::= '(solve-problem' *goal*')'
interaction-request ::= '(analyze-potential-interaction' *interaction* ')'
default-request ::= '(infer-default' *thing property-name property-specifiers answer* ')'
answer ::= *variable*
goal ::= *variable*
interaction ::= *variable*
property-name: := *variable*
property-specifiers ::= *variable*

Reasoning requests were not in the original RAPs language described by Firby. They are described in section 4.3.3.

Problem solvers

The *goal* variable in a solve-problem reasoning request must be bound to an *index-clause*. There is currently just one problem-solver in PARETO, for the (truck-travel-known-route ?start ?finish) goal. It finds a route from the location specified by ?start to that specified by ?finish, using simple breadth-first search. Problem solvers can be called explicitly via reasoning requests; they are also called automatically whenever

Interaction type	Subtasks to gather information	Avoidance method
(cut *instrument object*)	If the *instrument* is a knife, is its blade extended?	If the *instrument* is a knife, retract the blade.
(move *instrument*)		Cannot be avoided.
(puncture *instrument object*)	If the *object* is a bag, what is it made of?	Postpone until the interacting task has been completed.
(rip *instrument object*)	If the *object* is a bag, what is it made of?	Postpone until the interacting task has been completed.
(smear *instrument object*)	If the *object* is painted, is the paint wet?	Delay for 50 time units to allow the paint to dry.
(stain*instrument object*)		Postpone until the interacting task has been completed.

Figure A.1 Analyzing interactions

PARETO fails to find a RAP in its library. The latter functionality was provided in a slightly different form in Firby's RAPs system, but was undocumented.

Analyzing interactions

There are currently six types of interaction that PARETO can handle. The interactions are described in appendix B; they are shown in figure A.1, together with the possible information-acquisition subtasks that may arise out of the analysis, and the possible methods of avoiding genuine interactions (see section 7.3.2).

Default inference

Firby's original RAPs system automatically inferred default property values during memory retrieval. In PARETO this inference is performed as the result of explicit reasoning requests. The syntax of memory assertions concerning item properties is given in section A.4.3. The memory query

(infer-default*thing property-name property-specifiers answer*)

binds *answer* to the default value for *thing* of the property specified by *property-name* and *property-specifiers*. All variables except *answer* must be bound when this reasoning request is processed.

A.3 Task net descriptions

task-net-description ::= '(task-net' *task-net-step*$^+$ ')'

task-net-step ::= '(' *step-tag step-priority? index-clause annotations* ')'

step-tag ::= `symbol`

step-priority ::= `integer`

annotations ::= [[*temporal-annotation* resource-annotation* grouping-annotation**]]

Section 4.3.3 gives an overview of task nets. The *step-tag* must be unique within the task net and is used to identify the step in the annotations of other steps. If the *step-priority* is not supplied, it defaults to zero. It is added to the priority (see section 8.3.3) of the parent task to give the priority of the task represented by this step.

A.3.1 Temporal annotations

temporal-annotation ::= *ordering-annotation* | *start-annotation* | *end-annotation* | *window-annotation*
ordering-annotation ::= '(' *memory-formula*? 'for' *step-tag* ')'
start-annotation ::= '(until-start' *step-tag* ')'
end-annotation ::= '(until-end' *step-tag* ')'
window-annotation ::= '(window' *step-tag* *time-1* *time-2* ')'
time-1 ::= `integer`
time-2 ::= `integer`

All *step-tags* must refer to other steps in the same task-net. These annotations specify ordering constraints on tasks. The task represented by the step with the annotation must be processed before the task specified in an *ordering-annotation*; its processing must be terminated when the processing of the step indicated by a *start-annotation* starts; its processing must be terminated when the processing of the step indicated by an *end-annotation* is completed; and it must be processed between *time-1* and *time-2* time units after the step indicated by a window-annotation is completed. The *memory-formula* in an *ordering-annotation* describes an execution-time protection; it sets up an extra precondition for the indicated step. Free variables can be used to bind variables in the remainder of the task-net definition.

A.3.2 Resource annotations

resource-annotation ::= *set-add-effect* | *set-sub-effect* | *counter-add-effect* | *counter-sub-effect*
set-add-effect ::= '(set-add' *set-variable thing* ')'
set-sub-effect ::= '(set-sub' *set-variable thing* ')'
counter-add-effect ::= '(counter-add' *counter-variable amount* ')'
counter-sub-effect ::= '(counter-sub' *counter-variable amount* ')'
set-variable ::= `variable`
counter-variable ::= `variable`
thing ::= `variable`
amount ::= `variable` | `integer`

These annotations are used to update the resources described in section A.1.8. The resources are denoted by *set-variable* and *counter-variable,* which must be bound to resources of the appropriate type. The operations denoted by the annotations are performed on the successful completion of the step to which they are attached.

A.3.3 Grouping annotations

grouping-annotation ::= *clone-step-specifier* | *repeat-step-specifier*

clone-step-specifier ::= '(clone' *number* ')'
repeat-step-specifier ::= '(repeat' *number* ')'
number ::= variable | integer

These annotations were not in Firby's original language specification. They indicate the number of times a given step is to be performed: the *clone-step-specifier* simply ensures that the required number of copies of the step, and all its annotations, are placed on the task agenda; the *repeat-step-specifier* adds ordering constraints to the copies, to provide an iteration facility. If the *number* is a variable, it must be bound to an integer.

A.4 Memory queries

memory-formula ::= *belief-query* | *proposition* | *connective-query* | *progress-query* | *function-query*
proposition ::= *predicate* | *item-property*

Memory queries are used to find out whether there are matching assertions in PARETO's memory (see section 5.3.3); they are used to bind free variables in the usual way.

A.4.1 Belief queries

belief-query ::= '(believe' *proposition time-interval* ')'
time-interval ::= integer | *variable* | *arithmetic-term*

This query is used to investigate the age of assertions in memory. If *time-interval* is a free variable and there is a matching assertion then the query succeeds with *time-interval* being bound to the time since the assertion was made. If *time-interval* is bound, the query succeeds if there is a matching assertion in memory that was made within the specified amount of time.

A.4.2 Predicates

predicate ::= '(arm-at' *arm thing* ')' | '(arm-examined' *arm thing* ')' | '(arm-holding' *arm thing* ')' | '(arm-interference' *arm* ')' | '(arm-place' *arm place* ')' | '(direct-road' *start finish road* ')' | '(direction' *location road direction* ')' | '(examined' *thing aspect* ')' | '(examined-since-move' *thing aspect* ')' | '(examined-since-pour' *thing aspect* ')' | '(examined-since-toggle' *thing aspect* ')' | '(eye-examined' *thing aspect* ')' | '(eye-scanned' *location* ')' | '(my-fuel' *amount* ')' | '(my-heading' *direction* ')' | '(my-location' *location* ')' | '(my-speed' *speed* ')' | '(my-status' *status* ')' | '(my-time' *time* ')' | '(order-filled' *order-id thing* ')' | '(real-name' *internal-name external-name* ')' | '(scanned-since-move' *location* ')' | '(too-full-for' *place thing* ')' '(time-arrived' *location time* ')' | '(time-last-at' *location time* ')' | '(too-small-for' *place thing* ')'
amount ::= *variable* | rational
arm ::= *variable* | symbol
aspect ::= variable
direction ::= *variable* | symbol

external-name ::= *variable* | `symbol`

finish ::= *location*

internal-name ::= *variable* | `symbol`

order-id ::= *variable*

place ::= *variable* | `symbol`

road ::= *variable*

speed ::= *variable* | `symbol`

start ::= *location*

status ::= *variable* | `symbol`

thing ::= *variable*

time ::= *variable* | `integer`

The assertions matched by these queries are normal predicate calculus assertions. They do not describe physical properties of ordinary items. Free variables may be bound by the query. These predicates depend on the world in which PARETO operates: inasmuch as this differs from the world in which Firby's RAPs system operates, some of these predicates were not in the original language specification, or have a different syntax.

A.4.3 Item properties

item-property ::= '(birthday' *owner date* ')' | '(blade' *thing position* ')' | '(class' *thing class* ')' | '(color' *thing color* ')' | '(combination' *thing date* ')' | '(door' *thing state* ')' | '(liquid-held' *thing liquid* ')' | '(location' *thing place* ')' | '(made-of' *thing substance* ')' | '(maker' *thing maker* ')' | '(object-ok' *thing color type size truth-value* ')' | '(owner' *thing owner* ')' | '(paint' *thing state* ')' | '(quantity-held' *thing amount* ')' | '(road-type' *road road-type truth-value* ')' | '(size' *thing size* ')' | '(source-for' *type location truth-value* ')' | '(type' *thing type truth-value* ')'

amount ::= *variable* | `integer` | `symbol`

class ::= *variable* | `symbol`

color ::= *variable* | `symbol`

date ::= *variable* | `symbol`

liquid ::= *variable* | `symbol`

maker ::= *variable* | `symbol`

owner ::= *variable* | `symbol`

place ::= *location* | *thing*

position ::= *variable* | `symbol`

road-type ::= *variable* | `symbol`

size ::= *variable* | `integer` | `symbol`

state ::= *variable* | `symbol`

substance ::= *variable* | `symbol`

truth-value ::= *variable* | 'true' | 'false' | 'unknown'

type ::= *variable* | `symbol`

Item properties	Default value	Aspect to examine
birthday, combination, location	unknown	none
blade	depends on object	end
class, source-for, road-type, type	false	any
color	unknown	side
door	unknown	front
liquid-held, quantity-held	unknown	inside
made-of	depends on object	top
maker, owner, paint, size	unknown	top
object-ok	false	none

Figure A.2 Item properties

The assertions matched by these queries describe the properties of items in the world. Free variables may be bound by the query. These predicates depend on the world in which PARETO operates: inasmuch as this differs from the world in which Firby's RAPs system operates, some of these predicates were not in the original language specification, or have a different syntax. The general form of item properties is:

(property-name item-name [specifiers] property-value)

The memory system is set up to ensure that there is only ever one assertion for any given item property. Item properties have default values associated with them, which PARETO's reasoning mechanism makes use of as described in section 5.3.2. The default values are shown in figure A.2. As explained in section 5.3.2, the default mechanism is different from that used in Firby's original RAPs system. PARETO also knows which aspect of an object should be examined in order to find out about a given property; these are also shown in figure A.2. This is an extension of the original RAPs system.

A.4.4 Connectives

connective-query ::= '(and' memory-formula* ')' | '(not' memory-formula ')' | '(or' memory-formula* ')' |
    '(<' arithmetic-term arithmetic-term ')' | '(>' arithmetic-term arithmetic-term ')' |
    '(<=' arithmetic-term arithmetic-term ')' | '(>=' arithmetic-term arithmetic-term ')' |
    '(=' equality-term equality-term ')' | '(N=' equality-term equality-term ')'
equality-term ::= arithmetic-term | variable | `symbol`

These are the normal Boolean connectives. The "not" query succeeds if there is no matching assertion found in memory: it cannot be used to bind free variables. Free variables can be bound by both the "and" and "or" queries. The order of the arguments to these queries is significant in this regard. Except for those three, these queries cannot contain free variables.

Memory function	Semantics
(basic-place *place*)	Succeeds when *place* is one of (bay1 bay2 tire-bay fuel-gauge folded).
(examine-label *property-name aspect*)	The *aspect* of an object must be examined to find the value of the property with name *property-name*. Usually used to bind *aspect*.
(meets-criterion *thing criterion truth-value*)	The *truth-value* of whether *thing* meets the goal-criterion *criterion* is true, false, or unknown as appropriate. Can be used to bind *truth-value*.
(property-of *thing property-name property-specifiers*)	Succeeds when there is an assertion in memory about the property of *thing* specified by *property-name* and *property-specifiers*.
(unknown-prop-in-crit *thing criterion property-name*)	The value of the property *property-name* for *thing* is unknown, and is required in order to determine whether thing meets *goal-criterion* criterion. Usually used to bind *property-name*.
(examinable-property *property-name*)	Succeeds when the value of property *property-name* can be found out by examining an object.
(just-seen-p *thing*)	This is only used when the truck has just performed a scan operation: it succeeds if *thing* was seen.
(know-sensor-name *thing*)	Succeeds when the sensor-name of *thing* is known (see section 5.2.2)

Figure A.3 Memory functions

A.4.5 Arithmetic

arithmetic-term ::= '(+' *arithmetic-term arithmetic-term* ')' | '(-' *arithmetic-term arithmetic-term* ')' |
'(loc-interval)' | '(number-in-bay' *bay* ')' | '(size-of' *set-variable* ')' | '(time-now)' |
'(value' *counter-variable* ')' | *variable* | `number`
bay ::= *variable* | `symbol`

An arithmetic term returns a number instead of the usual Boolean result. They cannot contain free variables. The names of most of these queries are self-explanatory: loc-interval returns the time interval for scanning the environment (currently 15 time units).

A.4.6 Task progress

progress-query ::= '(last-result' *execution-result* ')' | '(method-tried' *method-name* ')' |
'(not-tried-yet' *method-name* ')' | '(result-with-method' *method-name execution-result* ')'
execution-result ::= *variable* | `symbol`

These queries return true or false and cannot contain free variables. They are used to inquire into the past history of the current task. For the possible *execution-results* arising from the performance of primitive actions, see section B.3.

A.4.7 Memory functions

function-query ::= *binding-function-query* | *non-binding-function-query*

binding-function-query ::= '(basic-place' *place* ')' | '(examine-label' *property-name aspect* ')' |
    '(meets-criterion' *thing criterion truth-value* ')' | '(member' *set-variable thing* ')' |
    '(property-of' *thing property-name property-specifiers* ')' | '(time-now-is' *time* ')' |
    '(unknown-prop-in-crit' *thing criterion property-name*')' |

non-binding-function-query ::= '(empty' *set-variable* ')' | '(examinable-property' *property-name* ')' |
    '(just-seen-p' *thing* ')' | '(know-sensor-name' *thing* ')'

property-name ::= *variable* | `symbol`

property-specifiers ::= *variable*

These queries return true or false. The *binding-function-queries* can be used to bind free variables; the *non-binding-function-queries* cannot contain free variables. Figure A.3 explains those queries whose meaning is not obvious from their names. Except for empty, just-seen-p, and know-sensor-name, either these queries were not present in Firby's original language specification or their definitions have changed.

APPENDIX B

PARETO'S WORLD AND REFERENCE FEATURES

This appendix describes the world in which PARETO operates, and how PARETO interacts with it. PARETO's world was built using a slightly modified version of the TRUCKWORLD simulator (Firby and Hanks 1987). Significant changes from the original version are noted in the descriptions in this appendix. The premise of the simulation is that a robot delivery truck travels between locations on a network of roads, encountering and manipulating various objects as it goes. PARETO, which controls the truck, can only interact with the world through the truck's actions. Aspects of PARETO's world are discussed in chapters 4 and 5, especially sections 4.1.1 and 5.2.

B.1 Locations and roads

Figure B.1 shows the world map. There are 13 locations, connected by 16 roads. The top three locations, maple-ave, newport-ave, and sheridan-rd, are the locations to which deliveries must be made. The center two locations, base and bank, are where the truck can refuel and where the safes and gold bars are kept (see section 4.4.1). The other eight locations are where objects to be delivered can be found. Section B.1.1 describes the locations in more detail, section B.1.2 describes the roads, and section B.1.3 describes global characteristics that apply to the whole world.

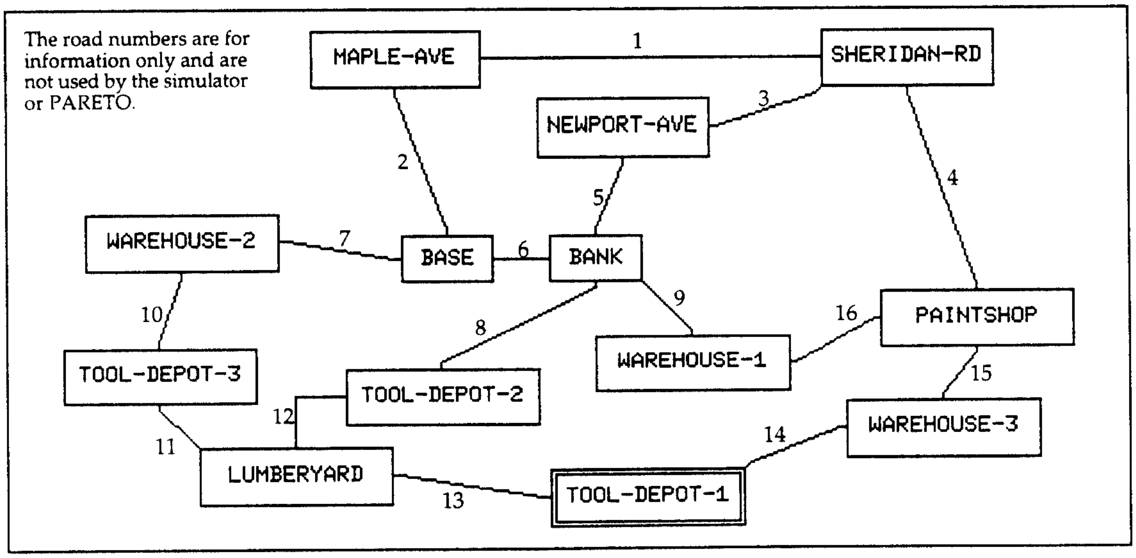


Figure B.1 A map of PARETO's world

Location	Immovable objects	Description
maple-ave	site	The ils delivery site
newport-ave	site	The house delivery site
sheridan-rd	site	The tech delivery site
bank	book safe safe safe	The directory containing names and birthdays. safe-1: maker chubb, owner mary, combination comb-12-06-59 safe-2: maker chubb, owner john, combination comb-07-10-60 safe-3: maker acme, owner mary, combination comb-12-06-59

Figure B.2 Immovable objects by location

B.1.1 Locations

Every location contains some immovable objects, many of which (paint-machine, road-sign, sapling, street-light, and tree) have no reference features other than their class and cannot be manipulated by the truck. Figure B.2 shows the other immovable objects by location.

Some locations contain *restockers*: invisible objects that generate new instances of a given type of object. Such locations are then sources for that type of object. PARETO knows that some of these are good places to look for those objects. This information is supplied to PARETO when it is started up, in the form of source-for assertions in memory (see section A.4.3). Each restocker has a fixed stock level: if there are already a certain number of objects of the relevant class present at the location, the restocker will not generate new ones. Restockers operate randomly: each has a fixed probability of generating a new object in every period of 60 time units. Figures B.3 (lumberyard and paintshop), B.4 (tool-depots), and B.5

Location	Object restocked	Prob.	Stock level	Description
lumberyard	wood	15%	1	Size 5, unpainted.
	wood	15%	2	Size 6, unpainted.
	wood	12%	2	Size 7, unpainted.
	wood	20%	1	Size 8, unpainted.
	box	8%	1	
	ladder	10%	1	
paintshop	wood	15%	2	Size 6, wet paint. Color depends on colors of pieces of wood present and color of last piece generated.
	wood	15%	2	Size 7, wet paint. Color depends on colors of pieces of wood present and color of last piece generated.
	paint	10%	1	Blue.
	paint	8%	1	Red.
	paint	15%	1	Yellow.
	paintbrush	15%	2	
	solvent	15%	2	
	wallpaper	15%	3	

Figure B.3 Restockers at the lumberyard and paintshop

Location	Object restocked	Prob.	Stock level	Description
tool-depot-1	bag	5%	1	Randomly made of fabric (1/3) or leather (2/3).
	chisel	5%	1	
	hammer	15%	2	
	knife	15%	2	Blade is extended.
	paintbrush	5%	1	
	saw	5%	1	
	screwdriver	5%	1	
tool-depot-2	chisel	5%	1	
	hammer	5%	1	
	knife	5%	1	Blade is extended.
	saw	15%	2	
	scissors	15%	2	
	screwdriver	15%	2	
tool-depot-3	chisel	15%	2	
	hammer	15%	2	
	knife	5%	1	Blade is extended.
	scissors	5%	1	
	wheelbarrow	15%	2	

Figure B.4 Restockers at the tool depots

(warehouses) give details of the restockers. There are no restockers at the base or bank. However, empty fuel drums left at the base are refilled.

Location	Object restocked	Prob.	Stock level	Description
warehouse-1	bucket	15%	2	
	scissors	5%	1	
	screws	15%	2	
	tape	5%	1	
	twine	5%	1	
warehouse-2	bag	15%	2	Randomly made of fabric (1/3) or leather (2/3).
	nails	5%	1	
	screws	5%	1	
	tape	5%	1	
	twine	15%	2	
warehouse-3	nails	15%	2	
	screws	5%	1	
	tape	5%	1	
	twine	15%	2	

Figure B.5 Restockers at the warehouses

Road	Location and direction		Location and direction		Length	Attributes
1	maple-ave	e	sheridan-rd	w	15	windy
2	maple-ave	s	base	n	20	
3	sheridan-rd	sw	newport-ave	w	15	windy
4	sheridan-rd	s	paintshop	n	30	
5	newport-ave	s	bank	n	20	
6	base	e	bank	w	15	
7	base	w	warehouse-2	e	15	
8	bank	s	tool-depot-2	n	10	bumpy, windy
9	bank	se	warehouse-1	n	25	
10	warehouse-2	s	tool-depot-3	n	15	
11	tool-depot-3	s	lumberyard	nw	15	bumpy, windy
12	tool-depot-2	w	lumberyard	n	15	
13	lumberyard	e	tool-depot-1	w	5	
14	tool-depot-1	ne	warehouse-3	w	20	
15	warehouse-3	n	paintshop	s	20	windy
16	warehouse-1	e	paintshop	w	15	

Figure B.6 The roads in PARETO's world

The objects in PARETO's world are described in section B.3, which also describes which locations PARETO believes to be sources for each class of object. PARETO does not attach reference features directly to locations (see section 6.2.1).

B.1.2 Roads

Each road connects two locations. It leaves each location in one of eight directions. It has a length, which PARETO does not know. The time taken to traverse a road, and the amount of fuel used, depend on the length of the road and the speed at which the truck is traveling, with a random element.

A road may have one or more attributes, which, together with the speed at which the truck is traveling, affect the likelihood of an accident. PARETO uses these attributes, which it finds out by examining the road (see section B.3.3), to set the truck's speed for each road. If the truck can fold its arms, and the road is neither bumpy nor windy, then the speed is set to fast.[1] If the road is both bumpy and windy, the speed is set to slow. In all other cases the speed is set to medium.

Figure B.6 shows the roads in PARETO's world, with the locations they join, their lengths, and their attributes. The road numbers match the numbers in figure B.1—these numbers are for information only and are not used by the simulator or PARETO. Each road has an internally assigned name that is used by the simulator and PARETO and that changes from run to run.

[1] The truck can travel fast only if its arms are folded. It can fold its arms only if they are empty.

B.1.3 Global characteristics

The chief global characteristic of PARETO's world is its dynamism—the tendency of objects to change locations regardless of the truck's actions (see section 5.2.2). This happens by every so often randomly removing an object from the truck's current location and randomly replacing an object that had previously been removed from somewhere. In PARETO's world, the frequency with which this occurs is every 5 time units. At every location except the base and bank, there is a 10% probability that an object will be removed; if an object is to be removed, one is chosen randomly from the objects that are present. Similarly, there is a 10% probability that a new object will appear at the current location; if an object is to appear, one is chosen randomly from objects that previously have been removed from any location. At the base and bank, the probabilities are 5%.

B.2 Objects and interactions

There are objects throughout PARETO's world. They were built using the tools provided by the TRUCKWORLD simulator: many of them were specially designed for the work described here.

B.2.1 Object types

The objects are classified in a hierarchy of *types*: the lowest type in the hierarchy is the object's *class*. Properties are inherited through the type hierarchy. Most classes of objects are simply ; some are also containers (which can hold other objects) or vessels (which can hold liquids); others are colored-objects or paintable-objects. The properties of these types are shown in figure B.7, together with those objects having properties other than those of simple physical-objects.

Type	Inherits from	Property	Comments
physical-object		bigness	A measure of the object's size; includes the bigness of the object's contents, if any.
container	physical-object	capacity	The maximum allowed total bigness of the container's contents.
vessel	physical-object	capacity	The maximum amount of liquid that the vessel can hold.
		liquid-held	The type of liquid contained in the vessel.
colored-object	physical-object	color	
paintable-object	colored-object	wet-paint	Whether the object's paint is wet or dry.
		time-painted	The time at which the object was painted.
bag	physical-object	made-of	Either leather or fabric. Fabric bags may be ripped by saws or chisels.
		ripped	Either true or false.
fuel-drum	vessel		
gold-bar	colored-object		
knife	physical-object	blade	Either extended or retracted.
paint	colored-object	lid-open	Either true or false. The paint is spilled if it is moved while the lid is open.
paint-machine	container		
safe	container	door	Either open or locked.
		owner	
		maker	
		combination	The owner's birthday.
site	container		Objects that achieve delivery goals by being placed inside a site disappear.
wood	paintable-object		

Figure B.7 Object types

B.2.2 Reference features

As described in section 6.2, PARETO attaches reference features to the objects in its world. It also considers some locations to be sources for certain objects (see section B.1.1). Figures B.8 and B.9 show this information for all objects in PARETO's world.

	Object	Reference features	Source location
	bag	carrier, bag, fabric	warehouse-2
	box	carrier, box	lumberyard
	bucket	carrier, bucket	warehouse-1
	chisel	prying, chisel, sharp	tool-depot-3
	hammer	prying, hammer	tool-depot-1
	knife	sharp, knife	tool-depot-1
	ladder	ladder, large	lumberyard
NAILS	nails	fastener, nails	warehouse-3
	paint	paint, colored	paintshop
	paintbrush	paintbrush	paintshop
	saw	saw, sharp	tool-depot-2
	scissors	sharp, prying, scissors	tool-depot-2
	screwdriver	prying, screwdriver	tool-depot-2
SCREWS	screws	fastener, screws	warehouse-1
	solvent	solvent	paintshop
	tape	fastener, tape	warehouse-3
	twine	fastener, twine, soft	warehouse-2
	wallpaper	wallpaper fabric	paintshop
	wheelbarrow	carrier, wheelbarrow, large	tool-depot-3
	wood	wood, colored	lumberyard, paintshop

Figure B.8 Delivery objects, reference features, and source locations

Object	Reference features	Comments
book	book	Used to look up details of birthdays to find safe combinations.
fuel-drum	fuel-drum, fuel	Contain fuel.
gold-bar	gold-bar	
mud-tires	tires	The truck must have a set of tires in its tire bay in order to move.
paint-machine		Used to change the color of paintable objects.
road-sign		
safe	safe	
sapling		
site	site	
standard-tires	tires	The truck must have a set of tires in its tire bay in order to move.
street-light		
tree		

Figure B.9 PARETO's reference features—other objects

B.2.3 Interactions between objects

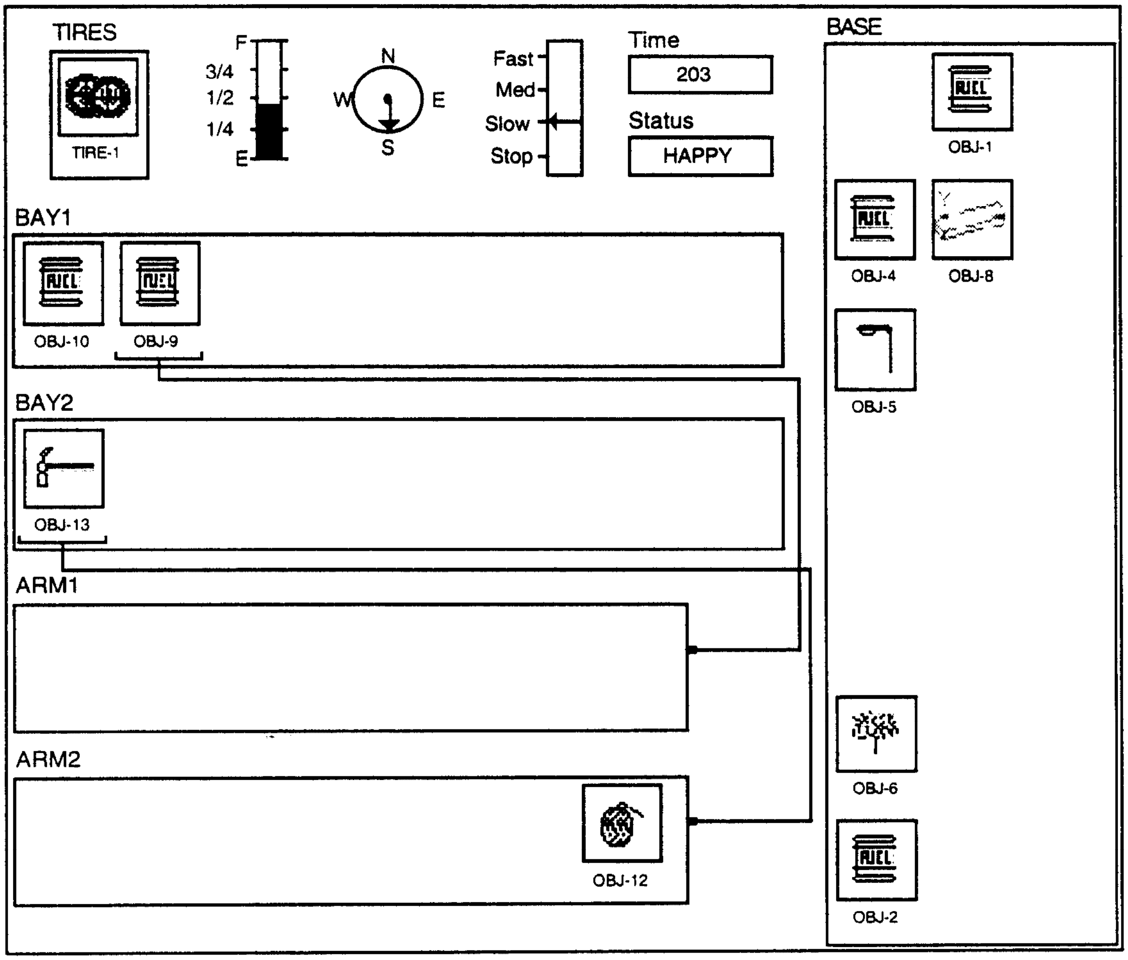
Objects may interact with each other in various ways. These interactions take place when the objects concerned are both in the same location, and that location is one of the truck's bays or arms, or inside another object. Figure B.10 shows the circumstances under which these interactions occur. The reference features that PARETO uses to indicate potential interactions (see chapter 7) are shown in figure B.11.

Interaction type	Objects involved	Circumstances	Results
cut	knife, twine knife, wallpaper	Knife's blade extended. Knife's blade extended.	Twine disappears. Wallpaper disappears.
puncture	chisel, bag chisel, wallpaper	Bag made of fabric.	Bag becomes ripped. Wallpaper disappears.
rip	saw, bag saw, wallpaper	Bag made of fabric.	Bag becomes ripped. Wallpaper disappears.
smear	paintable-object	Does not occur inside a paint-machine. Object has wet paint.	Color of object becomes smeared.
stain	solvent, paintable-object	Color of object is not unpainted.	Color of object becomes stained.

Figure B.10 Interactions between objects

Interaction type	Roles	Reference features
cut	instrument object	sharp soft
fill	instrument	large
move	instrument	change-loc
puncture	instrument object	pointed fibrous
rip	instrument object	sharp fibrous
smear	instrument	painted
stain	instrument object	solvent painted

Figure B.11 Interactions and their reference features



Time
203

Status
HAPPY

Figure B.12 The truck

Command	Sensor data	Action results	Duration
(truck-turn ?direction)	none	okay	2
(truck-set-speed ?speed)	none	okay	1
(truck-move)	new-location	okay truck-cant-move truck-mishap	depends on distance traveled
(wait ?delay)	none	okay	depends on ?delay

Figure B.13 Truck movement actions

B.3 The truck

The truck display is shown in figure B.12. The truck has the following components:

- Tire bay;
- Fuel gauge;
- Direction gauge;
- Speed gauge;
- Two cargo bays;
- Two arms.

In addition, the display shows the current time and the objects that are present at the truck's current location.

The display shows the objects in each cargo bay and arm. Both cargo bays and arms are of limited capacity: the total size of objects in bay1 cannot exceed 20 units, and in bay2 cannot exceed 30 units. Arm1 can hold objects of size up to 8 units, and arm2 up to 20 units. The arms are clumsy: each time an arm attempts to grasp an object (see section B.3.2), there is a chance that the grasping action will fail and the object will be dropped. Arm1 has a probability of 15% of dropping something, and arm2 a probability of 50%.

The actions that the truck can perform divide in three groups: those connected with movement, those connected with the manipulation of objects, and those connected with perception. There are two types of information that may result from the performance of an action: sensor data and proprioceptive feedback (see section 5.3.1). Each action takes a specific duration to perform: it is through executing actions that time passes in the simulator. In the rest of this section, all the actions that can be performed by the truck are described, showing their syntax and the possible results.

B.3.1 Movement actions

The truck moves from location to location along the roads connecting them. In order to move along a road, it must be facing in the correct direction and must have set its speed appropriately. The truck's movement actions are shown in figure B.13.

Truck-turn

The truck turns to face the specified direction, which may be any of: n, ne, e, se, s, sw, w, nw. The result is undefined if any other direction is specified.

Truck-set-speed

The truck's speed is set to that specified, which may be any of: stop, slow, medium, fast. The result is undefined if any other speed is specified.

Truck-move

The truck moves down the road that leaves the current location at the truck's current heading. The truck does not move (and the result of the action is okay) if there is no road with the relevant heading. The truck cannot move, and the result of the action is truck-cant-move, in the following circumstances:

- The truck's speed is set to stop;
- The truck has run out of fuel;
- The truck has no tires;
- One of the truck's arms is inside another object in the truck's current location;
- The truck's speed is set to fast and one of the truck's arms is extended.

In all other circumstances, the truck moves along the road. The duration of the action and the amount of fuel used both depend on the length of the road, the characteristics of the road and the speed at which the truck is traveling, subject to a random adjustment. If the truck has an accident or runs out of fuel along the way, the action result is truck-mishap, and the truck ends up in a temporary location in the middle of the road. If this occurs because the truck has run out of fuel, and it has some fuel with it, then it can refuel and continue: otherwise the run is over.

Wait

The truck waits for the specified duration.

B.3.2 Manipulation actions

To manipulate an object, one of the truck's arms must be moved to the object. An object can be grasped by an arm, and then moved to another place; the arm can pour or ladle liquids from one vessel to another; and it can toggle objects, the result of which depends on the particular object involved. The manipulation actions are shown in figure B.14; none of them result directly in the receipt of sensor data.

Arm-move

The arm that is to be moved may be either arm1 or arm2. The place to which it is to be moved may be one of the following:

- folded: the arm is retracted;
- external: the arm is moved to the area outside the truck;
- bay1, bay2, fuel-gauge, tire-bay, bay1, bay2: the arm is moved to the relevant area of the truck;
- An object: the arm is moved to that object;
- inside: if the arm is currently at an object, it is moved inside it;
- outside: if the arm is currently inside an object, it is moved outside it.

The arm does not move and the result of the action is arm-cant-move in the following circumstances:

- The place specified is not one of those listed above;
- The place is folded and the arm is not empty;
- The place is inside and the arm is currently at an object that has no interior or whose interior is not accessible;
- The place is outside and the arm is not currently inside anything;
- The arm is currently folded and the place is an object.

The arm does not move and the result of the action is arm-cant-find if the place is an object and that object cannot be found at the arm's current location.

Command	Action results	Duration
(arm-move ?arm ?place)	okay arm-cant-move arm-cant-find	3
(arm-grasp ?arm ?thing)	okay arm-not-at arm-too-full arm-interference arm-dropped	2
(arm-ungrasp ?arm ?thing)	okay arm-not-holding container-full	2
(arm-pour ?arm ?thing)	okay arm-not-holding	3
(arm-ladle ?arm ?thing)	okay arm-not-holding arm-not-at	3
(arm-toggle ?arm ?thing ?message)	okay arm-not-holding arm-cant-toggle	1

Figure B.14 Manipulation actions

Arm-grasp

To grasp an object, an arm must be positioned at that object: otherwise the object is not grasped and the result of the action is arm-cant-find. If the space available in the arm (allowing for the arm's capacity and the other objects that it is already holding) is less than the size of the object, the object is not grasped and the result of the action is arm-too-full. If the other arm is also positioned at the object, the object is not grasped and the result of the action is arm-interference. If the object is dropped due to the clumsiness of the arm (see section B.2.1) the result of the action is arm-dropped.

Arm-ungrasp

To ungrasp an object, the arm must currently be holding it: otherwise the result of the action is arm-not-holding. The object is deposited in the arm's current location; if the space available is less than the size of the object, the object remains in the arm and the result of the action is container-full.

Arm-pour

To pour liquid from a vessel, the vessel must be held by the arm: otherwise no liquid is poured and the result of the action is arm-not-holding. The liquid is transferred into the object at which or in which the arm is positioned, if that object is of a type that can hold liquids. If there is no such object, the liquid is simply poured away and the result of the action is okay. If the object already holds some liquid of a type different from that being poured from the vessel, no liquid is transferred and the result of the action is okay.

Arm-ladle

To ladle liquid into a vessel, the vessel must be held by the arm: otherwise no liquid is ladled and the result of the action is arm-not-holding. The liquid is transferred from the object at which or in which the arm is positioned, if that object is of a type that can hold liquids. If the arm is not positioned at such an object,

Object	Message	Current state of the object	Result of toggling action
knife	extend	retracted	The action extends the blade.
	retract	extended	The action retracts the blade.
paint-machine	open	Contains a closed can of paint and one of hammer, chisel, screwdriver.	The can of paint is opened.
	close	Contains an open can of paint.	The can of paint is closed.
	Any color	Contains an open can of paint of the correct color and an unpainted paintable-object.	The object is painted.
safe	The safe’s combination	locked	The action unlocks the safe.
	Any	unlocked	The action locks the safe.

Figure B.15 Toggling actions

the result of the action is arm-not-at. If the vessel already holds some liquid of a type different from that to be ladled from the object, no liquid is transferred and the result of the action is okay.

Arm-toggle

To toggle an object the arm must already be holding that object, or must be positioned at the object: otherwise the result of the action is arm-not-holding. The result of the toggling action depends on the object being toggled and the message with which it is toggled as shown in figure B.15 (the objects are described further in section B.2). If the toggling action specified is not possible, the result of the action is arm-cant-toggle.

B.3.3 Perception actions

The perception actions are intended to result in the receipt of sensor data. In general, the truck can only receive data from objects in its immediate surroundings. The perception actions are shown in figure B.16.

Eye-scan

The place that is scanned must be one of tire-bay, bay1, bay2, arm1, arm2, external: otherwise, no sensor data results from the action. An object-seen sensor datum is received for each object present in the place that is scanned. However, no sensor datum is received for objects that are not directly visible because they are inside something else. The object identifier is the sensor-name (see section 5.2.2). The objects in the world are classified in a type hierarchy: the class that is shown in the object-seen sensor datum is the most

Action	Sensor data	Results	Duration
(eye-scan ?place)	(object-seen *place object-id object-class*) (road-seen *direction road-name*)	okay	4
(eye-examine ?place ?thing ?message)	Depends on ?thing and ?message	okay eye-cant-find	1
(arm-examine ?arm ?thing ?message)	Depends on ?thing and ?message	okay arm-not-at	1

Figure B.16 Perception actions

Object type	Aspect examined	Sensor data
road	any	(road-type *road-id attribute*)
physical-object	any	(object-seen *place object-id class*)
	any	(thing *object-id class*)
	top	(bigness *container-id size*)
container	any	(thing *container-id* container)
	inside	(object-seen *container-id object-id class*)
vessel	any	(thing *vessel-id* vessel)
	inside	(liquid-held *vessel-id liquid amount*)
colored-object	any	(thing *object-id* colored)
	side	(color *object-id color*)
paintable-object	top	(paint *object-id state*)
bag	top	(bag *bag-id* made-of *stuff*)
	side	(bag *bag-id* ripped true)
book	a name	(book *bood-id* birthday *who when*)
knife	end	(knife *knife-id* blade *state*)
paint	top	(paint *object-id* lid *state*)
safe	front	(safe *safe-id* door *state*)
	top	(safe *safe-id* maker *who*)
	top	(safe *safe-id* owner *who*)

Figure B.17 Sensor data from examining objects

specific type applicable to the object.

Eye-examine

The examining action must specify both the place where the object can be found and the aspect of the object that should be examined. The latter requirement is an extension to the original TRUCKWORLD simulator. An object can be examined if it is in any place that can be scanned (see above). If the specified object is not in fact in the specified place no sensor data is received and the result of the action is eye-cant-find. Roads leading out of the current location can also be examined (this is an extension to the original TRUCKWORLD simulator). Figure B.17 shows what sensor data are received from various types of objects (recall there is a hierarchy of types as described in section B.2; all relevant sensor data are received).

Arm-examine

The arm-examine action is similar to the eye-examine action, in that it results in the receipt of the same sensor data (see figure B.17). If the arm is neither holding nor positioned at the object to be examined, no sensor data is received and the result of the action is arm-not-at.

B.4 Delivery goals

The premise underlying PARETO's world is that the three sites, ils at maple-ave, tech at sheridan-rd, and house at newport-ave, are populated by construction workers who ask the truck to fetch objects that they need for their work.

Goal criterion	Conditions of satisfaction	Reference features
blue-paint	(and (class ?object paint) (color ?object blue))	paint, blue, colored
blue-wood	(and (class ?object wood) (color ?object blue))	wood, blue, colored
carry-dirt	(or (class ?object box) (class ?object bucket))	carrier
carry-tools	(or (class ?object box) (class ?object bag))	carrier
carry-water	(class ?object bucket)	carrier
cut-twine	(or (class ?object scissors) (class ?object knife))	sharp
open-paint	(or (class ?object hammer) (class ?object screwdriver) (class ?object chisel))	prying
open-solvent	(or (class ?object scissors) (class ?object screwdriver))	prying
perm-fastener	(or (class ?object nails) (class ?object screws))	fastener
red-paint	(and (class ?object paint) (color ?object red))	paint, red, colored
red-wood	(and (class ?object wood) (color ?object red))	wood, red, colored
temp-fastener	(or (class ?object twine) (class ?object tape))	fastener
wood-5	(and (class ?object wood) (size ?object 5))	wood
wood-6	(and (class ?object wood) (size ?object 6))	wood
wood-7	(and (class ?object wood) (size ?object 7))	wood
wood-8	(and (class ?object wood) (size ?object8))	wood
yellow-paint	(and (class ?object paint) (color ?object yellow))	paint, yellow, colored
yellow-wood	(and (class ?object wood) (color ?object yellow))	wood, yellow, colored

Figure B.18 Goal-criterion reference features

B.4.1 Notification and achievement

Delivery goals are generated at random between time 90 and time 300. During this period the probability of at least one goal being generated in an interval of 60 time units is 90%. New goals are notified to the truck through sensor data; the information about each goal consists of an identifier for the goal, a goal-criterion that must be met by the object that is delivered (see section 4.3), the priority to be attached to the achievement of the goal (see section 8.3.3), and the site to which the delivery should be made.

Goal achievement is also notified through sensor data; the information supplied is the identifier of the goal that has been achieved and the object through whose delivery it was achieved. A goal is achieved when an object that meets the appropriate goal-criterion is placed inside the site to which the delivery must be made.

B.4.2 Goal criteria

The goal-criteria describing the objects whose delivery will achieve the goals describe the functions they will notionally perform for the supposed construction workers, and do not necessarily bear any relation to the ways in which they can be manipulated by the truck (see section B.3).

There are two types of goal-criterion: simple goal-criteria consist simply of the class of object that must be delivered, while complex goal-criteria describe more complex conditions that must be met. PARETO attaches the same reference features to simple goal-criteria as it attaches to the class of object that must be delivered: they are shown in figure B.8. Figure B.18 shows the possible complex goal-criteria, the conditions of meeting them, and the reference features that PARETO attaches to them.

APPENDIX C

A SPECIMEN RUN OF PARETO

This appendix gives a trace of one of PARETO's runs. It shows PARETO seizing opportunities, analyzing potential interactions, and ensuring that the truck does not run out of fuel. The output produced by PARETO is in `typewriter` font; output from the simulator is in sans serif; and comments are in *italics*.

C.1 Initialization

PARETO initializes itself by waking up, starting up some tasks to achieve preservation goals, and ensuring its fuel tank is full. The initialization process takes 117 time units. Towards the end of this period, new delivery goals are received.

C.1.1 Waking up

PARETO's first task is to wake up; it checks its location, where its arms are, and its heading and speed. Next, the tasks to keep track of its surroundings (MONITOR-CURRENT-LOCATION) and not to run out of fuel (MONITOR-FUEL-LEVEL) are started. A subtask of the latter is to ensure that the truck is carrying two full drums of fuel (REPLENISH-FUEL-RESERVES).

```
--> New top level goal: WAKEUP ::  [0:0]
    === New main task chosen at 0 :  <WAKEUP> [0:0]
    *** Addressing goal 0 <WAKEUP> at 0
        ... Processing task: <WAKEUP> ::  [0:0] at 0
            *** Arrived at NIL at time 0
        ... Processing task: <WAKEUP> ::  [0:0] at 19
        ... Task succeeded: <WAKEUP> ::  [0:0] at 19
--> Done.
--> New top level goal: MONITOR-CURRENT-LOCATION ::  [0:0]
```

This task makes the truck scan its surroundings every 15 time units and when it arrives at a new location.

```
--> Done.
--> New top level goal: MONITOR-FUEL-LEVEL ::  [1:1]
```

The truck must ensure that it refills its fuel tank if the level gets too low, and that it has reserves of two full fuel drums.

```
    === New main task chosen at 19 :  <MONITOR-FUEL-LEVEL> [1:1]
    *** Addressing goal 1 <MONITOR-FUEL-LEVEL> at 19
        ... Processing task: <MONITOR-FUEL-LEVEL> ::  [1:1] at 19
        ... Processing task: <REPLENISH-FUEL-RESERVES [SET: NIL]>
              ::  [1:3] at 19
    <=> Changing goals at 20 from 1 to 0 <MONITOR-CURRENT-LOCATION>
```

The MONITOR-CURRENT-LOCATION *task interrupts whatever other task PARETO is addressing.*

```
        ... Processing task: <MONITOR-CURRENT-LOCATION>
              ::  [0:0] at 20
```

APPENDIX C

A SPECIMEN RUN OF PARETO

This appendix gives a trace of one of PARETO's runs. It shows PARETO seizing opportunities, analyzing potential interactions, and ensuring that the truck does not run out of fuel. The output produced by PARETO is in `typewriter` font; output from the simulator is in sans serif; and comments are in *italics*.

C.1 Initialization

PARETO initializes itself by waking up, starting up some tasks to achieve preservation goals, and ensuring its fuel tank is full. The initialization process takes 117 time units. Towards the end of this period, new delivery goals are received.

C.1.1 Waking up

PARETO's first task is to wake up; it checks its location, where its arms are, and its heading and speed. Next, the tasks to keep track of its surroundings (MONITOR-CURRENT-LOCATION) and not to run out of fuel (MONITOR-FUEL-LEVEL) are started. A subtask of the latter is to ensure that the truck is carrying two full drums of fuel (REPLENISH-FUEL-RESERVES).

```
--> New top level goal: WAKEUP ::  [0:0]
    === New main task chosen at 0 :  <WAKEUP> [0:0]
    *** Addressing goal 0 <WAKEUP> at 0
        ... Processing task: <WAKEUP> ::  [0:0] at 0
            *** Arrived at NIL at time 0
        ... Processing task: <WAKEUP> ::  [0:0] at 19
        ... Task succeeded: <WAKEUP> ::  [0:0] at 19
--> Done.
--> New top level goal: MONITOR-CURRENT-LOCATION ::  [0:0]
```

This task makes the truck scan its surroundings every 15 time units and when it arrives at a new location.

```
--> Done.
--> New top level goal: MONITOR-FUEL-LEVEL ::  [1:1]
```

The truck must ensure that it refills its fuel tank if the level gets too low, and that it has reserves of two full fuel drums.

```
    === New main task chosen at 19 :  <MONITOR-FUEL-LEVEL> [1:1]
    *** Addressing goal 1 <MONITOR-FUEL-LEVEL> at 19
        ... Processing task: <MONITOR-FUEL-LEVEL> ::  [1:1] at 19
        ... Processing task: <REPLENISH-FUEL-RESERVES [SET: NIL]>
             ::  [1:3] at 19
    <=> Changing goals at 20 from 1 to 0 <MONITOR-CURRENT-LOCATION>
```

The MONITOR-CURRENT-LOCATION *task interrupts whatever other task PARETO is addressing.*

```
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 20
```

```
    --- Abandoning side task at 24 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 24 from 0 to 1 <MONITOR-FUEL-LEVEL>
```

Having scanned its surroundings, PARETO returns to the task at hand.

(FUEL-DRUM OBJ-17 disappeared at time 34)

```
    <=> Changing goals at 40 from 1 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 40
    --- Abandoning side task at 44 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 44 from 0 to 1 <MONITOR-FUEL-LEVEL>
    <=> Changing goals at 60 from 1 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 60
    --- Abandoning side task at 64 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 64 from 0 to 1 <MONITOR-FUEL-LEVEL>
    --- Abandoning main task at 70 :  REPLENISH-FUEL-RESERVES  [1:3]
    --- Abandoning main task at 70 :  MONITOR-FUEL-LEVEL  [1:1]
--> Done.
```

There are now two full fuel-drums in the truck's bay1.

C.1.2 Filling the tank

The final initialization task is to fill up with fuel (DUMP-IN-LOCAL-FUEL-DRUMS).

```
--> New top level goal: DUMP-IN-LOCAL-FUEL-DRUMS ::  [5:5]
    === New main task chosen at 70 :  <DUMP-IN-LOCAL-FUEL-DRUMS> [5:5]
        ... Processing task: <DUMP-IN-LOCAL-FUEL-DRUMS>
             ::  [5:5] at 70
    <=> Changing goals at 81 from 5 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 81
    --- Abandoning side task at 85 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 85 from 0 to 5 <DUMP-IN-LOCAL-FUEL-DRUMS>
    >>> Main task:  [5:6] supplanted by  [1:4]
        <PUT-DOWN-EMPTIES>
```

PARETO has a high priority task (spawned from the MONITOR-FUEL-LEVEL *task) to put down empty fuel drums after they have been used.*

```
    <=> Changing goals at 88 from 5 to 1 <MONITOR-FUEL-LEVEL>
        ... Processing task: <PUT-DOWN-EMPTIES> ::  [1:4] at 88
        ... Processing task: <PUT-DOWN-EMPTIES> ::  [1:4] at 89
    --- Abandoning main task at 94 :  PUT-DOWN-EMPTIES  [1:4]
    <=> Changing goals at 94 from 1 to 5 <DUMP-IN-LOCAL-FUEL-DRUMS>
--> New order ORDER-1: a SAW for ILS received at time 95
    There are now 1 orders outstanding
--> New top level goal: DELIVER-OBJECT ::  [9:9]
```

A delivery goal is received, but is ignored while refueling continues.

```
        ... Processing task: <DUMP-IN-LOCAL-FUEL-DRUMS>
             ::  [5:5] at 95
    <=> Changing goals at 101 from 5 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 101
    --- Abandoning side task at 105 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 105 from 0 to 5 <DUMP-IN-LOCAL-FUEL-DRUMS>
    >>> Main task:  [5:6] supplanted by  [1:4]
        <PUT-DOWN-EMPTIES>
    <=> Changing goals at 111 from 5 to 1 <MONITOR-FUEL-LEVEL>
        ... Processing task: <PUT-DOWN-EMPTIES> ::  [1:4] at 111
        ... Processing task: <PUT-DOWN-EMPTIES> ::  [1:4] at 112
    --> New side task chosen at 117 :
        <DUMP-IN-LOCAL-FUEL-DRUMS> [5:5]
    <=> Changing goals at 117 from 1 to 5 <DUMP-IN-LOCAL-FUEL-DRUMS>
        ... Processing task: <DUMP-IN-LOCAL-FUEL-DRUMS>
             ::  [5:5] at 117
        ... Task succeeded: <DUMP-IN-LOCAL-FUEL-DRUMS>
             ::  [5:5] at 117
```

The truck's fuel tank is now full, and it is holding no empty fuel-drums

```
    --- Abandoning main task at 117 :  PUT-DOWN-EMPTIES  [1:4]
```

C.2 Starting the deliveries

After the initialization process PARETO is free to address other goals that may arise. In fact, it already has one: to deliver a saw to ils.

C.2.1 Finding an object to deliver

The first step in this task is to find a saw; PARETO has not yet seen one, but believes that they are often to be found at tool-depot-2.

```
    === New main task chosen at 117 :
        <DELIVER-OBJECT SAW ILS ORDER-1> [9:9]
    *** Addressing goal 9 <DELIVER-OBJECT SAW ILS ORDER-1> at 117
        ... Processing task: <DELIVER-OBJECT SAW ILS ORDER-1>
             ::  [9:9] at 117
        ... Processing task: <FIND-OBJECT SAW => ?PLACE-NAME ?OBJECT>
             ::  [9:5] at 117
        ... Processing task: <TRUCK-TRAVEL-TO TOOL-DEPOT-2>
             ::  [9:10] at 117
```

The truck decides to set off for TOOL-DEPOT-2.

```
    <=> Changing goals at 122 from 9 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 122
--> New order ORDER-2: a roll of WALLPAPER for TECH received
    at time 126
    There are now 2 orders outstanding
--> New top level goal: DELIVER-OBJECT ::  [24:24]
```

Another delivery request is received, but is ignored for the time being as PARETO continues to address the task of finding a saw.

```
    --- Abandoning side task at 126 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 126 from 0 to 9
        <DELIVER-OBJECT SAW ILS ORDER-1>
            *** Arrived at BANK at time 144
```

The truck arrives at the bank *en route to* tool-depot-2.

```
--> New order ORDER-3: a roll of TAPE for HOUSE received at time 144
    There are now 3 orders outstanding
--> New top level goal: DELIVER-OBJECT ::  [15:15]
```

Delivery requests continue to be received and ignored.

```
    <=> Changing goals at 144 from 9 to 0 <MONITOR-CURRENT-LOCATION>
```

The truck has just arrived at anew location so must scan its surroundings.

```
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 144
    --- Abandoning side task at 148 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 148 from 0 to 9
        <DELIVER-OBJECT SAW ILS ORDER-1>
```

(SCREWDRIVER OBJ-1 disappeared at time 181)

```
            *** Arrived at TOOL-DEPOT-2 at time 181
--> New order ORDER-4: some SCREWS for ILS received at time 181
    There are now 4 orders outstanding
--> New top level goal: DELIVER-OBJECT ::  [13:13]
    <=> Changing goals at 181 from 9 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 181
```

The truck has now arrived at TOOL-DEPOT-2 and scans its surroundings. One of the objects present is a saw.

```
    --- Abandoning side task at 185 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 185 from 0 to 9
        <DELIVER-OBJECT SAW ILS ORDER-1>
        ... Processing task: <FIND-OBJECT SAW => ?PLACE-NAME ?OBJECT>
             ::  [9:5] at 185
        ... Task succeeded: <FIND-OBJECT SAW => TOOL-DEPOT-2 ITEM-24>
             ::  [9:5] at 185
```

There is an object here that meets the saw *goal-criterion.*

C.2.2 Loading and delivering

The next step is to load the saw into the truck. After that, the truck must travel to the location of the delivery site. Finally, the saw is placed inside the site.

```
   <=> Changing goals at 185 from 24 to 9
        <DELIVER-OBJECT SAW ILS ORDER-1>
        ... Processing task:
             <LOAD-PAYLOAD-OBJECT TOOL-DEPOT-2 ITEM-24 SAW> ::  [9:6]
             at 185
    <=> Changing goals at 201 from 9 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 201
```

(PAINTBRUSH OBJ-25 appeared at time 205)

```
    --- Abandoning side task at 205 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 205 from 0 to 9
        <DELIVER-OBJECT SAW ILS ORDER-1>
        ... Processing task:
             <LOAD-PAYLOAD-OBJECT TOOL-DEPOT-2 ITEM-24 SAW> ::  [9:6]
             at 210
        ... Task succeeded:
             <LOAD-PAYLOAD-OBJECT TOOL-DEPOT-2 ITEM-24 SAW> ::  [9:6]
             at 210
```

The truck has loaded the saw into its cargo bay, and must next take it to the delivery site.

```
        ... Processing task: <TAKE-OBJECT-TO-SITE ITEM-24 ILS>
             ::  [9:7] at 210
        ... Processing task: <TRUCK-TRAVEL-TO MAPLE-AVE>
             ::  [9:6] at 210
```

```
    <DELIVER-OBJECT WOOD TECH ORDER-8>
        --> Delaying DELIVER-OBJECT [25:25] at 404
            for 50 time units to avoid SMEAR interaction
--- Abandoning side task at 404 :  DELIVER-OBJECT  [25:25]
```

The wood's paint was wet so PARETO gives it time to dry and returns to the wallpaper delivery.

C.3.2 A method fails

The truck has arrived at the paintshop, which PARETO believes is a good source for wallpaper, but there is none there. PARETO has seen no wallpaper anywhere else, so it must just wander around the world for a period before returning to the paintshop.

```
<=> Changing goals at 404 from 25 to 24
    <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
    ... Processing task: <TRUCK-TRAVEL-TO PAINTSHOP>
         ::  [24:9] at 404
    ... Task succeeded: <TRUCK-TRAVEL-TO PAINTSHOP>
         ::  [24:9] at 404
    ... Processing task:
         <FIND-OBJECT WALLPAPER => ?PLACE-NAME ?OBJECT>
         ::  [24:5] at 404
    ... Processing task: <WANDER-RANDOMLY 404 120>
         ::  [24:9] at 404
```

The truck will wander round the world for 120 time units before returning to the paintshop.

C.3.3 Simultaneous opportunities for several goals

While wandering around looking for some wallpaper, the truck visits warehouse-3 and PARETO finds several other objects that it needs.

```
            *** Arrived at WAREHOUSE-3 at time 436
(HAMMER OBJ-21 appeared at time 436)
    <=> Changing goals at 436 from 24 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 436
    --- Abandoning side task at 440 :  MONITOR-CURRENT-LOCATION  [0:0]
    --> New side task chosen at 440 :
        <DELIVER-OBJECT TAPE HOUSE ORDER-3> [15:15]
```

The truck spotted some tape when it scanned its surroundings, and PARETO recognizes an opportunity.

```
    <=> Changing goals at 440 from 0 to 15
        <DELIVER-OBJECT TAPE HOUSE ORDER-3>
        ... Processing task: <DELIVER-OBJECT TAPE HOUSE ORDER-3>
             ::  [15:15] at 440
        ... Processing task: <FIND-OBJECT TAPE => ?PLACE-NAME ?OBJECT>
             ::  [15:14] at 440
        ... Task succeeded: <FIND-OBJECT TAPE => WAREHOUSE-3 ITEM-67>
             ::  [15:14] at 440
        ... Processing task:
             <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-67 TAPE>
             ::  [15:16] at 440
    <=> Changing goals at 456 from 15 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
```

```
            *** Arrived at BANK at time 245
--> New order ORDER-5: a pair of SCISSORS for HOUSE received
    at time 245
    There are now 5 orders outstanding
--> New top level goal: DELIVER-OBJECT ::  [20:20]
    <=> Changing goals at 245 from 9 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 245
    --- Abandoning side task at 249 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 249 from 0 to 9
        <DELIVER-OBJECT SAW ILS ORDER-1>
--> New order ORDER-6: a KNIFE for TECH received at time 254
    There are now 6 orders outstanding
--> New top level goal: DELIVER-OBJECT ::  [29:29]
            *** Arrived at BASE at time 269
--> New order ORDER-7: a ball of TWINE for ILS received at time 269
    There are now 7 orders outstanding
--> New top level goal: DELIVER-OBJECT ::  [17:17]
    --- Abandoning side task at 269 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 269 from 9 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 269
    --- Abandoning side task at 273 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 273 from 0 to 9
        <DELIVER-OBJECT SAW ILS ORDER-1>
(FUEL-DRUM OBJ-1 disappeared at time 295)
            *** Arrived at MAPLE-AVE at time 295
    <=> Changing goals at 295 from 9 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 295
--> New order ORDER-8: a plank of WOOD for TECH received at time 299
    There are now 8 orders outstanding
--> New top level goal: DELIVER-OBJECT ::  [25:25]
    --- Abandoning side task at 299 :  MONITOR-CURRENT-LOCATION  [0:0]
        ... Processing task: <TRUCK-TRAVEL-TO MAPLE-AVE>
             ::  [9:6] at 299
        ... Task succeeded: <TRUCK-TRAVEL-TO MAPLE-AVE>
             ::  [9:6] at 299
        ... Processing task: <TAKE-OBJECT-TO-SITE ITEM-24 ILS>
             ::  [9:7] at 299
        ... Task succeeded: <TAKE-OBJECT-TO-SITE ITEM-24 ILS>
             ::  [9:7] at 299
```

The truck has now arrived at the location where the delivery is to be made. The next task is to unload the saw into the delivery site.

```
        ... Processing task: <UNLOAD-AT ILS ITEM-24> ::  [9:8] at 299
    <=> Changing goals at 315 from 9 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 315
    --- Abandoning side task at 319 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 319 from 0 to 9
        <DELIVER-OBJECT SAW ILS ORDER-1>
*** ORDER-1 filled at time 330
        ... Processing task: <UNLOAD-AT ILS ITEM-24> ::  [9:8] at 330
        ... Task succeeded: <UNLOAD-AT ILS ITEM-24> ::  [9:8] at 330
        ... Processing task: <DELIVER-OBJECT SAW ILS ORDER-1>
             ::  [9:9] at 330
    ... Task succeeded: <DELIVER-OBJECT SAW ILS ORDER-1>
         ::  [9:9] at 330
--> Order ORDER-1 for a SAW at ILS
    filled at time 330 - 7 orders remain to be filled
```

C.3 A difficult delivery

The task to deliver a saw has been achieved, and PARETO must choose a new task to address. It chooses to analyze a potential interaction for the goal to deliver some `wallpaper` to `tech`.

```
=== New main task chosen at 330 :
    <ANALYZE-POTENTIAL-INTERACTION RIP> [24:5]
*** Addressing goal 24 <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
    at 330
```

The interaction turns out not to be genuine, so PARETO continues to address the goal.

```
    ... Processing task: <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
         ::  [24:24] at 330
    ... Processing task:
         <FIND-OBJECT WALLPAPER => ?PLACE-NAME ?OBJECT>
         ::  [24:5] at 330
    ... Processing task: <TRUCK-TRAVEL-TO PAINTSHOP>
         ::  [24:9] at 330
<=> Changing goals at 335 from 24 to 0 <MONITOR-CURRENT-LOCATION>
    ... Processing task: <MONITOR-CURRENT-LOCATION>
         ::  [0:0] at 335
```

(WOOD OBJ-35 appeared at time 339)

```
--- Abandoning side task at 339 :  MONITOR-CURRENT-LOCATION  [0:0]
<=> Changing goals at 339 from 0 to 24
    <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
```

(HAMMER OBJ-10 disappeared at time 364)

(LADDER OBJ-19 appeared at time 364)

```
        *** Arrived at SHERIDAN-RD at time 364
<=> Changing goals at 364 from 24 to 0 <MONITOR-CURRENT-LOCATION>
    ... Processing task: <MONITOR-CURRENT-LOCATION>
         ::  [0:0] at 364
--- Abandoning side task at 368 :  MONITOR-CURRENT-LOCATION  [0:0]
<=> Changing goals at 368 from 0 to 24
    <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
        *** Arrived at PAINTSHOP at time 397
<=> Changing goals at 397 from 24 to 0 <MONITOR-CURRENT-LOCATION>
    ... Processing task: <MONITOR-CURRENT-LOCATION>
         ::  [0:0] at 397
```

(SCREWS OBJ-32 appeared at time 401)

```
--- Abandoning side task at 401 :  MONITOR-CURRENT-LOCATION  [0:0]
```

C.3.1 A genuine interaction

```
--> New side task chosen at 401 :
    <ANALYZE-POTENTIAL-INTERACTION SMEAR> [25:28]
```

There is a potential opportunity to pick up some wood, but first PARETO must decide whether it will smear.

```
<=> Changing goals at 401 from 0 to 25
```

```
        <DELIVER-OBJECT WOOD TECH ORDER-8>
            --> Delaying DELIVER-OBJECT [25:25] at 404
                for 50 time units to avoid SMEAR interaction
    --- Abandoning side task at 404 :  DELIVER-OBJECT  [25:25]
```

The wood's paint was wet so PARETO gives it time to dry and returns to the wallpaper delivery.

C.3.2 A method fails

The truck has arrived at the paintshop, which PARETO believes is a good source for wallpaper, but there is none there. PARETO has seen no wallpaper anywhere else, so it must just wander around the world for a period before returning to the paintshop.

```
    <=> Changing goals at 404 from 25 to 24
        <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
        ... Processing task: <TRUCK-TRAVEL-TO PAINTSHOP>
             ::  [24:9] at 404
        ... Task succeeded: <TRUCK-TRAVEL-TO PAINTSHOP>
             ::  [24:9] at 404
        ... Processing task:
             <FIND-OBJECT WALLPAPER => ?PLACE-NAME ?OBJECT>
             ::  [24:5] at 404
        ... Processing task: <WANDER-RANDOMLY 404 120>
             ::  [24:9] at 404
```

The truck will wander round the world for 120 time units before returning to the paintshop.

C.3.3 Simultaneous opportunities for several goals

While wandering around looking for some wallpaper, the truck visits warehouse-3 and PARETO finds several other objects that it needs.

```
            *** Arrived at WAREHOUSE-3 at time 436
(HAMMER OBJ-21 appeared at time 436)
    <=> Changing goals at 436 from 24 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 436
    --- Abandoning side task at 440 :  MONITOR-CURRENT-LOCATION  [0:0]
    --> New side task chosen at 440 :
        <DELIVER-OBJECT TAPE HOUSE ORDER-3> [15:15]
```

The truck spotted some tape when it scanned its surroundings, and PARETO recognizes an opportunity.

```
    <=> Changing goals at 440 from 0 to 15
        <DELIVER-OBJECT TAPE HOUSE ORDER-3>
        ... Processing task: <DELIVER-OBJECT TAPE HOUSE ORDER-3>
             ::  [15:15] at 440
        ... Processing task: <FIND-OBJECT TAPE => ?PLACE-NAME ?OBJECT>
             ::  [15:14] at 440
        ... Task succeeded: <FIND-OBJECT TAPE => WAREHOUSE-3 ITEM-67>
             ::  [15:14] at 440
        ... Processing task:
             <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-67 TAPE>
             ::  [15:16] at 440
    <=> Changing goals at 456 from 15 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
```

```
             ::  [0:0] at 456
    --- Abandoning side task at 460 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 460 from 0 to 15
        <DELIVER-OBJECT TAPE HOUSE ORDER-3>
        ... Processing task:
             <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-67 TAPE>
             ::  [15:16] at 465
        ... Task succeeded:
             <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-67 TAPE>
             ::  [15:16] at 465
```

The truck has loaded the tape into its cargo bay. The next stage would be to go off and deliver it, but this would interfere with the main task of finding wallpaper and would lose other opportunities.

```
    --- Abandoning side task at 465 :  TAKE-OBJECT-TO-SITE  [15:18]
    --- Abandoning side task at 465 :  DELIVER-OBJECT  [15:15]
    --> New side task chosen at 465 :
        <DELIVER-OBJECT SCREWS ILS ORDER-4> [13:13]
```

PARETO has recognized another opportunity.

```
    <=> Changing goals at 465 from 15 to 13
        <DELIVER-OBJECT SCREWS ILS ORDER-4>
        ... Processing task: <DELIVER-OBJECT SCREWS ILS ORDER-4>
             ::  [13:13] at 465
        ... Processing task:
             <FIND-OBJECT SCREWS => ?PLACE-NAME ?OBJECT>
             ::  [13:14] at 465
        ... Task succeeded:
             <FIND-OBJECT SCREWS => WAREHOUSE-3 ITEM-65>
             ::  [13:14] at 465
        ... Processing task:
             <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-65 SCREWS>
             ::  [13:16] at 465
    <=> Changing goals at 476 from 13 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 476
    --- Abandoning side task at 480 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 480 from 0 to 13
        <DELIVER-OBJECT SCREWS ILS ORDER-4>
        ... Processing task:
             <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-65 SCREWS>
             ::  [13:16] at 482
        ... Task succeeded:
             <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-65 SCREWS>
             ::  [13:16] at 482
    --- Abandoning side task at 482 :  TAKE-OBJECT-TO-SITE  [13:30]
    --- Abandoning side task at 482 :  DELIVER-OBJECT  [13:13]
```

There is another potential opportunity, but it may interact problematically.

```
    --> New side task chosen at 482 :
        <ANALYZE-POTENTIAL-INTERACTION CUT> [20:21]
    <=> Changing goals at 482 from 13 to 20
        <DELIVER-OBJECT SCISSORS HOUSE ORDER-5>
```

In fact, there is no problem.

```
        ... Processing task: <DELIVER-OBJECT SCISSORS HOUSE ORDER-5>
             ::  [20:20] at 482
        ... Processing task:
             <FIND-OBJECT SCISSORS => ?PLACE-NAME ?OBJECT>
             ::  [20:14] at 482
```

```
  ... Task succeeded:
       <FIND-OBJECT SCISSORS => WAREHOUSE-3 ITEM-60>
       ::  [20:14] at 482
  ... Processing task:
       <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-60 SCISSORS>
       ::  [20:16] at 482
```

(TWINE OBJ-7 appeared at time 490)

```
  ... Processing task:
       <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-60 SCISSORS>
       ::  [20:16] at 495
  ... Task succeeded:
       <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-60 SCISSORS>
       ::  [20:16] at 495
--- Abandoning side task at 495 :  TAKE-OBJECT-TO-SITE  [20:21]
--- Abandoning side task at 495 :  DELIVER-OBJECT  [20:20]
```

PARETO has now taken advantage of opportunities to pick up three objects, and recognizes the possibility of another.

```
--> New side task chosen at 495 :
  <ANALYZE-POTENTIAL-INTERACTION CUT> [29:22]
<=> Changing goals at 495 from 20 to 29
  <DELIVER-OBJECT KNIFE TECH ORDER-6>
  ... Processing task: <DELIVER-OBJECT KNIFE TECH ORDER-6>
       ::  [29:29] at 495
  ... Processing task:
       <FIND-OBJECT KNIFE => ?PLACE-NAME ?OBJECT>
       ::  [29:14] at 495
--- Abandoning side task at 495 :  TRUCK-TRAVEL-TO  [29:34]
--- Abandoning side task at 495 :  FIND-OBJECT  [29:14]
--- Abandoning side task at 495 :  DELIVER-OBJECT  [29:29]
```

In fact, there was no knife present: the reference feature sharp was misleading. However, there is another potential opportunity.

```
--> New side task chosen at 495 :
  <ANALYZE-POTENTIAL-INTERACTION CUT> [17:23]
<=> Changing goals at 495 from 29 to 17
  <DELIVER-OBJECT TWINE ILS ORDER-7>
  ... Processing task: <DELIVER-OBJECT TWINE ILS ORDER-7>
       ::  [17:17] at 495
  ... Processing task:
       <FIND-OBJECT TWINE => ?PLACE-NAME ?OBJECT>
       ::  [17:23] at 495
  ... Task succeeded: <FIND-OBJECT TWINE => WAREHOUSE-3 ITEM-66>
       ::  [17:23] at 495
  ... Processing task:
       <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-66 TWINE>
       ::  [17:26] at 495
<=> Changing goals at 498 from 17 to 0 <MONITOR-CURRENT-LOCATION>
  ... Processing task: <MONITOR-CURRENT-LOCATION>
       ::  [0:0] at 498
--- Abandoning side task at 502 :  MONITOR-CURRENT-LOCATION  [0:0]
<=> Changing goals at 502 from 0 to 17
  <DELIVER-OBJECT TWINE ILS ORDER-7>
```

(SCISSORS OBJ-10 appeared at time 505)

(BUCKET OBJ-32 appeared at time 510)

```
  ... Processing task:
       <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-66 TWINE>
       ::  [17:26] at 512
```

```
        ... Task succeeded:
             <LOAD-PAYLOAD-OBJECT WAREHOUSE-3 ITEM-66 TWINE>
             ::  [17:26] at 512
    --- Abandoning side task at 512 :  TAKE-OBJECT-TO-SITE  [17:27]
    --- Abandoning side task at 512 :  DELIVER-OBJECT  [17:17]
```

Another successful opportunity.

C.3.4 An interaction that can be prevented

There are no more potential opportunities at the current location, so PARETO returns to its main task of looking for some wallpaper. It is not yet time to return to the paintshop, so the truck must continue to wander around the world. On the way it finds a knife, which it requires for one of its other delivery goals, so it takes advantage of the opportunity, first retracting the blade to avoid a problematic interaction.

```
    <=> Changing goals at 512 from 17 to 24
        <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
        ... Processing task: <WANDER-RANDOMLY 404 120>
             ::  [24:9] at 512
(SCISSORS OBJ-10 disappeared at time 514)
    <=> Changing goals at 518 from 24 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 518
    --- Abandoning side task at 522 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 522 from 0 to 24
        <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
            *** Arrived at TOOL-DEPOT-1 at time 541
    <=> Changing goals at 541 from 24 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 541
    --- Abandoning side task at 545 :  MONITOR-CURRENT-LOCATION  [0:0]
    --> New side task chosen at 545 :
        <TRUCK-TRAVEL-TO TOOL-DEPOT-2> [29:34]
        ... Processing task: <TRUCK-TRAVEL-TO TOOL-DEPOT-2>
             ::  [29:34] at 545
            *** Arrived at LUMBERYARD at time 555
    <=> Changing goals at 555 from 29 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 555
    --- Abandoning side task at 559 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 559 from 0 to 29
        <DELIVER-OBJECT KNIFE TECH ORDER-6>
(KNIFE OBJ-20 appeared at time 577)
(FUEL-DRUM OBJ-17 disappeared at time 577)
(SCISSORS OBJ-16 disappeared at time 577)
            *** Arrived at TOOL-DEPOT-2 at time 577
    <=> Changing goals at 577 from 29 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 577
(SCISSORS OBJ-32 appeared at time 581)
    --- Abandoning side task at 581 :  MONITOR-CURRENT-LOCATION  [0:0]
        ... Processing task:
             <TRUCK-TRAVEL-TO TOOL-DEPOT-2> ::  [29:34] at 581
        ... Task succeeded: <TRUCK-TRAVEL-TO TOOL-DEPOT-2>
```

```
         ::  [29:34] at 581
    --- Retracting knife blade to avoid CUT interaction at 591
```

The knife's blade must be retracted so that it does not cut the twine already in the truck's cargo bay.

```
    ... Processing task:
         <FIND-OBJECT KNIFE => ?PLACE-NAME ?OBJECT>
         ::  [29:14] at 596
    ... Task succeeded:
         <FIND-OBJECT KNIFE => TOOL-DEPOT-2 ITEM-109>
         ::  [29:14] at 596
    ... Processing task:
         <LOAD-PAYLOAD-OBJECT TOOL-DEPOT-2 ITEM-109 KNIFE>
         ::  [29:16] at 596
```

(SOLVENT OBJ-24 appeared at time 600)

```
<=> Changing goals at 600 from 29 to 0 <MONITOR-CURRENT-LOCATION>
    ... Processing task: <MONITOR-CURRENT-LOCATION>
         ::  [0:0] at 600
--- Abandoning side task at 604 :  MONITOR-CURRENT-LOCATION  [0:0]
<=> Changing goals at 604 from 0 to 29
    <DELIVER-OBJECT KNIFE TECH ORDER-6>
```

(FUEL-DRUM OBJ-9 appeared at time 615)

```
    ... Processing task:
         <LOAD-PAYLOAD-OBJECT TOOL-DEPOT-2 ITEM-109 KNIFE>
         ::  [29:16] at 615
    ... Task succeeded:
         <LOAD-PAYLOAD-OBJECT TOOL-DEPOT-2 ITEM-109 KNIFE>
         ::  [29:16] at 615
--- Abandoning side task at 615 :  TAKE-OBJECT-TO-SITE  [29:22]
--- Abandoning side task at 615 :  DELIVER-OBJECT  [29:29]
```

That's all on this opportunity: PARETO now returns to its main task.

C.3.5 An unsuccessful return to the main task

PARETO now returns to the task of finding some wallpaper: it is time to return to the paintshop.

```
<=> Changing goals at 615 from 29 to 24
    <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
    ... Processing task: <WANDER-RANDOMLY 404 120>
         ::  [24:9] at 615
    ... Task succeeded: <WANDER-RANDOMLY 404 120>
         ::  [24:9] at 615
    ... Processing task: <TRUCK-TRAVEL-TO PAINTSHOP>
         ::  [24:10] at 615
<=> Changing goals at 620 from 24 to 0 <MONITOR-CURRENT-LOCATION>
    ... Processing task: <MONITOR-CURRENT-LOCATION>
         ::  [0:0] at 620
--- Abandoning side task at 624 :  MONITOR-CURRENT-LOCATION  [0:0]
<=> Changing goals at 624 from 0 to 24
    <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
        *** Arrived at BANK at time 652
<=> Changing goals at 652 from 24 to 0 <MONITOR-CURRENT-LOCATION>
    ... Processing task: <MONITOR-CURRENT-LOCATION>
        ::  [0:0] at 652
--- Abandoning side task at 656 :  MONITOR-CURRENT-LOCATION  [0:0]
<=> Changing goals at 656 from 0 to 24
    <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
```

(TWINE OBJ-2 disappeared at time 687)

```
        *** Arrived at WAREHOUSE-1 at time 687
<=> Changing goals at 687 from 24 to 0
   <MONITOR-CURRENT-LOCATION>
    ... Processing task: <MONITOR-CURRENT-LOCATION>
         ::  [0:0] at 687
```

(PAINT OBJ-5 appeared at time 691)

```
--- Abandoning side task at 691 :  MONITOR-CURRENT-LOCATION   [0:0]
<=> Changing goals at 691 from 0 to 24
    <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
        *** Arrived at PAINTSHOP at time 708
<=> Changing goals at 708 from 24 to 0 <MONITOR-CURRENT-LOCATION>
    ... Processing task: <MONITOR-CURRENT-LOCATION>
         ::  [0:0] at 708
--- Abandoning side task at 712 :  MONITOR-CURRENT-LOCATION   [0:0]
```

Once again there is no wallpaper here, but there is some wood.

```
--> New side task chosen at 712 :
   <ANALYZE-POTENTIAL-INTERACTION SMEAR> [25:40]
```

PARETO checks the paint on the wood and finds that it is dry.

```
<=> Changing goals at 712 from 0 to 25
    <DELIVER-OBJECT WOOD TECH ORDER-8>
    ... Processing task: <DELIVER-OBJECT WOOD TECH ORDER-8>
         ::  [25:25] at 718
    ... Processing task: <FIND-OBJECT WOOD => ?PLACE-NAME ?OBJECT>
         ::  [25:11] at 718
    ... Task succeeded: <FIND-OBJECT WOOD => PAINTSHOP ITEM-47>
         ::  [25:11] at 718
    ... Processing task:
         <LOAD-PAYLOAD-OBJECT PAINTSHOP ITEM-47 WOOD>
         ::  [25:12] at 718
```

(PAINT OBJ-1 disappeared at time 722)

(NAILS OBJ-29 appeared at time 725)

```
<=> Changing goals at 728 from 25 to 0 <MONITOR-CURRENT-LOCATION>
    ... Processing task: <MONITOR-CURRENT-LOCATION>
         ::  [0:0] at 728
--- Abandoning side task at 732 :  MONITOR-CURRENT-LOCATION   [0:0]
<=> Changing goals at 732 from 0 to 25
    <DELIVER-OBJECT WOOD TECH ORDER-8>
    ... Processing task:
         <LOAD-PAYLOAD-OBJECT PAINTSHOP ITEM-47 WOOD>
         ::  [25:12] at 743
    ... Task succeeded:
         <LOAD-PAYLOAD-OBJECT PAINTSHOP ITEM-47 WOOD>
         ::  [25:12] at 743
--- Abandoning side task at 743 :  TAKE-OBJECT-TO-SITE  [25:16]
--- Abandoning side task at 743 :  DELIVER-OBJECT  [25:25]
```

PARETO has taken advantage of the opportunity and returns to the job of finding some wallpaper.

```
<=> Changing goals at 743 from 25 to 24
    <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
    ... Processing task: <TRUCK-TRAVEL-TO PAINTSHOP>
         ::  [24:10] at 743
    ... Task succeeded: <TRUCK-TRAVEL-TO PAINTSHOP>
         ::  [24:10] at 743
    ... Processing task:
```

```
             <FIND-OBJECT WALLPAPER => ?PLACE-NAME ?OBJECT>
             ::  [24:5] at 743
```

There is none present and PARETO doesn't know of any anywhere else, so must wander around again.

```
        ... Processing task: <WANDER-RANDOMLY 743 120>
             ::  [24:9] at 743
    <=> Changing goals at 748 from 24 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 748
(BOX OBJ-33 appeared at time 752)
    --- Abandoning side task at 752 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 752 from 0 to 24
        <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
```

C.3.6 Some delivery opportunities

The truck now arrives at sheridan-rd where some deliveries must be made. It takes advantage of the opportunity to deliver some of the objects it picked up earlier. Next, its wanderings take it to newport-ave where there are more opportunities.

```
            *** Arrived at SHERIDAN-RD at time 777
    <=> Changing goals at 777 from 24 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 777
(BAG OBJ-3 appeared at time 781)
    --- Abandoning side task at 781 :  MONITOR-CURRENT-LOCATION  [0:0]
    --> New side task chosen at 781 :
        <ANALYZE-POTENTIAL-INTERACTION MOVE> [29:14]
        ... Processing task: <TAKE-OBJECT-TO-SITE ITEM-109 TECH>
             ::  [29:22] at 781
        ... Task succeeded: <TAKE-OBJECT-TO-SITE ITEM-109 TECH>
             ::  [29:22] at 781
        ... Processing task: <UNLOAD-AT TECH ITEM-109>
             ::  [29:33] at 781
    <=> Changing goals at 797 from 29 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 797
    --- Abandoning side task at 801 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 801 from 0 to 29
        <DELIVER-OBJECT KNIFE TECH ORDER-6>
(LADDER OBJ-32 disappeared at time 810)
*** ORDER-6 filled at time 812
        ... Processing task: <UNLOAD-AT TECH ITEM-109>
             ::  [29:33] at 812
        ... Task succeeded: <UNLOAD-AT TECH ITEM-109>
             ::  [29:33] at 812
        ... Processing task: <DELIVER-OBJECT KNIFE TECH ORDER-6>
             ::  [29:29] at 812
        ... Task succeeded: <DELIVER-OBJECT KNIFE TECH ORDER-6>
             ::  [29:29] at 812
--> Order ORDER-6 for a KNIFE at TECH
    filled at time 812 - 6 orders remain to be filled
```

This delivery goal has been achieved entirely through the use of opportunities: it has never been PARETO's main task.

```
    --> New side task chosen at 812 :
       <ANALYZE-POTENTIAL-INTERACTION MOVE> [25:26]
    *** Addressing goal 25 <DELIVER-OBJECT WOOD TECH ORDER-8> at 812
        ... Processing task: <TAKE-OBJECT-TO-SITE ITEM-47 TECH>
             ::  [25:16] at 812
        ... Task succeeded: <TAKE-OBJECT-TO-SITE ITEM-47 TECH>
             ::  [25:16] at 812
        ... Processing task: <UNLOAD-AT TECH ITEM-47>
             ::  [25:23] at 812
    <=> Changing goals at 818 from 25 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 818
    --- Abandoning side task at 822 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 822 from 0 to 25
        <DELIVER-OBJECT WOOD TECH ORDER-8>
    <=> Changing goals at 840 from 25 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 840
```

(SCISSORS OBJ-15 appeared at time 844)

```
    --- Abandoning side task at 844 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 844 from 0 to 25
        <DELIVER-OBJECT WOOD TECH ORDER-8>
```

***** ORDER-8 filled at time 849**

```
        ... Processing task: <UNLOAD-AT TECH ITEM-47>
             ::  [25:23] at 849
        ... Task succeeded: <UNLOAD-AT TECH ITEM-47>
             ::  [25:23] at 849
        ... Processing task: <DELIVER-OBJECT WOOD TECH ORDER-8>
             ::  [25:25] at 849
        ... Task succeeded: <DELIVER-OBJECT WOOD TECH ORDER-8>
             ::  [25:25] at 849
--> Order ORDER-8 for a plank of WOOD at TECH
    filled at time 849 - 5 orders remain to be filled
```

Another goal achieved entirely through opportunities.

```
    *** Addressing goal 24 <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
        at 849
        ... Processing task: <WANDER-RANDOMLY 743 120>
             ::  [24:9] at 849
```

It's still not time to return to the paintshop.

```
            *** Arrived at NEWPORT-AVE at time 877
    <=> Changing goals at 877 from 24 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 877
    --- Abandoning side task at 881 :  MONITOR-CURRENT-LOCATION  [0:0]
    --> New side task chosen at 881 :
        <ANALYZE-POTENTIAL-INTERACTION MOVE> [15:14]
        ... Processing task: <TAKE-OBJECT-TO-SITE ITEM-67 HOUSE>
             ::  [15:18] at 881
        ... Task succeeded: <TAKE-OBJECT-TO-SITE ITEM-67 HOUSE>
             ::  [15:18] at 881
```

There's an opportunity to deliver the tape that was picked up earlier.

```
        ... Processing task: <UNLOAD-AT HOUSE ITEM-67>
             ::  [15:19] at 881
```

(LADDER OBJ-35 appeared at time 891)

```
    <=> Changing goals at 897 from 15 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 897
    --- Abandoning side task at 901 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 901 from 0 to 15
        <DELIVER-OBJECT TAPE HOUSE ORDER-3>
```

***** ORDER-3 filled at time 912**

```
        ... Processing task: <UNLOAD-AT HOUSE ITEM-67>
             ::  [15:19] at 912
        ... Task succeeded: <UNLOAD-AT HOUSE ITEM-67>
             ::  [15:19] at 912
        ... Processing task: <DELIVER-OBJECT TAPE HOUSE ORDER-3>
             ::  [15:15] at 912
        ... Task succeeded: <DELIVER-OBJECT TAPE HOUSE ORDER-3>
             ::  [15:15] at 912
--> Order ORDER-3 for a roll of TAPE at HOUSE
    filled at time 912 - 4 orders remain to be filled
```

Again, this goal has never been PARETO's main task.

```
    --> New side task chosen at 912 :
        <ANALYZE-POTENTIAL-INTERACTION MOVE> [20:16]
```

There is another delivery that can be made here.

```
    *** Addressing goal 20 <DELIVER-OBJECT SCISSORS HOUSE ORDER-5>
        at 912
        ... Processing task: <TAKE-OBJECT-TO-SITE ITEM-60 HOUSE>
             ::  [20:21] at 912
        ... Task succeeded: <TAKE-OBJECT-TO-SITE ITEM-60 HOUSE>
             ::  [20:21] at 912
        ... Processing task: <UNLOAD-AT HOUSE ITEM-60>
             ::  [20:32] at 912
    <=> Changing goals at 918 from 20 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 918
    --- Abandoning side task at 922 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 922 from 0 to 20
        <DELIVER-OBJECT SCISSORS HOUSE ORDER-5>
```

(LADDER OBJ-20 disappeared at time 936)

***** ORDER-5 filled at time 938**

```
    <=> Changing goals at 938 from 20 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 938
    --- Abandoning side task at 942 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 942 from 0 to 20
        <DELIVER-OBJECT SCISSORS HOUSE ORDER-5>
        ... Processing task: <UNLOAD-AT HOUSE ITEM-60>
             ::  [20:32] at 942
        ... Task succeeded: <UNLOAD-AT HOUSE ITEM-60>
             ::  [20:32] at 942
        ... Processing task: <DELIVER-OBJECT SCISSORS HOUSE ORDER-5>
             ::  [20:20] at 942
        ... Task succeeded: <DELIVER-OBJECT SCISSORS HOUSE ORDER-5>
             ::  [20:20] at 942
--> Order ORDER-5 for a pair of SCISSORS at HOUSE
    filled at time 942 - 3 orders remain to be filled
```

The fourth delivery goal to be achieved entirely through opportunities.

C.3.7 Opportunities on the way to refuel

It is now time for the truck to return to the paintshop to look for some wallpaper, but it discovers it is low on fuel. PARETO decides to go to the base to refuel. On the way there are more opportunities to make deliveries.

```
   *** Addressing goal 24 <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
       at 942
        ... Processing task: <WANDER-RANDOMLY 743 120>
             ::  [24:9] at 942
        ... Task succeeded: <WANDER-RANDOMLY 743 120>
             ::  [24:9] at 942
        ... Processing task: <TRUCK-TRAVEL-TO PAINTSHOP>
             ::  [24:10] at 942
```

(BAG OBJ-7 disappeared at time 970)

```
            *** Arrived at SHERIDAN-RD at time 970
    >>> Main task:  [24:19] supplanted by  [1:2]
        <WATCH-FOR-LOW-FUEL [SET: (ITEM-5 ITEM-7)]>
```

Refueling is all-important, so supplants all delivery goals.

```
    <=> Changing goals at 970 from 24 to 0 <MONITOR-CURRENT-LOCATION>
```

However, the truck must continue to scan its surroundings regularly.

```
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 970
```

(SCISSORS OBJ-24 disappeared at time 974)

```
    --- Abandoning side task at 974 :  MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 974 from 0 to 1 <MONITOR-FUEL-LEVEL>
        ... Processing task:
             <WATCH-FOR-LOW-FUEL [SET: (ITEM-5 ITEM-7)]>
             ::  [1:2] at 974
        ... Processing task: <LOOK-FOR-FUEL> ::  [1:14] at 974
        ... Processing task: <TRUCK-TRAVEL-TO BASE> ::  [1:18] at 974
            *** Arrived at MAPLE-AVE at time 1001
```

(LADDER OBJ-10 appeared at time 1001)

```
    <=> Changing goals at 1001 from 1 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 1001
    --- Abandoning side task at 1005 :
        MONITOR-CURRENT-LOCATION  [0:0]
    --> New side task chosen at 1005 :
       <ANALYZE-POTENTIAL-INTERACTION MOVE> [13:23]
```

There is a delivery that can be made here; as unloading does not use fuel, PARETO takes advantage of the opportunity.

```
        ... Processing task: <TAKE-OBJECT-TO-SITE ITEM-65 ILS>
             ::  [13:30] at 1005
        ... Task succeeded: <TAKE-OBJECT-TO-SITE ITEM-65 ILS>
             ::  [13:30] at 1005
        ... Processing task: <UNLOAD-AT ILS ITEM-65> :: [13:31] at 1005
    <=> Changing goals at 1021 from 13 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 1021
    --- Abandoning side task at 1025 : MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 1025 from 0 to 13
        <DELIVER-OBJECT SCREWS ILS ORDER-4>
```

```
(FUEL-DRUM OBJ-12 appeared at time 1034)
*** ORDER-4 filled at time 1036
        ... Processing task: <UNLOAD-AT ILS ITEM-65>
             ::  [13:31] at 1036
        ... Task succeeded: <UNLOAD-AT ILS ITEM-65>
             ::  [13:31] at 1036
        ... Processing task: <DELIVER-OBJECT SCREWS ILS ORDER-4>
             ::  [13:13] at 1036
        ... Task succeeded: <DELIVER-OBJECT SCREWS ILS ORDER-4>
             ::  [13:13] at 1036
--> Order ORDER-4 for some SCREWS at ILS
    filled at time 1036 - 2 orders remain to be filled
```

The fifth goal to be accomplished entirely through opportunities.

```
    --> New side task chosen at 1036 :
        <ANALYZE-POTENTIAL-INTERACTION MOVE> [17:25]
```

There is another delivery that can be made

```
    *** Addressing goal 17 <DELIVER-OBJECT TWINE ILS ORDER-7> at 1036
        ... Processing task: <TAKE-OBJECT-TO-SITE ITEM-66 ILS>
             ::  [17:27] at 1036
        ... Task succeeded: <TAKE-OBJECT-TO-SITE ITEM-66 ILS>
             ::  [17:27] at 1036
        ... Processing task: <UNLOAD-AT ILS ITEM-66> :: [17:36] at 1036
    <=> Changing goals at 1042 from 17 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 1042
    --- Abandoning side task at 1046 : MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 1046 from 0 to 17
        <DELIVER-OBJECT TWINE ILS ORDER-7>
(SCISSORS OBJ-2 appeared at time 1054)
(WOOD OBJ-17 disappeared at time 1054)
*** ORDER-7 filled at time 1062
    <=> Changing goals at 1062 from 17 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 1062
    --- Abandoning side task at 1066 : MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 1066 from 0 to 17
        <DELIVER-OBJECT TWINE ILS ORDER-7>
        ... Processing task: <UNLOAD-AT ILS ITEM-66>
             ::  [17:36] at 1066
        ... Task succeeded: <UNLOAD-AT ILS ITEM-66> :: [17:36] at 1066
        ... Processing task: <DELIVER-OBJECT TWINE ILS ORDER-7>
             ::  [17:17] at 1066
        ... Task succeeded: <DELIVER-OBJECT TWINE ILS ORDER-7>
             ::  [17:17] at 1066
--> Order ORDER-7 for a ball of TWINE at ILS
    filled at time 1066 - 1 orders remain to be filled
```

Six out of the seven deliveries so far have relied on opportunities.

C.3.8 Refueling

The presence of a fuel drum at the truck's current location constitutes an opportunity to refuel, which PARETO takes.

```
    *** Addressing goal 1 <MONITOR-FUEL-LEVEL> at 1066
         ... Processing task: <LOOK-FOR-FUEL> ::  [1:14] at 1066
         ... Processing task: <LOOK-FOR-FUEL> ::  [1:14] at 1067
         ... Task succeeded: <LOOK-FOR-FUEL> ::  [1:14] at 1067
```

Although PARETO was planning to go the base for fuel, there is a fuel-drum here.

```
         ... Processing task: <DUMP-IN-LOCAL-FUEL-DRUMS>
              ::  [1:15] at 1067
(LADDER OBJ-10 disappeared at time 1081)
     <=> Changing goals at 1084 from 1 to 0 <MONITOR-CURRENT-LOCATION>
         ... Processing task: <MONITOR-CURRENT-LOCATION>
              ::  [0:0] at 1084
     --- Abandoning side task at 1088 : MONITOR-CURRENT-LOCATION  [0:0]
     <=> Changing goals at 1088 from 0 to 1 <MONITOR-FUEL-LEVEL>
         ... Processing task: <DUMP-IN-LOCAL-FUEL-DRUMS>
              ::  [1:15] at 1089
         ... Task succeeded: <DUMP-IN-LOCAL-FUEL-DRUMS>
              ::  [1:15] at 1089
     --- Abandoning main task at 1089 :  WATCH-FOR-LOW-FUEL  [1:2]
     >>> Main task:  [24:19] supplanted by  [1:4]
         <PUT-DOWN-EMPTIES>
         ... Processing task: <PUT-DOWN-EMPTIES> ::  [1:4] at 1089
     --- Abandoning main task at 1094 :  PUT-DOWN-EMPTIES  [1:4]
```

C.3.9 The task continues

The fuel in the fuel drum is enough to avert imminent danger of running out of fuel. The truck heads off for the paintshop again to look for some wallpaper, and at last finds some.

```
     <=> Changing goals at 1094 from 1 to 24
         <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
         ... Processing task: <TRUCK-TRAVEL-TO PAINTSHOP>
              ::  [24:10] at 1094
(WOOD OBJ-34 appeared at time 1120)
(WOOD OBJ-34 disappeared at time 1120)
              *** Arrived at SHERIDAN-RD at time 1120
     <=> Changing goals at 1120 from 24 to 0 <MONITOR-CURRENT-LOCATION>
         ... Processing task: <MONITOR-CURRENT-LOCATION>
              ::  [0:0] at 1120
(BAG OBJ-23 appeared at time 1124)
     --- Abandoning side task at 1124 : MONITOR-CURRENT-LOCATION  [0:0]
     <=> Changing goals at 1124 from 0 to 24
         <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
              *** Arrived at PAINTSHOP at time 1152
(WOOD OBJ-34 appeared at time 1152)
     <=> Changing goals at 1152 from 24 to 0 <MONITOR-CURRENT-LOCATION>
         ... Processing task: <MONITOR-CURRENT-LOCATION>
              ::  [0:0] at 1152
     --- Abandoning side task at 1156 : MONITOR-CURRENT-LOCATION  [0:0]
     <=> Changing goals at 1156 from 0 to 24
         <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
         ... Processing task:
              <FIND-OBJECT WALLPAPER => ?PLACE-NAME ?OBJECT>
```

```
             ::  [24:5] at 1156
        ... Task succeeded:
             <FIND-OBJECT WALLPAPER => PAINTSHOP ITEM-198>
             ::  [24:5] at 1156
```

And about time too. PARETO started to look for wallpaper at time 330.

```
        ... Processing task:
             <LOAD-PAYLOAD-OBJECT PAINTSHOP ITEM-198 WALLPAPER>
             ::  [24:6] at 1156
```

(WOOD OBJ-34 disappeared at time 1170)

```
    <=> Changing goals at 1173 from 24 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 1173
    --- Abandoning side task at 1177 : MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 1177 from 0 to 24
        <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
        ... Processing task:
             <LOAD-PAYLOAD-OBJECT PAINTSHOP ITEM-198 WALLPAPER>
             ::  [24:6] at 1182
        ... Task succeeded:
             <LOAD-PAYLOAD-OBJECT PAINTSHOP ITEM-198 WALLPAPER>
             ::  [24:6] at 1182
```

The wallpaper has been loaded into the truck: next, it must be taken to the delivery site.

```
        ... Processing task: <TAKE-OBJECT-TO-SITE ITEM-198 TECH>
             ::  [24:7] at 1182
```

(NAILS OBJ-9 appeared at time 1212)

(BAG OBJ-39 disappeared at time 1212)

```
        ... Processing task: <TRUCK-TRAVEL-TO SHERIDAN-RD>
             ::  [24:5] at 1182
            *** Arrived at SHERIDAN-RD at time 1212
```

C.3.10 Completing the delivery

The truck's fuel level is again too low. However, it is at the delivery site so can take advantage of the opportunity to deliver the wallpaper before going off to find some fuel.

```
    >>> Main task:  [24:14] supplanted by  [1:2]
        <WATCH-FOR-LOW-FUEL [SET: (ITEM-5 ITEM-7)]>
    <=> Changing goals at 1212 from 24 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 1212
    --- Abandoning side task at 1216 : MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 1216 from 0 to 1 <MONITOR-FUEL-LEVEL>
        ... Processing task:
             <WATCH-FOR-LOW-FUEL [SET: (ITEM-5 ITEM-7)]>
             ::  [1:2] at 1216
        ... Processing task: <LOOK-FOR-FUEL> ::  [1:10] at 1216
    --> New side task chosen at 1216 :  <TRUCK-MOVE-P> [24:14]
```

There is an opportunity to deliver the wallpaper.

```
        ... Processing task: <TRUCK-TRAVEL-TO SHERIDAN-RD>
             ::  [24:5] at 1216
        ... Task succeeded: <TRUCK-TRAVEL-TO SHERIDAN-RD>
             ::  [24:5] at 1216
        ... Processing task: <TAKE-OBJECT-TO-SITE ITEM-198 TECH>
```

```
             ::  [24:7] at 1216
        ... Task succeeded: <TAKE-OBJECT-TO-SITE ITEM-198 TECH>
             ::  [24:7] at 1216
        ... Processing task: <UNLOAD-AT TECH ITEM-198>
             ::  [24:8] at 1216
```

(NAILS OBJ-9 disappeared at time 1220)

```
    <=> Changing goals at 1232 from 24 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 1232
    --- Abandoning side task at 1236 : MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 1236 from 0 to 24
        <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
```

*** ORDER-2 filled at time 1247

```
        ... Processing task: <UNLOAD-AT TECH ITEM-198>
             ::  [24:8] at 1247
        ... Task succeeded: <UNLOAD-AT TECH ITEM-198> ::[24:8] at 1247
        ... Processing task: <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
             ::  [24:24] at 1247
        ... Task succeeded: <DELIVER-OBJECT WALLPAPER TECH ORDER-2>
             ::  [24:24] at 1247
--> Order ORDER-2 for a roll of WALLPAPER at TECH
    filled at time 1247 - 0 orders remain to be filled
```

This goal has finally been achieved, 917 time units after first being addressed.

C.3.11 More refueling

There are no more delivery goals, but the truck is still low on fuel and PARETO has a goal to refuel. It goes to the base, where it finds plenty of fuel.

```
    *** Addressing goal 1 <MONITOR-FUEL-LEVEL> at 1247
        ... Processing task: <TRUCK-TRAVEL-TO BASE> ::  [1:13] at 1247
    <=> Changing goals at 1252 from 1 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 1252
    --- Abandoning side task at 1256 : MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 1256 from 0 to 1 <MONITOR-FUEL-LEVEL>
            *** Arrived at MAPLE-AVE at time 1278
```

(SCISSORS OBJ-6 disappeared at time 1278)

```
    <=> Changing goals at 1278 from 1 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 1278
    --- Abandoning side task at 1282 : MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 1282 from 0 to 1 <MONITOR-FUEL-LEVEL>
            *** Arrived at BASE at time 1304
```

(SCISSORS OBJ-26 appeared at time 1304)

```
    <=> Changing goals at 1304 from 1 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 1304
    --- Abandoning side task at 1308 : MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 1308 from 0 to 1 <MONITOR-FUEL-LEVEL>
        ... Processing task: <LOOK-FOR-FUEL> ::  [1:10] at 1308
        ... Processing task: <LOOK-FOR-FUEL> ::  [1:10] at 1309
        ... Task succeeded: <LOOK-FOR-FUEL> ::  [1:10] at 1309
        ... Processing task: <DUMP-IN-LOCAL-FUEL-DRUMS>
```

```
             ::  [1:11] at 1309
    <=> Changing goals at 1325 from 1 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 1325
    --- Abandoning side task at 1329 : MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 1329 from 0 to 1 <MONITOR-FUEL-LEVEL>
        ... Processing task: <DUMP-IN-LOCAL-FUEL-DRUMS>
             ::  [1:11] at 1338
```

(WOOD OBJ-5 appeared at time 1344)

```
    <=> Changing goals at 1347 from 1 to 0 <MONITOR-CURRENT-LOCATION>
        ... Processing task: <MONITOR-CURRENT-LOCATION>
             ::  [0:0] at 1347
```

(TWINE OBJ-3 appeared at time 1351)

```
    --- Abandoning side task at 1351 : MONITOR-CURRENT-LOCATION  [0:0]
    <=> Changing goals at 1351 from 0 to 1 <MONITOR-FUEL-LEVEL>
        ... Processing task: <DUMP-IN-LOCAL-FUEL-DRUMS>
             ::  [1:11] at 1361
        ... Task succeeded: <DUMP-IN-LOCAL-FUEL-DRUMS>
             ::  [1:11] at 1361
        ... Processing task: <PUT-DOWN-EMPTIES> ::  [1:4] at 1361
    --- Abandoning main task at 1366 :  PUT-DOWN-EMPTIES  [1:4]
--> Done.
```

PARETO has no more goals to address.

```
ORDER REPORT
  Number of orders generated: 8
  Number of orders completed: 8
  Number of orders pending:   0
  Number of orders failed:   0
  Completed orders:
   ORDER-1: a SAW to ILS  -   95, 330
   ORDER-6: a KNIFE to TECH  -  254, 812
   ORDER-8: a plank of WOOD to TECH  -  299, 849
   ORDER-3: a roll of TAPE to HOUSE  -  144, 912
   ORDER-5: a pair of SCISSORS to HOUSE  -  245, 942
   ORDER-4: some SCREWS to ILS  -  181,1036
   ORDER-7: a ball of TWINE to ILS  -  269,1066
   ORDER-2: a roll of WALLPAPER to TECH  -  126,1247
```

C.4 Summary

In this run PARETO fulfilled eight delivery goals. Only two of these were addressed as main tasks: the remaining six were achieved solely through the taking of opportunities. After the initialization process, PARETO addressed the refueling task twice. During the whole run, the task to scan the truck's surroundings was addressed 55 times, and PARETO changed the task being addressed a total of 128 times.