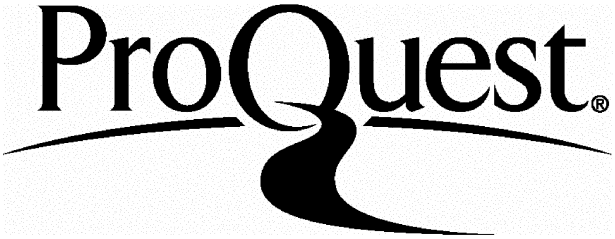
Self-Motivated Planning in Autonomous Agents

Alexandra Margrit Coddington

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of the
University of London.

Department of Computer Science
University College London

2001



Abstract

This thesis describes research which is concerned principally with the design of a planning/execution architecture to be used within an autonomous motivated agent situated within a real world environment. To further its aims, such an agent is capable of generating its own goals and of planning and acting to achieve those goals in real or simulated time. Research in planning tends to assume that goals are externally generated by a human operator and that planning is complete once all outstanding goals have been achieved. Because our agent is able to continually generate its own goals, planning may never be complete which means planning and execution are ongoing activities. In addition, constraints upon time may mean it is not possible for the agent to achieve all goals. The agent must therefore be able to both reason about time (in particular the durations of actions) and prioritise its goals.

A prototype planning/execution architecture designed to address these issues is described. The proposed architecture extends the classical planning framework to take into account both the context of the agent (where context may be interpreted as a function of both the perceived external environment and the internal state of the agent), and problems associated with planning and acting in real time. We argue that context, captured by modelling the motivations of an agent situated within an environment, plays an important role in the generation of goals and enables the agent to determine the importance of such goals. A crucial component of the planning/execution architecture is a temporal manager which enables the agent to reason about whether or not there is sufficient time available to execute all of the actions within a partial plan and to calculate deadlines for actions and outstanding goals. The importance and deadlines associated with goals enables the agent to prioritise those goals so that should there be insufficient time to achieve all goals, the agent can abandon some goals in favour of others. The architecture also demonstrates how modelling the motivations of an agent provides an effective means for evaluating and selecting partial plans. Finally, the architecture enables the agent to "execute" actions and to cope with unpredictable changes that may occur within its environment.

Acknowledgements

I would like to thank my supervisor, Maria Fox, for her support and encouragement. In addition, I thank Michael Luck, for his friendship and support, inspiration, and for the many hours spent discussing this thesis.

My close friends, Paul Archbold, Bridget Carey and Simon Webster, my sister-in-law, Kathryn, my brother, Peter, my nephews, Sebastian and Conor, and my parents, Alan and Marianne, have provided much love and friendship over the years.

Finally I would like to thank Jay Smith for his love and for helping me meet the deadline for this thesis.

This research was carried out with the help of a Science and Engineering Research Council studentship.

Contents

Chapter 7

Chapter 8

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

There are many aspects to intelligent behaviour. Of these aspects, perhaps the most important is the ability to reason about the world in such a way that allows intelligent agents to take actions that achieve their chosen aims or goals. This problem of designing and following sequences of actions is the focus of a major branch of Artificial Intelligence (AI) known as *planning*.

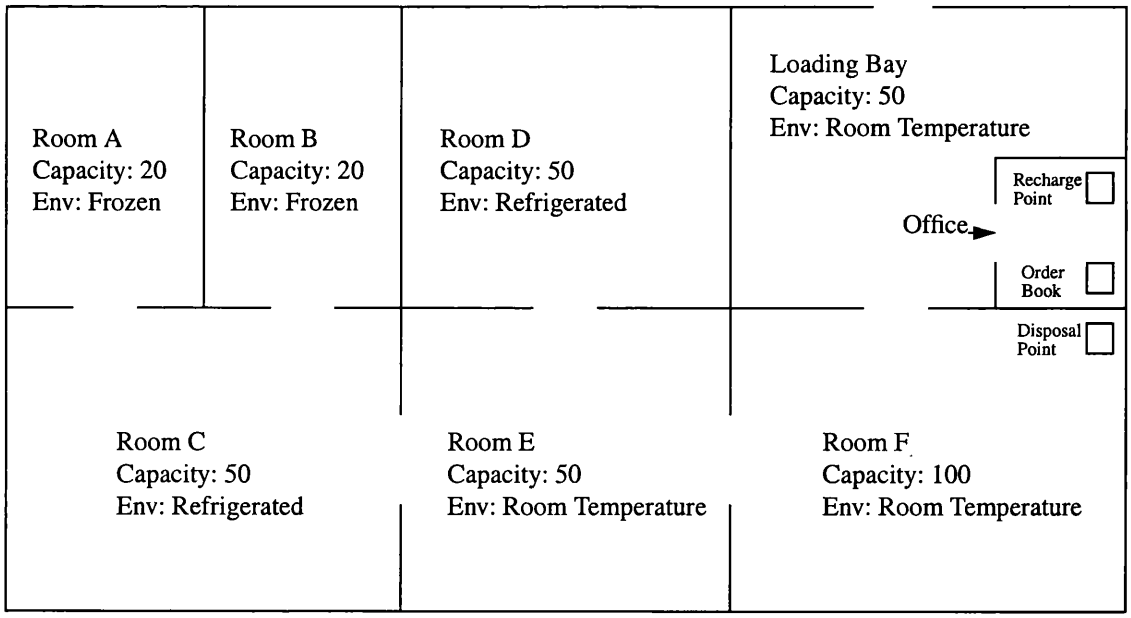


Figure 1.1 A Warehouse Environment

The most vivid example of planning is the HSTS/Remote Agent planner [Muscettola 94] which was used to generate plans during the Remote Agent Experiment [Muscettola et al 98] onboard the NASA Deep Space One spacecraft. This planner successfully generated complex plans that included turns, observations, navigation, and other spacecraft opera-

tions, while taking into account limited resources, task durations and time limits. Figure 1.1 illustrates a simple warehouse environment developed by [Norman 97] which is inhabited by two or more warehouse agents.

The warehouse environment consists of a number of rooms with doors between those rooms. Each room has two characteristics: capacity - which determines how many units of stock can be stored in the room; and the type of storage environment that the room provides - this may be either frozen, refrigerated or at room temperature.

The type of storage environment governs the type of commodity that can be stored in the room. For example, Room A can hold 20 units of stock and is kept at a temperature a few degrees below freezing which makes it suitable for the storage of frozen goods such as frozen peas but unsuitable for the storage of commodities such as vegetables. The delivery of goods from suppliers and the collection of orders by customers is done through the loading bay, which is used for the temporary storage of delivered goods (from suppliers) and for the preparation of orders that are to be collected by customers. The office contains an order book and a recharge point used by the warehouse agents to recharge their batteries. The warehouse also contains a disposal point (in Room F) which is used to dispose of commodities that are past their sell-by date.

The warehouse agents are responsible for running and maintaining the warehouse which involves performing the following tasks.

1. Preparing orders for customers. Customers send requests for orders (indicating which types of commodity, the amount required, together with a deadline indicating when they wish to receive their order) via the order book. Preparing an order involves fetching the appropriate commodities from the storage rooms and placing them in the loading bay in time for collection by the customer.
2. Restocking the warehouse - the warehouse agents must ensure there is sufficient stock available in the warehouse. This task involves ordering new stock from suppliers, and, when that stock is delivered to the loading bay, moving the stock into the appropriate storage environment.
3. Disposing of old stock - if stock is not sold before its sell-by date, the warehouse agents must dispose of that stock at the disposal point in Room F.
4. Recharging batteries - the warehouse agents consume battery power as they move around the warehouse environment and must therefore periodically recharge their batteries.

It can be seen that in order to run the warehouse efficiently, the warehouse agents need to be able to plan a sequence of activities. For example, the warehouse agents need to plan to ensure they have sufficient battery power in order to prepare orders or to stock shelves, and that there is enough stock in the warehouse to satisfy orders. The design of a planning component suitable for use by agents such as the warehouse agents is the focus of research addressed in this thesis.

1.2 Planning

1.2.1 What is planning?

Table 1.1 Operator schemas

name:	*stack(?x ?y)*
precondition:	*holding(?x) & clear(?y)*
delete:	*holding(?x) & clear(?y)*
add:	*on(?x ?y) & armempty*

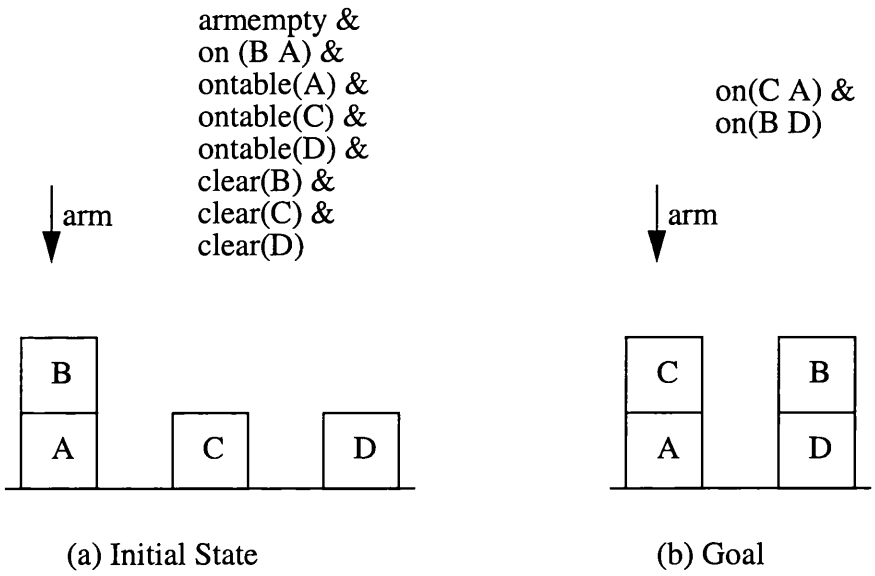
name:	*unstack(?x ?y)*
precondition:	*on(?x ?y) & clear(?x) & armempty*
delete:	*on(?x ?y) & armempty*
add:	*holding(?x) & clear(?y)*

name:	*putdown(?x)*
precondition:	*holding(?x)*
delete:	*holding(?x)*
add:	*ontable(?x) & armempty*

name:	*pickup(?x)*
precondition:	*ontable(?x) & clear(?x) & armempty*
delete:	*ontable(?x) & armempty*
add:	*holding(?x)*

In essence, planning is the process of formulating a sequence of actions that, when executed, will achieve a goal. Any planning agent will be able to perform a certain set of actions within its competence. For example, a robot may be able to pick things up or put things down, move forwards, turn, and so on. These are the actions which it must organise into a plan for achieving some state of the world or goal, such as moving and stacking crates in one corner of a warehouse. A goal is simply a description of some state of the world that is to be achieved. A simple formulation of the planning process requires three inputs [Weld 99].

1. A description of the initial state of the world.
2. A description of the agent's goal.
3. A description of the possible actions that can be performed by the agent.



(a) Initial State (b) Goal

Figure 1.2 A Blocks World Problem

The planner's output is a sequence of actions which, when executed in any world satisfying the initial state description, will achieve the goal. A variety of languages such as propositional logic or first order predicate calculus may be used to describe/represent the initial state of the world, the goal and the possible actions. Many planning algorithms use the representation developed for use by STRIPS [Fikes & Nilsson 71] (STanford Research Institute Problem Solver), an influential planner built in the 1970s to control a mobile robot. In the representation used by STRIPS, the initial state of the world is described as a complete set of ground literals while the agent's goal is described as a propositional conjunction. Each possible action is described using a conjunctive *precondition*, representing the facts which must be true in the current state of the world in order to execute the action, and a conjunctive *effect*, which represents facts which are no longer true (a conjunction of propositions known as *delete* propositions) as well as facts which become true (a conjunction of propositions known as *add* propositions) once the action has been executed. Each action therefore describes a transition function mapping one state or world to another - an action can be executed in any state/world *s* satisfying its precondition while the resulting state/world (i.e. the state/world that arises once the action has been executed) is described by removing/deleting the action's *delete* proposi-

tions from *s* and adding the action's *add* propositions to *s*. For example, in order to perform the activity of stacking some block *?x* onto some block *?y* (see *stack(?x ?y)* in table 1.1), the agent must be holding *?x* and *?y* must be clear (this is the *precondition*). Once the action has been executed, the agent is no longer holding *?x* and *?y* is no longer clear (these are the *delete* effects), *?x* is now on *?y* and the agent's arm is empty (these are the *add* effects).

1.2.2 An example

A typical toy planning problem, related to the the warehouse example above, involves the manipulation of toy blocks on a tabletop. Figure 1.2 shows the current or initial state on the left, and the desired goal state on the right. The planning agent is capable of performing four types of action shown in table 1.1 (these are known as operator schemas or action templates) which involve stacking some block *?x* onto some block *?y*, unstacking some block *?x* from some block *?y*, putting some block *?x* onto the table, and picking up some block *?x* off the table (*?x* and *?y* are variables). The task of the planner is to construct a plan using these operator schemas/action templates to achieve the goal of figure 1.2. Such a plan is shown in table 1.2. The planning problem is precisely this task of constructing such plans.

Table 1.2 A plan to achieve a goal

order:	*1*	*2*	*3*	*4*
name:	*unstack(B A)*	*stack(B D)*	*pickup(C)*	*stack(C A)*
precondition:	*on(B A) & clear(B) & armempty*	*holding(B) & clear(D)*	*ontable(C) & clear(C) & armempty*	*holding(C) & clear(A)*
delete:	*on(B A) & armempty*	*holding(B) & clear(D)*	*ontable(C) & armempty*	*holding(C) & clear(A)*
add:	*holding(B) & clear(A)*	*on(B D) & armempty*	*holding(C)*	*on(C A & armempty*

1.2.3 Recent developments in planning

In 1995 Graphplan [Blum & Furst 97] was developed - this planner was significantly more efficient than earlier classical planners such as STRIPS [Fikes & Nilsson 71], NOAH [Sacerdoti 75], NONLIN [Tate 77], TWEAK [Chapman 87], SNLP [McAllester

& Rosenblitt 91] and UCPOP[Penberthy & Weld 92], which could only solve simple problems containing a small number of actions. Since Graphplan, there have been a number of significant developments in planning which we summarise in the following sections.

The AIPS planning competitions

In 1998, the first planning competition was held at AIPS 98 (the 4th International Conference on Artificial Intelligence Planning Systems) in which five planners competed to determine how quickly and how optimally they could solve various benchmark problems. These planners were Blackbox [Kautz & Selman 99], IPP [Koehler et al 97], HSP [Bonet & Geffner 99], STAN [Long & Fox 99] and SGP [Weld et al 98]. Three of these planners, IPP, STAN and SGP, extended Graphplan in various ways to further increase efficiency.

In 2000, AIPS 2000 held the second planning competition in which fifteen planners, including improved versions of STAN and HSP (HSP-2), competed. The most notable planners in this competition were FF ([Hoffmann & Nebel 00], [Hoffman 00]), TALplanner [Doherty & Kvarnström 99], and the improved version of STAN. FF is a domain-independent, forward-chaining state space planner which uses several heuristic techniques to guide its search - the main heuristic principle behind FF was originally developed for the HSP system which also took part in the AIPS competition. TALplanner is based on ideas developed by Bacchus and Kabanza [Bacchus & Kabanza 98] in which temporal logics were used to express search control knowledge for planning (these ideas were implemented in the TLplan system [Bacchus & Kabanza 98]). TALplanner is domain-dependent, however, which means that its performance relies upon hand-crafted control rules.

Decision-theoretic planning

A large body of work has been undertaken by researchers investigating the problem of planning under uncertainty. Many planning problems of interest to researchers in this field have been modelled as Markov Decision Processes (MDPs) or as Partial Order Markov Decision Processes (POMDPs). [Boutilier et al 99], and [Blythe 99] present a detailed overview of work undertaken in this area.

1.3 Weaknesses of Current Planning Technology

A large body of work in planning (such as that described in the previous section) has been concerned with developing fast, efficient algorithms which solve user-supplied planning problems in an optimal manner. However, these planners still ignore many of the issues which must be addressed for effective real-world planning, and which are itemised below.

- **Situatedness**. Goals are independently posed to the planner by an external agent (such as a user) and the planner is not situated within a real environment (i.e. the planner is computer-based). This means that there is no information available to the planner as to the circumstances that caused those goals to be generated. Such information is potentially very valuable in constraining plan formulation which can be combinatoric. [Brooks 86].
- **Unpredictability**. Classical planning assumes complete (or at least sufficient) knowledge of the domain, and that changes in the world are brought about only by the *planning* agent. Moreover, it assumes perfect execution of plans, so that when the agent executes actions within the plan, the outcome of those actions will be as intended. These restrictions will not necessarily hold in real-world environments, where actions may not result in the intended outcome and where other agents may change the world unexpectedly. Classical planners are therefore brittle - the plans they produce cannot easily adapt to unforeseen changes in the world or to unexpected outcome of action.
- **Planning and Execution**. Typically, once a plan is generated, one or more agents must execute that plan to completion. However, if the world changes during either planning or execution, the plan or parts of the plan may fail. In addition, with separate planning and execution, all goals are presented simultaneously to the planner prior to the start of plan construction, and no new goals may be incorporated into the plan after that point.
- **Execution Time**. Classical planning has no notion of real time. It assumes that the execution of an action is a function that instantaneously maps one state to the next. However, in the real world actions take time to execute, and this may cause problems for goals which must be achieved by some deadline.
- **Embodiment**. Heuristics governing the selection of partial plans for further refinement take into account such factors as the number of outstanding goals, the number of

actions and the number of goals already achieved. However, the desires and preferences of the planning agent are not considered, but these may provide additional constraints that allow a better selection of the actions available to achieve a goal. This relates strongly to the concerns of Brooks, for example, who argues that *embodiment* is necessary for effective functioning in the real world [Brooks 86], [Brooks 91]. As a consequence, all goals are treated with equal priority so that there is no notion of one goal being more important than another.

1.4 Aims and Motivation

In response to the weaknesses of current planning technology identified above, this research is concerned with the design of a planning/execution component to be incorporated onboard an autonomous motivated planning agent. We propose an architecture that extends the classical planning framework to take into account the *context* of the agent (where *context* may be interpreted as being a function of both the external environment and the internal state of the agent), problems associated with planning and acting in real time and problems associated with interleaving planning and execution within an unpredictable environment. We argue that *context*, captured by modelling the motivations of an agent situated within an environment, plays in important role in defining optimal plans.

In the following section we describe the properties required of a planning agent (such as the warehouse agent) in order that it can act within its environment to achieve its aims.

1.5 The Agent

1.5.1 Capability

Planning

The most important requirement of an agent is that it must be able to reason about its environment in such a way that allows it to take action to achieve its chosen aims. To do this, the agent must be able to generate and execute plans - this is the focus of the research described in this thesis. For example, in the warehouse domain, the warehouse agent must be able to generate and execute plans to enable it to satisfy order requests from customers, to restock the warehouse, to recharge its batteries, and so on.

Sensors

The agent requires sensors in order to reason and act effectively within its environment. It is assumed that the agent's environment is accessible [Russell & Norvig 95] - this means that the agent's sensors are perfect (i.e. the sensors deliver data which is accurate) and that they enable the agent to construct a sufficient (i.e. the sensors detect all aspects of the environment that are relevant to the choice of action), accurate, symbolic representation of the agent's environment. The agent's sensors may, in addition, enable the agent to perceive aspects of its own internal state (for example, the warehouse agent may be able to monitor battery charge). However, the agent's sensors only allow it to detect the external state of objects or other agents within the environment. For example, a warehouse agent can only perceive the external features of commodities such as frozen peas or of other agents, and cannot perceive the internal workings of other agents.

Actuators

The agent must be capable of acting within its environment in such a way as to enable it to achieve its goals. For example, the warehouse agent must be able to perform actions such as picking stock up off shelves, placing stock on shelves, moving stock from one location (such as the loading bay) to another location (such as Room B), recharging its battery, disposing of out-of-date stock, and so on. When planning to achieve a set of goals or aims, it is assumed that the agent's actions are deterministic (i.e. that the outcome of each action is known). For example, when the warehouse agent plans to pick up some commodity, it is assumed that as a consequence of picking up that commodity, the agent will be holding that commodity. However, when executing an action (as opposed to when creating a plan which contains that action) such as picking up some commodity, the actual outcome may not be the same as the predicted outcome. For example, the warehouse agent may drop the commodity whilst picking it up off the shelf, or the time taken to pick up the commodity may be longer or shorter than anticipated.

In addition, the work addressed in this thesis assumes that the agent cannot execute more than one action in parallel.

Goal Autonomy

In order to further its aims, the agent should be able to generate goals in response to immediate changes in the environment as well as to predictions about future changes that

may occur (the ability to plan gives the agent the means to predict what might happen in the future as a consequence of executing that plan). Such goals may be generated to enable the agent to prevent undesirable situations from occurring or to take advantage of opportunities presented by the current situation or by future predicted situations. For example, as a consequence of creating a plan which involves satisfying various order requests, the agent can predict that at some point in the future the warehouse will run out of some commodity. In response to this prediction, the warehouse agent might generate the goal of restocking the warehouse with that commodity. If the agent perceives that certain items of stock have become out-of-date, it might generate the goal of disposing of such stock items.

Because the environment is constantly changing, the agent must continually generate goals (the warehouse agent is continually generating goals in response to requests from customers for example). Planning is therefore ongoing which means that planning and acting must be interleaved.

In order to be effective, the agent must, in addition, be able to direct its attention to goals that are most appropriate for action at any given moment. The agent may have multiple conflicting goals. For example, the warehouse agent must conserve battery power whilst preparing an order in time for a customer. To deal with such conflicts, the agent must be able to prioritise goals. As the environment changes, goals and their priorities change which means that the agent must be able to alter its focus of attention (i.e. by changing which goal it is presently acting on). If battery power is extremely low, it becomes more important that the warehouse agent acts to recharge its battery. If battery power is less low and the customer is important, it becomes more important that the warehouse agent acts to prepare the customer's order.

1.5.2 The agent and the environment

Situatedness

The agent must be situated within an environment. The warehouse agent is situated within the warehouse environment - it interacts with that environment through sensors and actuators and, to successfully achieve its aims, must respond to that environment in a timely fashion.

Embodiment

The agent is grounded within an environment - it experiences that environment directly through sensors and is capable of acting within that environment using its actuators. The actions taken by the agent are part of a dynamic between the agent and its environment and have immediate feedback on the agent's sensations.

Unpredictability

The environment is not predictable to the agent. This may be due to various factors.

1. When executing an action, the actual outcome may differ from the predicted outcome (as discussed above).
2. Other agents act in the environment - because the agent in question has only access to the external state of other agents acting within the environment, the agent is unable to predict the consequences of the activities of other agents. For example, there may be several other agents of differing capabilities acting within the environment, but because the agent does not have knowledge of the capabilities of these agents, it will be unable to predict how they might act and change the environment. In the warehouse domain, another warehouse agent might be occupying the battery recharge point, thereby causing it to be unavailable.
3. Physical processes occurring within the environment may cause changes to that environment (for example, an ice cube will melt if it is exposed to a temperature greater than 0° Celcius) - the agent can only predict the consequences of physical processes if it has knowledge of such processes.

1.5.3 Time

The agent should plan and act in real or simulated time. Time passes (and the environment may change) while the agent both *deliberates* and acts. Constrained by passing time, it may not be possible for the agent to achieve all of its goals within the time available. In order to be effective, the agent must be able to prioritise goals. For example, the warehouse agent may have two goals which involve preparing two orders for two different customers. Due to time constraints, it may not be possible to prepare both orders in time to meet their deadlines. One customer may be a long-valued customer whilst the other may be unreliable (i.e. they may not yet have paid for a previous order) in which case the first customer's order may take priority.

1.5.4 Context

The context of the agent is important as it constrains the goals that the agent might generate, enables the agent to prioritise those goals, and constrains the plan selection process. Depending on context, the agent may wish to prioritise goals and allocate its resources accordingly (more resources are likely to be devoted to the achievement of high priority goals). The context of an agent is determined by the following factors.

- The agent's capabilities - this includes the actions the agent is capable of performing.
- The environment in which the agent is placed - this includes the current state of the environment (including the agent's internal state) as perceived by the agent as well as predicted future states of the environment.
- The agent's desires or preferences which are captured by modelling the agent's motivations. A *motivation* is any desire or preference which affects the outcome of a given reasoning task [Kunda 90] (motivations will be discussed further in chapter 2, section 2.2.3).

For example, the context of the warehouse agent constrains which goals the agent might generate. Firstly, the warehouse agent will generate goals it is capable of achieving - these are goals which require actions that it is capable of performing (there is no point generating a goal which involves flying if the agent cannot fly). Secondly, goals may be generated in response to changes within the environment - for example, if the warehouse agent perceives certain commodities to be past their sell-by date, it may generate the goal of disposing of such commodities. In addition, goals may be generated in response to predicted future states of the environment - for example, if the agent predicts (on the basis of customer orders) that in the future it will run out of certain commodities, it will generate the goal of replenishing such commodities. Finally, goals may be generated partly in order to fulfil the warehouse agent's desires - for example, the warehouse agent may desire to satisfy as many order requests as possible in order to maximise profits.

In addition, context enables the agent to prioritise goals. The warehouse agent may attach a higher priority to fulfilling one customer's order request than another, simply because it might prefer the first customer, the first customer is a long-standing regular customer, or the first customer is more reliable (these constitute the warehouse agent's preferences). Changes in the environment, such as changes in the agent's power supply, might cause the agent to attach a high (or low) priority to the goal of recharging its bat-

tery.

Finally, context constrains which plan the agent chooses to follow. For example, the warehouse agent might choose one plan in favour of another as it contains actions which conserve battery power, and which satisfy a large number of high priority orders.

1.5.5 Plan failure and recovery

Whilst acting within an environment, an agent may fail to achieve some of its goals or objectives. This may occur simply because there is insufficient time available or as a consequence of the environment being unpredictable. When executing an action, the outcome of execution may not be as predicted - the action may take longer to execute than expected, or may result in unintended effects. In addition, other agents or physical processes may cause unforeseen changes. Such unpredicted changes may undermine the agent's attempts to achieve its goals. An unforeseen change to the environment may mean that the agent is no longer able to achieve one or more of its goals simply because there is not enough time.

The agent should have the ability to detect when unforeseen changes to the environment may undermine its attempts to achieve its objectives, and to recover from such changes by replanning. If there is insufficient time available to achieve one or more goals, the agent should have the ability to recover by abandoning these goals and focussing attention on its remaining goals.

1.5.6 Summary

The agent should be able to generate and prioritise goals and generate plans to achieve those goals. However, because the agent is able to prioritise goals, it may devote more resources to ensuring the achievement of goals of higher priority which means there is a greater chance that such goals will be achieved. Nevertheless, due to time constraints, the agent cannot be guaranteed to achieve even high priority goals, and will therefore not be very effective within safety-critical domains or within any domain where it is essential that goals are achieved. In addition, because planning and acting are interleaved we cannot guarantee the agent will come up with optimal solutions when achieving goals. In order to guarantee optimal solutions to planning problems it is essential the planning component is presented at the outset of planning with the goals it must achieve. The agent we propose might begin planning to achieve a goal, and, whilst planning, be pre-

sented with new goals. We are principally concerned with the design of an ongoing planning and execution mechanism and not with the design of a planning mechanism that produces optimal plans.

1.6 Contributions of this Thesis

This thesis is concerned with the design and implementation of a planning/execution component for the class of agents with the properties described above in section 1.5. The planning component is designed to be domain-independent as well as agent-independent so long as the agent has the properties described above. The underlying problem being addressed is the use of motivation to direct problem solving behaviour in realistic environments when time is a critical resource. The specific aims and contributions of this research are described below.

- **Context.** The use of context to generate and prioritise goals as well as to act as a heuristic in the selection of plans. *Good* plans take into account the context of the agent and maximise goal achievement. Context is partially represented by modelling the motivations of the agent.
- **Planning and acting in real or simulated time.** Time constraints affect which goals can be achieved - goals have deadlines and actions take time to execute.
- **Interleaving planning and execution.** Planning and execution are ongoing activities as the agent continually generates new goals. This means that planning and execution must be interleaved.
- **Planning and acting within an unpredictable environment.** The planning/execution system must be able to cope with unforeseen changes that may occur within the environment.

The planning/execution architecture is implemented as a computer program which means that the agent and its sensors and actuators are modelled and are not real.

1.7 Thesis Overview

The architecture of the planning/execution component is presented in chapter 2 with a brief description of the purpose and function of each component. In chapter 3, the control of the planning/execution system is discussed, and comparisons are made with other sys-

tems. The representations used for the various data structures used by the planning/execution system are presented in chapter 4. We then describe an example domain, the truck world domain, which is used to demonstrate the behaviour of components in the remainder of the thesis. Finally in chapter 4 we describe the component reponsible for deciding whether to plan to achieve a goal or whether to execute. In chapters 5 and 6 the processes involved in planning to achieve a goal are described while chapter 7 focusses upon execution and plan recovery. Finally, in chapter 8 the work is evaluated and further work is discussed.

Chapter 2

An Overview of the Architecture

2.1 Introduction

Currently the focus of much work in Artificial Intelligence is on "agent" designs in which previously isolated processes such as planning, scheduling, learning, etc., are integrated [Wooldridge & Jennings 95]. The objective of the work presented in this thesis is to present a planning/execution architecture capable of being used within an agent which is situated within an environment [Brooks 86], [Brooks 91] and which is capable both of generating goals to further its aims (such an agent displays *goal autonomy*) and of achieving those goals by acting within that environment. Context, a function of the agent's internal state and the perceived current and predicted future states of the environment, plays a vital role in enabling the agent to generate goals. Such goals may allow the agent to avoid undesirable situations or to take advantage of opportunities. It will be more important for the agent to achieve some goals rather than others - context also enables the agent to therefore prioritise its goals. The context of the agent (which may be captured by modelling the motivations of the agent) not only enables the agent to generate and prioritise goals but enables the agent to choose amongst the plans developed to achieve such goals. In addition, because the agent is acting in real time there will not always be sufficient time for it to achieve all of its outstanding goals. The agent therefore tries to achieve as many of its more important goals as it can within the time available. The agent's domain is unpredictable to the agent - the outcome of execution may not be as predicted while other agents may be acting to change the environment in unforeseen ways. Finally, while acting within its environment, the agent may fail to achieve some of its goals due to time constraints as well as to the unpredictability of the environment. The agent must be capable of detecting and of recovering from such failure in certain situations.

In this chapter we present the design of a planning/execution component to be used by an agent with the attributes described above. The focus of much work in planning is primarily concerned with developing planners that are efficient and which produce plans that are both sound and complete ([Chapman 87], [Penberthy & Weld 92], [Blum & Furst 97]). However, such planners tend not to be situated within an environment (i.e. they are mainly computer-based) and so do not display goal autonomy - goals are presented to these planners by an external agent (a user). Planning ceases once all goals have been achieved - there is no facility for the achievement of new goals that are presented to the planner once the planner has begun to plan. In addition, these planners are primarily concerned with generating plans rather than executing those plans - once the planner has created a plan to achieve the goals posed by a user, planning ceases and the plan remains unexecuted. Many planners do not take into account time - in the real world, goals may have deadlines and actions take time to execute. Such time constraints may mean it is not possible to generate a plan to achieve all goals within the time available - many planners will simply report that they have failed to return a plan. Such planners are also unable to prioritise amongst goals - either a complete plan is generated which achieves all goals or the planner fails to return a plan. Finally, because many planners are not situated within an environment, context cannot be used as a search heuristic to aid in the selection of partial plans.

We present the architecture of a planning/execution component which extends the classical planning framework to take into account the agent requirements described earlier. To summarise, the planning/execution component has the following capabilities.

1. Planning and execution are ongoing activities - because the agent is capable of continually generating new goals, planning is never complete. This requires a planning algorithm which supports the interleaving of planning and execution.
2. The planner must be capable of accepting new goals generated by the agent at any point during the planning process.
3. The agent plans and executes in real time (time passes while the agent acts) which means the planner must be able to reason about time.
4. Time constraints mean the planner may be unable to achieve all goals within the time available - in order to be effective the planner must therefore be able to prioritise amongst its set of outstanding goals. In addition, once it has become apparent the planner cannot achieve one or more goals within the time available it must be

able to remove such goals together with their associated actions and constraints from the partial plan.

5. The planner must aim to achieve as many of its goals (preferably ones with high priority) as possible.
6. The *context* of the agent, captured by modelling the motivations of that agent, plays an important role in the planning process in that it enables the agent to generate goals, to prioritise amongst goals as well as to select the best plan to achieve such goals.
7. The environment is unpredictable - actions performed by the agent may not result in the intended outcome and other agents or physical processes may make unforeseen changes to the environment. This requires the agent to have the ability to recover from situations in which unforeseen changes to the environment adversely affect the agent's plan.

In the following sections we begin by presenting and describing a planning/execution architecture designed to meet the requirements listed above. Each component of the architecture is presented in turn beginning with a description of how the motivations of the agent are modelled and how such motivations influence the generation of goals. The process used to enable the planner to choose between achieving a goal and executing an action is described. Finally, the main component of the architecture, the goal achievement module, is described, followed by the execution and recovery components.

2.2 The Planning Architecture

In this section we present a high-level architecture of the planning/execution requirements for an agent with the attributes described in chapter 1. We describe the processes involved in achieving goals before discussing the purpose and function of each component within the architecture.

2.2.1 Components of the system

The system architecture is illustrated in figure 2.1.

Solid rectangular boxes represent the various processes within the planning/execution architecture that are the focus of this research. These processes have been implemented using Allegro Common Lisp. The dashed boxes represent two components

responsible for updating the agent's motivations and for generating/updating goals, two necessary components of any agent architecture, but which will not be addressed in the present work. These components have not been implemented.

The oval boxes represent knowledge sources. These include a partial plan which is passed between the various processes as well as the motivations of the planning agent. The partial plan representation includes a model of the agent's perception of the current and future states of the environment which are required by the reasoning processes. Thin arrows indicate the flow of data between knowledge sources and processes. For example the motivations are used by both the goal generation ("Generate/update goals") and the goal achievement ("Plan to achieve goal") components. All components require the knowledge encapsulated within a partial plan. Finally, thick arrows represent the flow of control between the various components of the system.

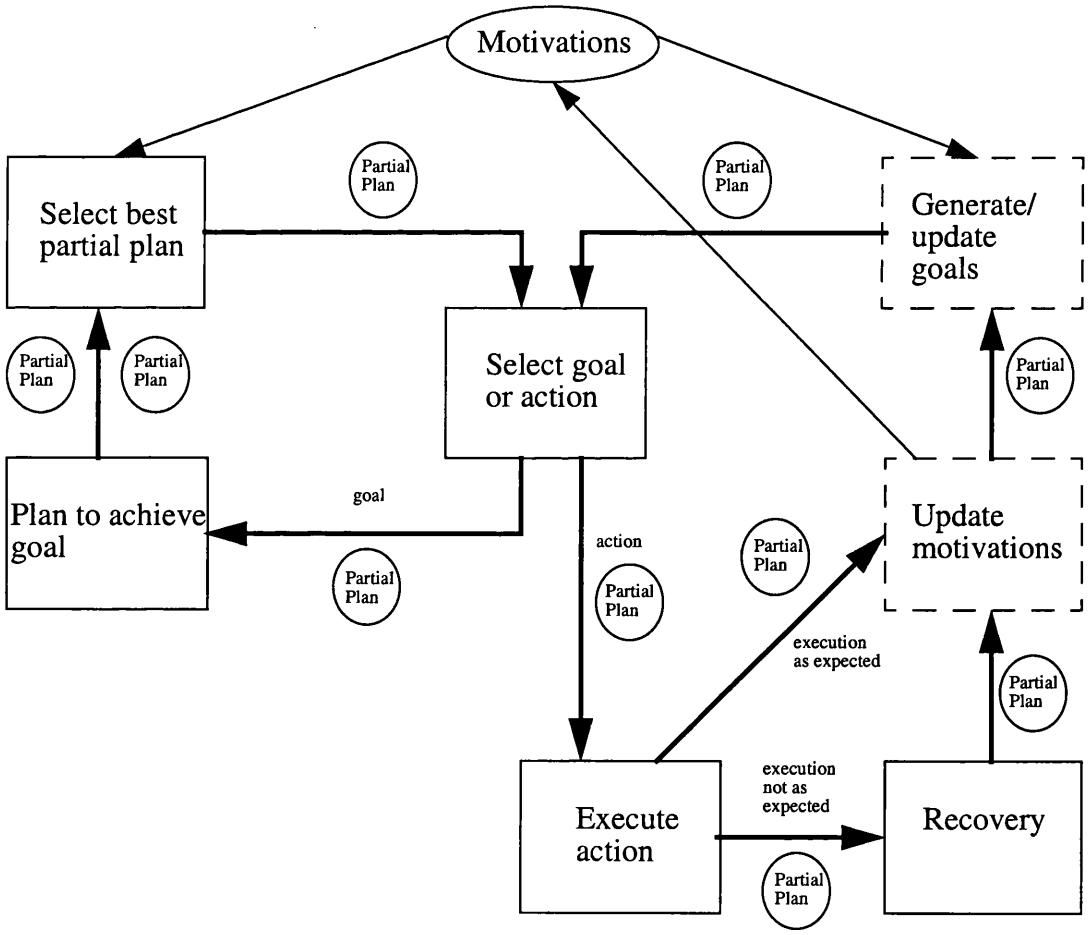


Figure 2.1 The Planning/Execution System

2.2.2 Interaction of components

This architecture can be viewed as a dynamic system in which the planning/execution

agent continually generates goals in response to the perceived current and predicted future states of the environment (encapsulated within a partial plan) as well as in response to its motivations. These newly generated goals, created by the component "Generate/update goals", are added to a partial plan together with deadlines for their achievement and values indicating their importance. The process, "Select goal or action", determines whether to achieve one of the goals/subgoals within a partial plan or whether to execute an action.

When a decision is made to achieve a goal/subgoal, a partial plan is passed to a non-linear goal achievement component ("Plan to achieve goal"). A new or existing action is selected by this component (using a set of operator schemas/action templates to represent the agent's capabilities which are not shown in the diagram) to achieve the goal/subgoal when executed, and constraints are posted. More than one new partial plan may be created by the "Plan to achieve goal" process as there may be more than one way of achieving the goal. In addition, new subgoals may be created (these are the preconditions of newly selected actions), as well as a new action, binding and temporal constraints, all of which are placed within each newly generated partial plan. The agent's motivations are used as a heuristic to evaluate each newly generated partial plan. The best partial plan is chosen from the search space of partial plans for subsequent refinement - this partial plan is passed back to the "Select goal or action" component which again determines whether to achieve one of the goals/subgoals within the partial plan or whether to execute an action.

When a decision is made to execute an action, the "Execute action" component "executes" the action by updating the partial plan to reflect any changes which have occurred within the environment following execution. An important part of this process involves determining whether or not the actual state of the environment following execution differs from the predicted state of the environment (this might occur if execution does not result in the intended outcome or if other agents have made unforeseen changes to the environment), and, if so, whether the actual state of the environment has an adverse effect on the newly updated partial plan. If the plan has been adversely affected by unpredicted changes occurring within the environment, the recovery component, "Recovery", may try to repair the partial plan. (This may involve adding goals which are to be reachieved to the partial plan.) The newly updated partial plan is then passed to the component "Update motivations" which is responsible for updating the agent's motivations

to reflect the changes that have been made to the environment following execution. The component "Generate/update goals" may then generate new goals in response to the changes in the environment as well as to changes in the agent's motivations. Finally, the cycle is repeated - the partial plan is passed to the "Select goal or action" component which decides whether to achieve a goal/subgoal or whether to execute an action.

The following sections describe the components of the architecture in more detail.

2.2.3 Motivations

Human, real world problem-solving involves both objective and subjective elements. Researchers such as [Picard 97] and [Bates et al 92] have been investigating how emotions affect human problem-solving. One of the main aims of this research is to examine how the *context* of an agent affects its problem-solving capabilities. To do this we must ensure that the planning component of an autonomous agent takes into account the *context* of that agent. We take *context* to be comprised of the following factors.

1. The current and predicted future states of the environment (this includes the current and predicted future physical states of the agent) which are encapsulated within a partial plan.
2. The current and predicted future desires and preferences of the agent - these are represented by modelling the motivations of the planning agent as discussed below.

This section discusses how *context*, in particular, how the agent's desires and preferences which are represented by modelling the motivations of a planning agent situated within an environment, affects the planning process.

When planning to achieve the same goal, two agents may create very different plans to achieve that goal [Luck 93], even though their external environment is the same. The different plans may be seen to arise as a result of differences in the internal states of those agents. These differences can be said to be due to the motivations of each agent.

An agent may be modelled as having a set of motivations or, in human terms, a set of basic needs and desires. For example, in animals these motivations might be such attributes as *hunger*, *curiosity* or *fear*. Different kinds of agent will have different motivations - while biological agents have the motivation, *hunger*, an agent concerned with the control of an aeroplane might have the motivation, *conserve fuel*. Associated with each motivation is a measure of strength which varies with changing circumstances and represents the driving force that directs action to satisfy the motivation. For example, the value

indicating the measure of strength associated with the motivation *hunger* might be low just after a creature has eaten but will gradually increase over time until the creature has its next meal. The creature will only act to satisfy the motivation *hunger* when the strength associated with *hunger* is sufficiently high. No creature will be sufficiently motivated to act to satisfy the *hunger* motivation at all times.

The value indicating the strength associated with each motivation, also known as the *motivational value*, will vary in relation to the following factors.

- The feedback of information from the agent's environment. For example, the strength associated with the motivation, *self-preservation* will normally be low. However, should the agent perceive immediate danger in some form, the strength associated with *self-preservation* will increase to a high level causing the agent to act in order to preserve its life. A direct relation can exist between what the agent perceives within its environment and the agent's motivational values.
- The internal workings of the agent. Some motivational values can be directly derived from the internal workings of an agent - for example the motivational values associated with the motivations *hunger*, and *rest*.

Motivations directly affect the generation of goals - by achieving a goal an agent is able to satisfy the motivations that led to the generation of that goal. If the motivational value associated with *hunger* is high, the goal, *have food*, might be generated. Once that goal is satisfied, the strength associated with *hunger* is reduced. In addition, the strength associated with the motivations of an agent is directly related to the relative *importance* of the various goals generated. If a motivation is high in strength any goal generated to satisfy that motivation will also be high in importance. In the warehouse domain (see chapter 1, section 1.1), a warehouse agent may have two motivations; *tidy* - a motivation which causes the agent to keep the warehouse tidy; and *make-profit* - a motivation which may cause the agent to satisfy as many customer order requests as possible. If, at a certain point, the motivational value associated with *make-profit* is higher than that associated with *tidy*, then goals concerned with satisfying customer orders will have a higher priority than goals concerned with keeping the warehouse tidy. Because motivations change in strength in response to changes in the environment, the priority or importance associated with goals that are generated to mitigate certain motivations may also change in value.

Motivations also enable an agent to evaluate the plans generated to achieve its vari-

ous goals. For example, if a human agent executes a plan which includes walking down a dark alley in order to achieve some goal, they might experience a small rise in their level of *fear*. By being able to predict the fact that walking down a dark alley will cause their level of *fear* to rise, the human agent might choose an alternative plan (for example one that involves driving to their destination) - a plan which will not cause their level of *fear* to rise - when evaluating the possible alternatives. Should one sequence of actions that achieve a goal conflict with the motivations of an agent (for example by causing the strength associated with a motivation such as *fear* to increase), the agent may choose an alternative solution.

In summary, an autonomous planning agent can be modelled as having a set of motivations which affect the planning process in the following ways.

1. They enable goals to be generated (the goals are generated in order to satisfy the agent's motivations).
2. They affect the *importance* of various goals - this enables the planner to prioritise goals.
3. They serve as a heuristic enabling the planner to determine the best partial plan for subsequent plan refinement. The partial plan containing the set of actions that best support the motivations of the agent is preferred (see section 2.2.6 "Evaluating partial plans" below).

In this thesis, we make the above assumptions concerning motivations - we do not address the process of updating motivations to reflect changes within the environment, nor do we address the problem of generating and prioritising goals in response to changes in the motivations. Such issues are addressed by [Norman 97] and are beyond the scope of this work.

2.2.4 Generating and updating goals

Within the planning/execution architecture (see figure 2.1) presented in this thesis, goals are generated by three distinct mechanisms.

1. The process "Generate/update goals" in figure 2.1 both creates and updates goals in response to the agent's current and predicted future motivational values and beliefs (beliefs are encapsulated within the partial plan and consist of the perceived current and future states, the goals the agent wishes to achieve and the actions the agent is

intending to execute).

2. The goal achievement component ("Plan to achieve goals" in figure 2.1) generates subgoals in order to satisfy the preconditions of actions within the partial plan.
3. The recovery component ("Recovery") specifies goals or subgoals which are to be reachieved in order to repair a partial plan when actual changes to the environment are not the same as the predicted changes to the environment.

In this section we briefly consider the first of these, goal generation in response to the agent's motivations and beliefs. An important feature of an autonomous agent is its ability to generate goals, and to do so for a variety of reasons. Such goals may be generated both to further the aims of the agent, to take advantage of opportunities that arise, or to prevent undesirable situations from occurring. The context of the agent (captured partly by modelling the motivations of that agent) is important in order to effectively generate goals. The following factors directly influence the generation of goals.

1. The current state of the environment.
2. The current strength of the agent's motivations.
3. Predicted future states of the environment.
4. The predicted future strength associated with the agent's motivations.

The agent's perception of its immediate environment may directly affect the strength associated with its motivations in such a way as to lead to the generation of goals. For example, the sudden, unexpected appearance of an angry lion may cause a sudden increase in a motivation concerned with self-preservation, which in turn may lead the agent to generate the goal of preserving its life. As well as being generated in response to the immediate environment, goals may be generated in response to future predicted states of the environment. For example, if a human agent plans to visit Canada to attend a conference, the prediction that they will be in Canada at some stage, together with a prediction of the future strength of the agent's motivations (this is directly related to the prediction that the agent will be in Canada) may cause such goals to be generated as to visit an old college friend or to view the Niagara Falls. These goals would probably not have been generated if the agent were not already planning to attend the conference. An agent may have a motivation to maintain the widget stock in a warehouse [Norman 97]. As widgets are sold, or when the quantity of widgets is below some threshold, the stock must be replenished by ordering new widgets from a supplier. However, if the agent

waited until the actual quantity of widgets was below this threshold before generating the goal to acquire more widgets, the time delay which might occur between ordering and receiving the widgets could be such that the warehouse might run out of widgets. The agent decides when to order new widgets on the basis of its beliefs about future states of the environment. If the agent plans to sell widgets, it can predict that in the future the quantity of widgets will be reduced. In addition, it can predict that the future strength associated with the motivation to maintain the widget stock will increase as a direct response to the predicted reduction in the quantity of widgets. On the basis of this prediction, the agent might generate the goal to replenish the widgets.

In some contexts it may be impossible to achieve certain goals. For example, whilst travelling by train, although it is possible to pursue such activities as reading or eating, it is impossible to pursue many other activities such as going to a bank or rowing a boat. It is vital the "Generate/update goal" component has knowledge of which goals may be achieved within which contexts.

In the previous section we discussed how changes in an agent's motivations might cause a change in the priority or importance of goals that were previously generated to mitigate those motivations. In addition to generating goals, the "Generate/update goals" component is responsible for updating existing goals in response to changes both in the agent's motivations and that occur within the environment.

This research is not concerned with the details of generating or updating goals. A detailed account concerning goal generation in response to the agent's motivations and beliefs can be found in [Norman 97] in which two distinct classes of goal generator are identified: D-Goals - goals which the agent has *decided* to generate as a consequence of deliberating about its beliefs; and R-Goals - goals that recur periodically, or at particular times of the day or week (known as replenishment goals).

For the purpose of this work, we assume that goals are generated in response to the motivations and beliefs of an agent and that such goals have an associated goal proposition, a deadline (generated through the agent's knowledge of the environment) and a value indicating the goal's importance (determined by the strength associated with the motivations which caused the goal to be generated). Goals thus generated are provided to the system described in this thesis. This demarcates the limits of the work, and the interface with the other components within the architecture. To facilitate further discussion, *goals* are produced by the "Generate/update goals" component (or goal generator), while

subgoals are generated by the goal achievement component ("Plan to achieve goal").

2.2.5 Choosing whether to plan or execute

In section 2.2.4 above we described how goals/subgoals are generated by the "Generate/ update goals" process (goals); the goal achievement component, "Plan to achieve goal" (subgoals); the recovery component, "Recovery". Once created, such goals/subgoals are added to a set of outstanding goals/subgoals belonging within the partial plan. In addition, during the goal achievement process ("Plan to achieve goal"), actions are selected and added to the partial plan. The planning architecture therefore requires a procedure which is responsible for choosing whether to achieve a goal/subgoal or whether to execute an action (see "Select goal or action" in figure 2.1).

Choosing whether to achieve a goal/subgoal or whether to execute an action involves determining whether any of the actions within the partial plan are ready for execution (an action may be executed if its preconditions are true within the current state and if it is first in the sequence of actions). If no actions are ready for execution the "Select goal or action" procedure decides which the "best" goal/subgoal is to achieve. This is determined by taking into account such factors as the importance and deadline associated with each outstanding goal or subgoal. If one or more actions are ready to be executed, the "Select goal or action" procedure decides (taking into account the importance and deadline associated with each goal, subgoal and executable action) the best goal, subgoal or executable action. As a general rule, goals, subgoals or executable actions of high importance and imminent deadlines are chosen in favour of goals, subgoals or executable actions of low importance and whose deadlines are far into the future. This procedure ("Select goal or action") is discussed in detail in chapter 4, section 4.5.

2.2.6 Planning to achieve goals

In this section we describe the main focus of this research, the goal achievement component, which corresponds to the process "Plan to achieve goal" in figure 2.1. An extension of the goal achievement process used in the nonlinear planner SNLP (see [McAllester & Rosenblitt 91]) is used which takes into account the fact that goals have deadlines and actions have durations. The goal achievement algorithm, illustrated in figure 2.2, takes the best outstanding goal or subgoal and achieves that goal/subgoal by selecting a new or existing action (a set of action templates is used to enable a new action to be selected)

and by posting new temporal and binding constraints (see "Achieve goal" in figure 2.2).

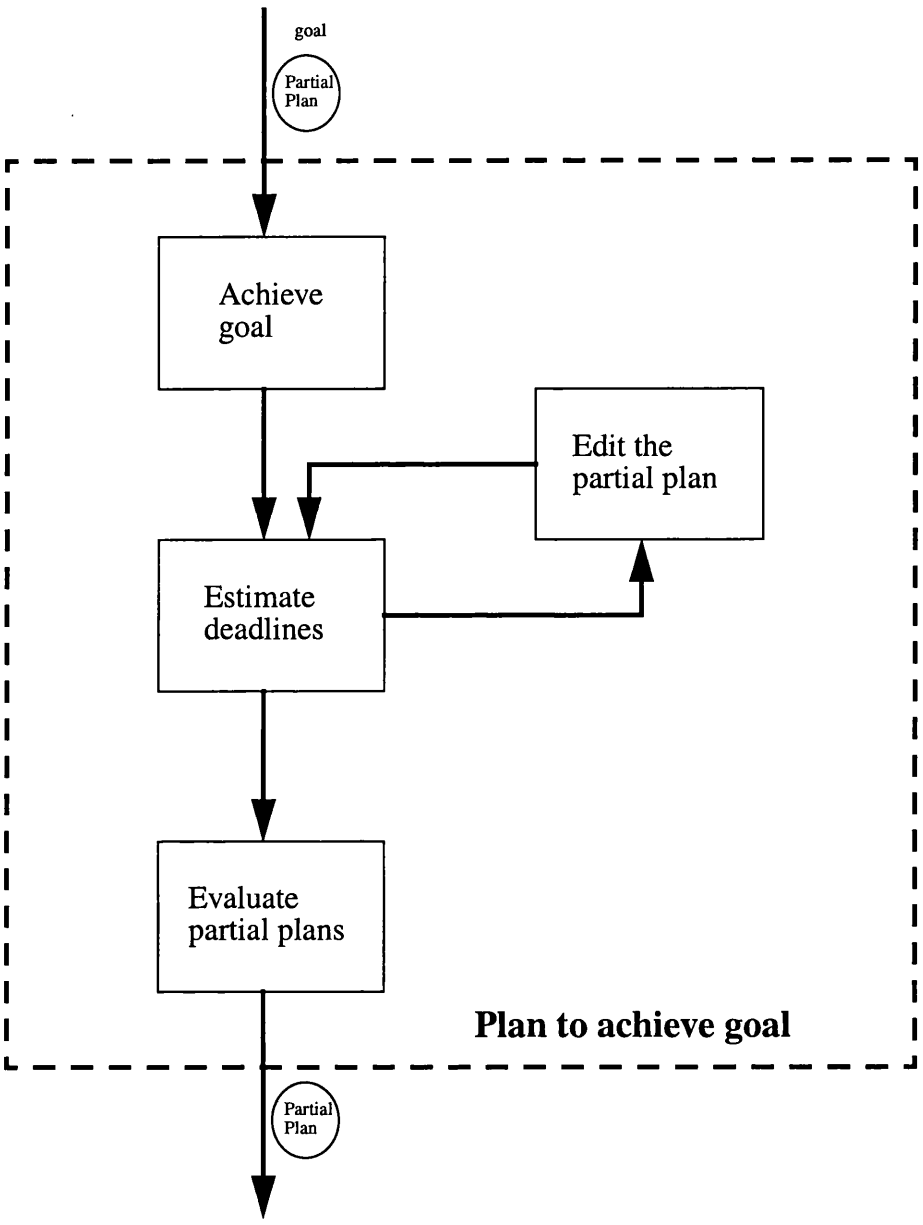


Figure 2.2 The Goal Achievement Process

In addition, further temporal and binding constraints may be posted to ensure that the achievement of the goal/subgoal does not conflict with goals/subgoals which have already been achieved (this process is known as conflict resolution). Once the goal/subgoal has been achieved a procedure is called which estimates the deadlines for each action and subgoal (subgoals are derived from the preconditions of actions) within the partial plan ("Estimate deadlines" in figure 2.2). If this process fails it means there is insufficient time available to achieve all of the goals within the partial plan. In this case, an editing process, "Edit the partial plan" removes goals together with their associated

actions and constraints from the partial plan. A number of alternative partial plans will have been generated by the processes described above. The motivations of the agent are therefore used as a heuristic to evaluate and rank each partial plan (see "Evaluate partial plan") prior to adding them to an ordered search space of partial plans (the "best" partial plan is the head of this ordered search space). In the following sections we describe each component in more detail.

Achieving a goal

Goals generated by the "Generate/update goal" component of figure 2.1 have an associated deadline and a value indicating their importance, while actions have duration. To take these factors into account, an extended version of the goal achievement algorithm used by the nonlinear planner SNLP is used. The nonlinear planning goal achievement algorithm, "Achieve goal", consists of two stages: selecting a new or selecting an existing action to achieve the goal (known as step addition and simple establishment respectively); and ensuring that there are no conflicts (known as conflict resolution). The algorithm therefore consists of three main processes.

1. Step addition - this involves creating a new action (by selecting and instantiating an action template) and adding that action together with new binding and temporal constraints to the partial plan. (The "Achieve goal" process therefore has access to a set of action templates which collectively represent the agent's execution capabilities.)
2. Simple establishment - new temporal and binding constraints are posted to ensure the goal or subgoal is achieved by an action currently within the partial plan.
3. Conflict resolution - new temporal and binding constraints are posted to ensure that the action selected to achieve the goal/subgoal does not conflict with goals/subgoals that have already been achieved and that actions currently within the partial plan do not conflict with the newly achieved goal/subgoal.

In order to reason about goals with deadlines and actions with duration the following extensions have been made to the goal achievement algorithm described above.

Estimating the duration of actions

To determine the latest time by which an action should be executed in order to achieve some goal by its deadline, it is necessary to estimate how long it will take the agent to execute that action. The duration of each action is estimated when the action is both created (step addition) and further instantiated (simple establishment). Durations are assigned to actions by using a look-up table which stores the estimated amount of time required to execute each action. This process is described further in chapter 5, section 5.2.2.

Assigning importance to actions

Each goal generated by the "Generate/update goals" component has an associated deadline and a value indicating its importance. When the goal achievement process selects a new action a_1 (using step addition) to achieve a newly generated goal g_1, a_1 assumes the same value of importance as that associated with g_1. In fact, all actions that contribute towards the achievement only of g_1 (for example the actions that are selected to achieve the preconditions of a_1) assume the same value of importance as that associated with g_1. However, if a_1 is later selected to achieve the newly generated goal g_2 (using simple establishment), it becomes more important, as executing a_1 contributes to the achievement of two goals, g_1 and g_2. The new value indicating the increased importance of a_2 is the sum of the values indicating the importance of the two goals g_1 and g_2. In addition, the importance value associated with all actions that contribute towards the achievement of the preconditions of a_1 also increases. A mechanism has been developed to propagate such changes in the value of importance associated with actions.

Estimating the deadlines of actions and subgoals

The planning/execution architecture presented in this thesis assumes that goals generated by the "Generate/update goals" component have deadlines and actions have duration. A temporal constraint manager capable of reasoning about actions with duration and goals with deadlines is required (such as [Vere 83], [Dean et al 87]).

An important part of planning to achieve goals with deadlines is to be able to determine whether or not those deadlines can be met. To achieve a goal by its deadline, there

must be sufficient time to both create a sequence of actions to achieve the goal and to execute that sequence of actions. This means that all actions which contribute towards the achievement of the goal must be executed by their own earlier deadlines. Such deadlines are determined by reasoning about the deadline associated with goals generated by the "Generate/update goals" component, the temporal constraints within the partial plan (which constrain the ordering of actions) as well as the duration associated with each action. An algorithm has been developed which estimates the deadlines associated with actions and subgoals (the preconditions of actions). The deadline of an action is defined to be the latest possible time by which execution of that action should commence in order to ensure that the goal to which the action contributes is achieved by its associated deadline. By implication, the preconditions of an action must be true by the action's deadline. This enables us to assign deadlines to subgoals as well as to actions within the partial plan. We require a mechanism that estimates the deadlines of actions and subgoals for the following reasons.

1. To enable the "Select goal or action" mechanism to select which goal/subgoal is to be achieved next or which action is to be executed.
2. To enable the planner to reason about whether or not there is sufficient time available to achieve all goals generated by the "Generate/update goals" component prior to their deadlines.

The algorithm developed for estimating the deadlines associated with actions and subgoals adopts a pessimistic approach (in contrast to DEVISER, [Vere 83]) - when actions are only partially ordered with respect to each other those actions are assigned the earliest possible deadline. For example, three actions a_1, a_2 and a_3, each with a duration of 3 minutes, remain unordered with respect to each other. These actions were selected to achieve preconditions of the action a_g with the deadline 5pm. The algorithm estimates the deadline for the actions a_1, a_2 and a_3, to be 4.51pm (i.e. 5pm - (3 mins + 3 mins + 3 mins)) because, in the absence of further information, any of the three actions could be executed first. This means that during the early stages of plan refinement, the deadlines estimated for actions and subgoals are likely to be too early.

In addition to estimating the deadlines associated with actions and subgoals, the algorithm indicates when there is insufficient time available to achieve all of the goals generated by the "Generate/update goals" component. The algorithm is able to determine both whether there is insufficient time available to achieve individual goals by their dead-

lines and whether there is insufficient time available to achieve all goals collectively by their deadline. When there is insufficient time available to achieve one or more of the goals generated by the "Generate/update goals" component, extra time is created by removing such goals from the partial plan (this is the responsibility of the plan editing process, "Edit the partial plan", described in the next section).

Editing partial plans

Plan editing is an essential part of the goal achievement process. The intention is to try to capture the way humans abandon the achievement of certain goals when there is insufficient time. If the deadline estimation process (described in the previous section) fails, it does so because there is insufficient time available to collectively achieve the goals generated by the goal generation component, "Generate/update goals".

When there is insufficient time to collectively achieve the goals generated by the "Generate/update goals" component, the plan editing process first removes a single goal together with its associated actions, temporal and binding constraints. The deadline estimation algorithm, "Estimate deadlines" is then invoked to reestimate the deadlines of the remaining actions and subgoals. If this fails (indicating there is still insufficient time to achieve all of the goals), the plan editing process, "Edit the partial plan", removes another goal (together with its associated actions, temporal and binding constraints). This cycle is repeated until the deadline estimation process successfully assigns deadlines to the remaining actions and subgoals.

Evaluating partial plans

When achieving a goal, the goal achievement algorithm, "Achieve goal", described above (see "Plan to achieve goal" illustrated in figure 2.2) generates a number of new partial plans. The "Estimate deadlines" process then assigns deadlines to actions and subgoals within each newly generated partial plan. If the "Estimate deadlines" process fails, the plan editing procedure, "Edit the partial plan", may then remove goals from each newly generated partial plan[2]. Once this process is complete each newly generated partial plan is added to a search space of partial plans. A partial plan evaluation procedure, "Evaluate partial plans", is required to rank each partial plan so that it is possible to select the best partial plan for further refinement.

---

2. Note: A filtering mechanism is used to ensure that duplicate partial plans are not generated as a consequence of the plan editing process.

The heuristic employed by the "Evaluate partial plans" process favours plans which maximise the number of achieved goals, which contain actions of short duration, and which contain actions that best support the agent's motivations. The action representation has been extended so that each action has two sets known as *pros* and *cons* which contain values that are used to provide a crude prediction of the degree to which the action, when executed, will support or undermine the agent's motivations. In order to determine the degree to which a partial plan will support or undermine the agent's motivations, the "Evaluate partial plans" process examines the degree to which the set of motivations will be supported or undermined by each action within that plan following execution. For example, a human agent needs to satisfy the goal of being in Edinburgh by 2pm. The options available for achieving this goal include actions that enable the agent to travel by aeroplane, travel by train, travel by car or travel by coach. The agent has the motivations *save money* and *feel alert.* Travelling by aeroplane undermines the motivation *save money*, travelling by train supports both *save money* and *feel alert*, while travelling by car or travelling by coach undermines the motivation *feel alert.* The plan which best supports the agent's motivations might be one which includes the action of travelling by train. Each partial plan is thus evaluated to determine the extent to which it will support or undermine the motivations of the agent.

When a new action is both created (step addition) and further instantiated (simple establishment), the *pros* and *cons* fields associated with that action need to be instantiated. This is currently achieved by using domain knowledge represented in the form of a look-up table - this is a similar process to that used in estimating the duration of actions (see section "Estimating the duration of actions" above). For example, to achieve the goal of having eaten in a restaurant the action *(eat-at ?x)* is selected where *?x* is a variable. The values assigned to the fields *pros* and *cons* associated with the action will depend upon how *?x* is instantiated (this is explained in more detail in chapter 5, section 5.2.3).

Executing actions

When an action is executed, the execution component is responsible for updating the partial plan (see "Execute action" in figure 2.1). An action may only be executed if it is fully instantiated, its preconditions are true, and it is possibly first within the partial order of actions. When the action is executed a number of changes occur which require the partial

plan to be updated.

1. The "initial world model" within the partial plan must be updated to reflect the fact that by executing the action, the agent has made changes to the environment. In addition, the "initial world model" is also updated to reflect any changes that are made to the environment by the activities of other agents or by physical processes. This reflects the fact that in real world environments, the environment may change unpredictably. Such changes will directly affect the motivations of the agent which in turn might lead to the generation of new goals.
2. The action which has been executed, together with its associated temporal and binding constraints must be removed from the partial plan.
3. All persistence constraints that maintain the truth of the action's preconditions can be removed from the partial plan. (Persistence constraints represent goals/subgoals which have been achieved.)
4. The search space of previously generated partial plans must be deleted. When the "Plan to achieve goal" process achieves a goal, new partial plans are generated and added to a search space of partial plans. Each partial plan contains the same "initial world model" representing the current state of the environment. However, each partial plan contains a different set of actions which reflects the fact that there are alternative ways of achieving goals. While the environment remains unchanged (i.e. during cycles when the "Select goal or action" process chooses to achieve a goal/subgoal as opposed to execute an action), the "Plan to achieve goal" process achieves a goal/subgoal, each time increasing the search space of partial plans. However, when an action is executed the world changes, which means the "initial world model" of previously generated partial plans no longer corresponds to the actual state of the environment. It is no longer possible for the "Evaluate partial plans" process to backtrack or select such previously generated partial plans. Once an action has been executed, all previously generated partial plans must be deleted (with the exception of the current partial plan). To continue planning, a new search space of partial plans containing the new "initial world model" must be generated.

Updating the world model

The planning/execution architecture illustrated in figure 2.1 makes the assumption that changes do not occur to the environment whilst the agent is planning to achieve goals or subgoals. This is a simplification as it is possible that changes occur within the environment (due to the activities of other agents and physical processes) whilst the agent is planning. A part of the "Execute action" component is responsible for initiating the appropriate sensing activities in order that the agent can update its world model to reflect changes that have been made to the environment, as a consequence of the agent executing an action, other agents acting and physical processes occurring within the environment. (Because we do not model sensors in our implementation of the planning/ execution architecture, we capture all sensing activities by asking a user to input a description of the state of the environment as well as the time following execution.) The intention is to emulate what happens when humans plan - whilst planning humans may assume that their world model is correct even though other agents or physical processes may have caused changes in the environment. When executing, or carrying out plans, humans monitor the environment and update their plans accordingly if adverse changes have occurred.

Recovering from unexpected situations

Once an action has been executed, the actual state of the environment may differ to what was predicted. This may be due to the following reasons.

1. Executing the action may not produce the intended outcome (for example an agent, whilst picking up a block may accidentally drop that block).
2. The environment may change in unforeseen ways due to the activities of other agents.

Unexpected and unforeseen changes in the environment can affect the validity of the partial plan. For example such changes may violate some of the persistence constraints within the partial plan (persistence constraints ensure that goals or subgoals remain true over a period of time) which means that some of the preconditions of later actions may no longer hold. A persistence constraint may be violated in one of two ways. Firstly, the action may have been selected in order to achieve some goal (for example to satisfy the precondition of some later action) but when that action is executed, the goal proposition

(an effect of the action) does not become true. Secondly, the action may unexpectedly deny or undo the goal proposition of a persistence constraint. For example, an action a_1 is selected to achieve the precondition g_1 associated with the action a_2, a_1 is executed and achieves g_1. However, when the action a_3 is executed (following a_1 but prior to a_2), it unexpectedly undoes or denies the proposition g_1.

When a persistence constraint is violated due to unexpected and unforeseen changes within the environment, the recovery component adds the goal proposition associated with the constraint to the set of outstanding goals so that it can be reachieved by the "Plan to achieve goal" process[3].

The recovery component is also required if the time taken to execute an action is longer than expected. Because the action was selected to contribute towards the achievement of some goal, it may be difficult or impossible to achieve that goal by its deadline. In such cases the recovery component may decide to abandon the achievement of the goal. When execution takes longer than expected, the "Estimate deadlines" process is called to determine whether or not it is still possible to achieve all goals by their deadlines. If not, the "Edit the partial plan" procedure is then required to remove any unachievable goals, together with their associated actions, temporal and binding constraints, from the partial plan.

2.3 Summary

The architecture of a planning/execution system to be used by an autonomous agent has been described in this chapter. The architecture makes an important contribution to planning research in that it addresses the idea of continual planning (i.e. where goals are continually generated while the agent acts within an environment), which requires the interleaving of planning and execution, as well as the idea that context (partly captured by modelling the motivations of the agent) plays an important role in enabling the agent to choose between alternative partial plans.

This chapter presents and summarises the components of the architecture. In the following chapters we describe in detail each of the components.

3. If a precondition of an action is no longer true because some persistence constraint has been violated, it may not always be efficient simply to reachieve that precondition. For example, the three actions, a_1, a_2 and a_3 are selected in order to achieve the goal g_1 generated by the goal generator. When a_1 is executed, the persistence constraint protecting the subgoal sg_3 is violated where sg_3 is a precondition of the action a_2. Instead of simply replanning to achieve sg_3 it may be more efficient to replan to achieve the goal g_1.

Chapter 3

The Control of the System

3.1 Introduction

The planning/execution architecture described in this thesis was illustrated in chapter 2, figure 2.1 - in this chapter we discuss some of the issues that led to the decision to adopt this architecture. In particular we begin by describing which factors cause both the agent's motivations to change in strength and goals to be generated/updated. We then discuss how frequently it is necessary to update the motivations and generate/update goals and present an alternative control strategy, describing why we decided to adopt the architecture presented in chapter 2, figure 2.1. The issue of how much planning is undertaken by the "Plan to achieve goal" process each time it is called is discussed and two alternative control strategies for this component are described. The algorithm used to implement the control strategy of the planning/execution architecture is then outlined. Finally, we present two other planning systems in which planning and execution are interleaved, and compare these with the work presented in this thesis.

3.2 The Control of the System

3.2.1 What causes motivations to change?

In chapter 2, section 2.2.3, we discussed how the basic needs, desires and preferences of an agent may be represented by modelling the motivations of that agent. Associated with each motivation is a measure of strength, or motivational value, which varies with changing circumstances. In this section we discuss which factors lead to a change in the strength associated with an agent's motivations.

The motivations of an agent change in response to the following changes in an agent's beliefs.

1. The agent perceives changes have occurred within itself (i.e. changes in its physical state) and/or within its environment. These changes may have been brought about by the agent executing an action, by the activities of other agents, or by physical processes occurring within the agent/environment. Such changes are reflected by updating the agent's model of the environment.
2. Changes occur in what the agent believes will happen in the future. Such changes encompass beliefs about which actions the agent intends to execute; the goals the agent believes it will achieve by executing those actions; the goals the agent would like to achieve (the set of outstanding goals). Once an action has been selected to achieve a goal (by the component "Plan to achieve goal") the agent believes that after it has executed that action the goal will become true at some point in the future. This constitutes a new belief - the agent believes this in addition to what it believed prior to planning to achieve that goal.

The main factor affecting the strength associated with an agent's motivations is achieving goals that were generated in order to mitigate some motivation(s). Goals are achieved by *executing* some set of actions. For example, if the strength associated with the motivation *hunger* is high, a goal to have eaten some food is generated. A plan (or sequence of actions) which achieves this goal is then created. Once this plan has been executed, the goal is achieved and so the strength associated with *hunger* drops. Likewise, if the strength associated with a warehouse agent's motivation *tidy* (this motivates the warehouse agent to tidy the warehouse by disposing of commodities that are beyond their sell-by date) is high, a goal to tidy the warehouse is generated. Again, once the warehouse is tidy, the strength associated with the motivation *tidy* drops. Satisfying a goal which was generated in order to mitigate certain motivations leads to a decrease in the strength associated with those motivations. Such goals may not necessarily be satisfied as a consequence of the agent executing some action, however. For example, the warehouse may be tidied by another agent. As long as such goals are achieved (either by the agent or opportunistically by some other agent), the strength associated with the motivations that led to the generation of such goals drops.

Changes to an agent's motivations are also brought about by changes occurring within the environment - such changes may be brought about directly by the agent through executing actions. All actions, when executed, will support or undermine a subset of the agent's motivations to some degree. The plan to achieve the goal of having

eaten some food may contain the action of purchasing food - executing this action will undermine a motivation concerned with obtaining or with saving money. The plan may also contain an action concerned with walking down a dark alley - executing such an action may undermine or cause an increase in the strength associated with a motivation concerned with self-preservation. Actions which were selected as part of a plan to achieve some goal may actually conflict with the motivation(s) that caused the goal to be generated. For example, through executing some of the actions that were selected in order to mitigate the motivation *hunger*, the agent may make itself more hungry (for example if the agent has to walk a long way in order to acquire food). Because each action supports or undermines a subset of the agent's motivation to some degree, it is possible, given a plan containing actions, to crudely predict the future state (i.e. strength) of the agent's motivations. This facility is used as part of a heuristic to evaluate and rank partial plans and is addressed in detail in chapter 6, section 6.4. In addition, changes to the environment caused by other agents or physical processes may support or undermine the agent's motivations. The sudden, unexpected appearance of an angry lion may lead to a sudden increase in a motivation concerned with self-preservation, for example. Being given a large sum of money will lead to a decrease in a motivation concerned with obtaining or with saving money. In the warehouse domain described in chapter 1, section 1.1, perceiving that a number of commodities are beyond their sell-by date increases the strength associated with the motivation to keep the warehouse tidy.

Finally, the strength associated with certain motivations may change in response to changes in the agent's internal state, i.e. in response to changes made to a partial plan by planning to achieve some goal. For example, the strength associated with a motivation concerned with keeping the agent as busy as possible might change as a consequence of adding a new action to the partial plan in order to achieve some goal - the motivational value will decrease because the agent will have one extra action to execute which means it will be busier.

The changes outlined above directly affect the motivations of the agent by causing changes in the strength associated with each motivation. The agent's motivations should change once during each plan/execute cycle, either in response to changes in the agent's plan (as a consequence of planning to achieve some goal) or in response to changes that have occurred within the environment (as a consequence of the agent executing actions or due to the activities of other agents or physical processes). In turn, such changes in the

agent's motivations may cause goals to be generated. In the following sections we consider how the mechanism responsible for determining changes in an agent's motivations, "Update motivations" (see chapter 2, figure 2.1), fits within our planning/execution architecture.

3.2.2 Generating/updating goals

In chapter 2, section 2.2.4, we described how goals are generated in response to the current state of the environment, the current strength of the agent's motivations (which are directly influenced by the current state of the environment), predicted future states of the environment, and the predicted future strength associated with the agent's motivations (again, this is directly influenced by predicted future states of the environment). In addition, we also saw that changes in the agent's motivations directly affect the priority or importance of goals that were generated to mitigate those motivations. The process of generating and updating goals ("Generate/update goals") should occur once every plan/execute cycle for the following reasons.

- Goals might be generated or their priority updated in response to changes in the agent's motivations. In the previous section we saw that the agent's motivations change either in response to changes in the agent's plan (as a consequence of planning to achieve some goal) or in response to changes that have occurred within the environment (as a consequence of the agent executing some action, or due to the activities of other agents or physical processes). If the motivations change once every plan/execute cycle, then new goals might be generated or changes might occur to existing goals once every plan/execute cycle.
- Goals might be generated in response to changes in the agent's plan as a consequence of planning to achieve some goal. If an agent changes a plan it means it also changes what it predicts will happen in the future (i.e. changes will occur in predicted future states both of the environment and of motivational values). This implies that new goals might be generated or existing goals updated each time a goal is achieved through planning.

3.2.3 When to update the motivations and generate/update goals

In sections 3.2.1 and 3.2.2 above we described the various factors that cause motivations to change and goals to be generated/updated. To reflect any changes that occur to both

the plan (as a consequence of planning to achieve some goal or subgoal), as well as to the environment (as a consequence of executing some action, the activities of other agents and physical processes), it was seen that motivations should be updated and goals generated/updated once every plan/execute cycle. This suggests that the overall control of the system should be that of figure 3.1 below.

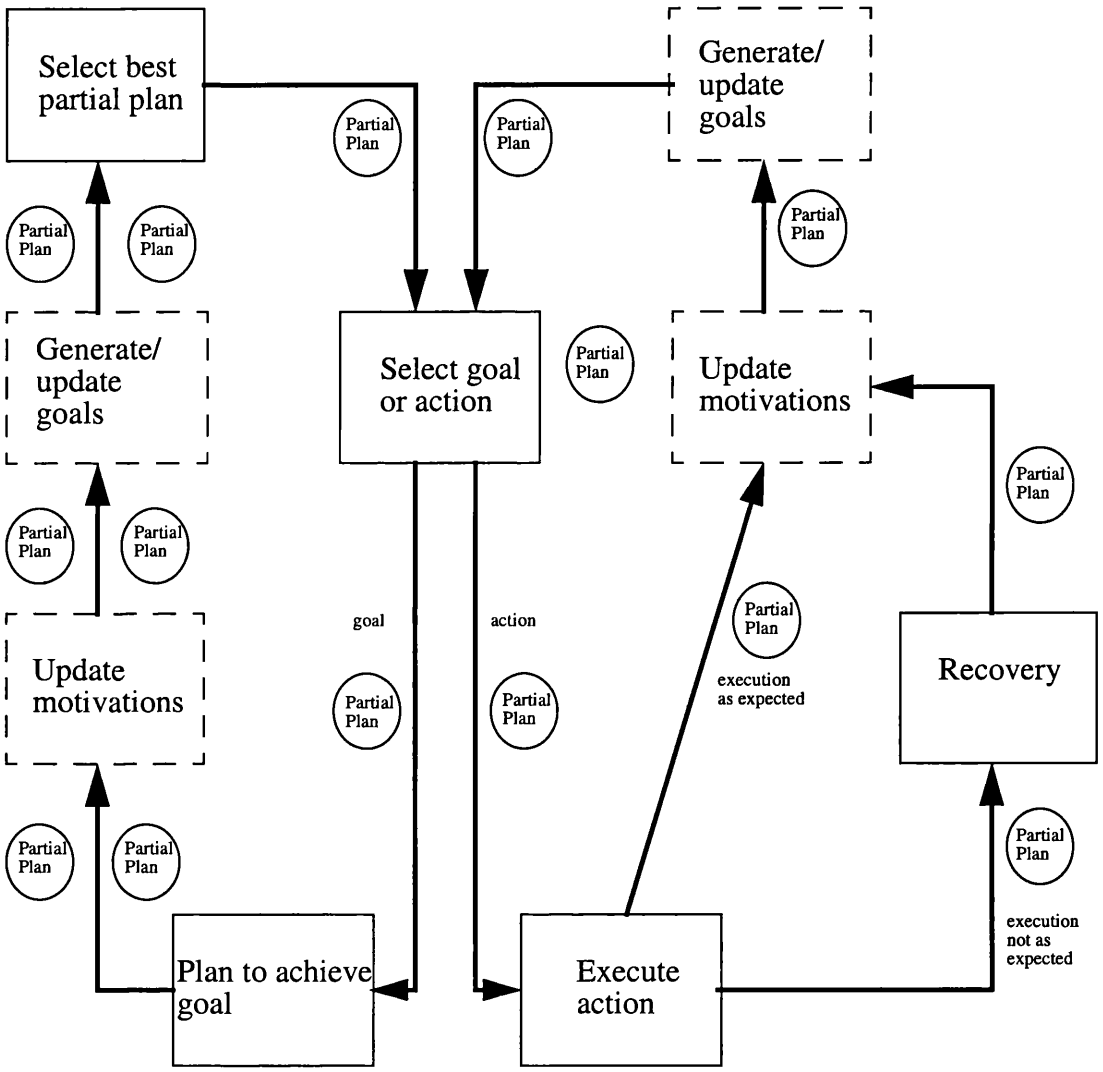


Figure 3.1 An Alternative Control Strategy

In figure 3.1, when the "Select goal or action" component decides to achieve a goal/ subgoal, a partial plan is passed to a nonlinear goal achievement component ("Plan to achieve goal"). One or more new partial plans are generated in order to achieve the goal/ subgoal - each new partial plan is generated by selecting a new or existing action (using a set of operator schemas/action templates to represent the agent's capabilities which are not shown in the diagram) to achieve the goal/subgoal when executed, and by posting various constraints. Once a new partial plan has been generated, the agent's beliefs about

the future will have changed which will cause changes to occur in the agent's motivations. The motivations therefore need to be updated ("Update motivations") to reflect these changes. In addition, such changes in the agent's beliefs and motivations means that goals might be generated and/or updated ("Generate/update goals"). More than one new partial plan may be created by the "Plan to achieve goal" component as there may be more than one way of achieving the goal/subgoal. The newly updated motivations and newly updated set of outstanding goals are used as part of a heuristic to evaluate and rank each newly generated partial plan. The best partial plan is chosen from the search space of partial plans for subsequent refinement - this partial plan is passed back to the "Select goal or action" component which again determines whether to achieve one of the goals within the partial plan or whether to execute an action.

When a decision is made to execute an action, the flow of control is the same as that in chapter 2, figure 2.1.

The advantages of the architecture illustrated in figure 3.1 are as follows.

1. Motivations are updated to reflect changes that are made to the partial plan as a consequence of planning to achieve some goal or subgoal. This is in addition to updating motivations to reflect changes that have occurred within the environment.
2. Goals are generated/updated to reflect changes that are made to the motivations as well as to partial plans as a consequence of planning to achieve some goal or subgoal (this is in response to changes in predicted future states of the environment and motivations).
3. Newly generated partial plans that are generated as a consequence of planning to achieve some goal or subgoal, are evaluated with respect to the newly updated motivations and newly generated or updated goals.

It could therefore be argued that figure 3.1 is a better control strategy than chapter 2, figure 2.1. However, the control system shown in figure 3.1 has several disadvantages.

The main disadvantage of this approach is that motivations have to be updated, and goals generated/updated for each newly generated partial plan thereby increasing the computational load. A part of this requirement is that each plan must have an associated set of motivations - if motivations change to reflect the current plan, then the motivations associated with each newly generated partial plan will differ because each partial plan is different. One way to capture changing motivations is to incorporate them within the partial plan representation.

```
plan == (motivations, actions, timings, bindings, unachieved_goals,
pers_constraints)
```

In this representation, each partial plan in the search space of partial plans has an associated set of motivations whose current strength directly reflects the contents of the partial plan - i.e. the initial world model (representing what the agent perceives to be currently true in the world), the set of unachieved goals (goals the agent will plan to achieve), persistence constraints (representing the goals the agent has planned to achieve) and the set of actions (the actions the agent intends to execute). The motivations associated with a plan are a particular strength because the agent believes it will achieve the goals in that plan by executing the actions in that plan and that it intends to achieve the set of outstanding goals in that plan. This representation enables the motivations to be updated once during each plan/execute cycle to reflect the changes made within the partial plan as a consequence of planning to achieve some goal. In turn, the changes in motivations directly determine the goals that will be generated by the "Generate/update goals" component. Once changes to the motivations have been made, new goals may be generated - because the motivations differ within each partial plan, goals may be generated/updated differently within each partial plan. In addition, once goals have been generated/updated, the partial plan is evaluated/ranked with respect to its associated set of motivations - because each partial plan has a different associated set of motivations, the ranking of each partial plan is done with respect to different sets of motivations.

It could also be argued that the changes made to a partial plan as a consequence of planning to achieve some goal/subgoal are too minor to significantly affect both the strength associated with motivations and the generation/update of goals. It might not be worthwhile adopting this strategy in view of the required increased computational load.

A Separate set of Motivations

An alternative way to capture changing motivations is to have a single separate set of motivations (as opposed to one set of motivations per partial plan). With this approach it is no longer possible to capture the idea of motivations changing solely in response to changes in the agent's beliefs (i.e. as a consequence of planning to achieve some goal/subgoal). For example, when the "Plan to achieve goal" process plans to achieve a goal by selecting an action and posting new constraints, several new partial plans are gener-

ated, one for each action capable of achieving the goal. Each new partial plan encapsulates a different belief as to the way that goal might be achieved. If there is only a single set of motivations, this set cannot be simultaneously updated in different ways to reflect the alternative ways there are of achieving the goal.

In order to overcome this problem, the algorithm illustrated in chapter 2, figure 2.1, can be used. Instead of the set of motivations being updated each time a goal/subgoal is achieved, the set of motivations is only updated once the agent has executed an action. When an action is executed, the current partial plan's "initial world model" is updated to reflect all changes that have occurred within the environment (changes brought about as a consequence of execution, the activities of other agents, and physical processes) and all other partial plans in the search space are deleted as their "initial world model" no longer corresponds to the newly perceived state of the environment. If the agent's motivations are only updated after the agent has executed some action, then, while the agent is not acting (i.e. during cycles when the agent is planning to achieve goals/subgoals) the motivations remain unchanged. This means all partial plans in the search space are evaluated with respect to the same set of motivations (unlike the approach described in figure 3.1 above). When the motivations are updated, not only are they updated to take into account the perceived changes within the environment, they are also updated to take into account the changes in the agent's beliefs (i.e. a new belief represents the way a particular goal/subgoal is achieved - a number of goals/subgoals may have been achieved prior to the agent executing some action which means that the agent may have a number of new beliefs since the motivations were last updated). Once the motivations have been updated, goals are generated/updated in order to reflect the changes to the motivations as well as the changes to the agent's plan. The advantage of this approach is that the set of motivations is updated and goals are generated/updated less (i.e. only once the agent has executed some action during the execute cycle, and only in response to changes in one partial plan), leading to a reduced computational load. In addition, only a single set of motivations is required which remains independent of the partial plan representation. Disadvantages include the fact that motivations no longer accurately reflect changes in the agent's beliefs (i.e. the more goals achieved prior to executing an action, the less accurately the motivations reflect the agent's current beliefs). This algorithm can be seen in chapter 2, figure 2.1, and is the approach adopted as the control strategy for the planning/execution architecture described in this thesis.

3.3 Interleaving Planning and Execution

Another issue that arose when designing the control strategy for the planning/execution system, was how much planning should occur during each planning cycle - i.e. how much planning should the "Plan to achieve goal" component undertake once the "Select goal or action" component of chapter 2, figure 2.1 chooses to plan to achieve a goal? Two alternative control structures were devised for the "Plan to achieve goal" component.

1. The "Select goal or action" component chooses either to plan to achieve a goal (i.e. a newly generated or updated goal created by the "Generate/update goals" component), or to execute an action. If a goal is chosen, the "Plan to achieve goal" component generates a complete plan which achieves that goal (i.e. not only does the "Plan to achieve goal" component achieve the goal, it achieves all subgoals/open conditions that are created as part of the planning process - planning is complete when there are no outstanding subgoals/open conditions and when all conflicts have been resolved). As part of this process, the "Estimate deadlines" procedure assigns deadlines to actions belonging within each partial plan generated, and, if there is insufficient time available to achieve all of the goals in a plan (i.e. the "Estimate deadlines" process fails to assign deadlines to actions), that plan may be edited (by the "Edit the partial plan" process). Alternatively, once a complete plan has been generated, deadlines are estimated for each action in that plan, and, if there is insufficient time available to achieve all of the goals within that plan, either the original plan is returned or the plan may be edited.
2. The "Select goal or action" component chooses either to plan to achieve a goal/subgoal (i.e. a newly generated/updated goal created by the "Generate/update goals" component or a subgoal/open condition created by the "Plan to achieve goal" component), or whether to execute an action. If a goal or subgoal is chosen, the "Plan to achieve goal" component performs only one cycle of goal achievement (i.e. the "Plan to achieve goal" component achieves only the selected goal/subgoal by choosing a new or existing action and by posting constraints to resolve conflicts). More than one new partial plan may be generated in which the goal or subgoal has been achieved. Deadlines are estimated for the actions and subgoals within each newly generated plan, and, if there is insufficient time available to achieve all goals within a plan, that plan may be edited. Finally, each newly generated partial plan is

evaluated/ranked and added/merged with the ordered search space of partial plans.

The first strategy makes fewer calls to the "Plan to achieve goal" procedure. However, each time the "Plan to achieve goal" process is called, more planning is undertaken (which requires more time) as a complete plan is generated to achieve the goal. The main drawback with this approach is that the planning/execution architecture is unable to change which goal it is focussing upon whilst planning. For example, the "Generate/ update goals" component generates two new goals, g_1 and g_2. If the "Select goal or action" procedure decides that the goal g_1 should be achieved, the "Plan to achieve goal" component creates a complete plan to achieve g_1 leaving the goal g_2 to be achieved at a later stage (i.e. during a later cycle, when the "Select goal or action" decides that the goal g_2 should be achieved). This strategy does not allow the agent to change its focus of attention whilst planning - for example, whilst planning to achieve g_1, the agent is unable to switch its attention to plan to achieve g_2 instead. (Whilst planning to achieve g_1, the importance associated with g_2 may increase, which may cause the agent to change its focus of attention.) This strategy does not require a global search space of partial plans as the search space is generated only during the planning process - once a complete plan is generated to achieve a goal, the search space is no longer required. The advantage of this strategy is that optimal plans are generated to achieve each goal individually which means it performs well on problems in which goals interact (such as Sussman's anomaly).

In contrast, the second strategy performs much less planning each time the "Plan to achieve goal" process is called, but, as a consequence, this process is called much more frequently. The main advantage of this strategy is that it enables the agent to change its focus of attention by changing which goal it is focussing upon at any particular time. This resembles the ability humans have of changing their focus of attention to deal with more important or urgent tasks, and of changing which plan they currently prefer. The disadvantage of this strategy is that it does not produce good plans when goals interact with each other (i.e. it cannot produce an optimal plan to solve Sussman's anomaly).

Both strategies have been implemented as alternative control strategies of the "Plan to achieve goal" component which is described in more detail in chapters 5 and 6.

3.4 The Control Implementation

In this section we describe the algorithm used to determine the flow of control between

components of the planning/execution architecture shown in chapter 2, figure 2.1. A sin-

Table 3.1 The planning/execution architecture

1. Choose the best partial plan (this is the plan at the head of the search space of partial plans which contains a set of *(plan, value)* tuples ordered on the basis of *value*).
2. Select whether to achieve a goal/subgoal or execute an action. This process involves determining which actions are both first (i.e. in the partial order of actions) and ready to execute (i.e. their preconditions must be true in the current state). Outstanding goals/ subgoals and actions that are both first and ready to be executed are evaluated on the basis of their importance, deadlines and the amount of effort currently expended in achieving their associated goals. This algorithm is described in further detail in chapter 4, section 4.5.
   (a) If a decision is made to achieve a goal/subgoal, new partial plans are generated using the following procedures: a new or existing action is chosen and constraints are posted to achieve the goal/subgoal - more than one new partial plan is generated; the deadlines associated with actions in each newly generated partial plan are estimated; each newly generated partial plan may be edited (if the deadline estimation process fails); each newly generated partial plan is evaluated/ranked with respect to the agent's motivations. Finally, the newly generated partial plans are sorted and then merged based on their ranking with the (ordered) search space of partial plans. This algorithm was discussed in section 3.3 above and is described in further detail in chapters 5 and 6.
   (b) Else, if it is decided to execute an action the following procedures are invoked. These are described in further detail in chapter 7.
      i. The user is asked to key in the actual outcome following execution - both the state of the environment and the time. The plan is then updated to reflect the outcome - this includes monitoring the post execution state of the environment to determine whether any achieved goals/subgoals (i.e. persistence constraints) have been clobbered/undone (this may happen if execution does not result in the expected outcome). When execution does not go as expected, the clobbered goals/subgoals are converted into goals/subgoals to be reachieved which are added to the newly updated partial plan.
      ii. If execution takes longer than expected (i.e. the actual time following execution is later than the predicted time), the deadlines associated with actions in the partial plan are reestimated and, if there is insufficient time available, the partial plan may be edited.
      iii. The motivations are updated to reflect the changes in the partial plan (brought about as a consequence of planning) as well as changes to the world that are brought about as a consequences of execution, of the activities of other agents, and of physical processes (in this implementation, the user is asked to key in the new strength associated with each motivation).
      iv. New goals are generated and existing goals updated (in this implementation the user is asked to key in new goals and update existing goals).
      v. All previous partial plans in the search space are deleted as their "initial world" model no longer corresponds to the current (i.e. post execution) state of the world. The newly updated partial plan (updated as a consequence of execution) is evaluated with respect to the (newly updated) motivations, and forms the sole plan in the search space.

gle set of motivations is used (see section 3.2.3) - the motivations are updated and new goals are generated and/or existing goals updated only after an action has been executed. This algorithm has been used for the following reasons.

1. It is assumed that changes made to a partial plan as a consequence of planning to achieve some goal/subgoal will be too small to significantly affect both the strength associated with motivations and the generation/update of goals. Motivations are therefore updated and goals generated/updated only after an action has been executed.
2. Less computation is required as a consequence - motivations are only updated and new goals are generated only once an action has been executed instead of once for each newly generated partial plan per plan/execute cycle.
3. A single set of motivations is used which means less memory is required.

The algorithm is illustrated in chapter 2, figure 2.1 and is described in table 3.1.

3.5 Other Related Work

In this section we examine two planning systems, Sage ([Knoblock 95], [Knoblock 96]) and the HSTS planner used as part of the Remote Agent architecture [Muscettola et al 98], which interleave planning and execution.

3.5.1 Sage

The most closely related work is the planner Sage ([Knoblock 95], [Knoblock 96]) - an extension of the partial-order planner UCPOP [Penberthy & Weld 92], which supports simultaneous action execution and which tightly integrates planning and execution. Sage served as the underlying query-planning component for early versions of the SIMS information mediator - the overall goal of the SIMS project was to provide integrated access to information distributed over multiple, heterogeneous sources such as databases, knowledge bases, flat files, Web pages and programs [Ambite et al 97]. In Sage, UCPOP has been modified to include two extra flaws (in addition to open conditions and threats): an unexecuted action flaw and an executing action flaw. The Sage algorithm, which is illustrated in table 3.2 below, is designed so that execution only occurs if there are no threats or open conditions (i.e. once a complete plan has been generated).

The planner starts with an initial plan, where the goals are the open conditions - initially, the set of current plans (i.e. the search space) contains only this initial plan. The planner repeats the algorithm until it produces a plan in which every action has been executed. In each cycle, the planner selects a plan from the search space of current plans.

Whenever an action is executed, an action terminates, or a new goal is added, the set of current plans is replaced by a new set containing only this plan. The first two conditions in this algorithm ensure that the planner finds a plan with no open conditions or threats before it commits to a plan and initiates any actions.

This algorithm supports simultaneous planning and execution (unlike the work presented in this thesis). Before the system initiates execution of any action, it constructs an initially complete plan. However, once execution starts, an action could fail, a new goal could arise, or the system may require additional information (sensing) to continue planning. In any of these cases, once the new open condition has been added to the list of flaws, the system can augment the executing plan to achieve these conditions while it continues executing any actions that have already been initiated.

Table 3.2 Sage - algorithm for planning and execution

Remove a plan from the set of *current plans* and apply the first applicable condition:

1. If there are any threats, these are resolved by adding additional constraints to the plan. The possible refinements are added to the set *current plans* (i.e. this contains the search space of partial plans).
2. If there are any open conditions, additional actions or ordering links are added to achieve them. The possible refinements are added to the set *current plans*.
3. If any executing actions have completed.
   (a) If an action is executed successfully the results are recorded and the plan is updated. The set *current plans* is replaced with the newly updated plan.
   (b) If the action fails, the failed portion of the plan is removed, the model is updated and the set *current plans* is replaced with this new plan.
4. If there are any new goals to solve, they are added to the open conditions and the set *current plans* is replaced with the (newly updated) plan.
5. If any unexecuted actions are now executable, create a process to execute them and replace the set *current plans* with the plan.

Sage also handles action failures and replanning - the planner aims to replan the failed portion of the plan, while maintaining as much of the executing plan as possible. This is currently supported by requiring the designer of a domain to define a set of domain-specific handlers. When a failure occurs, the failure handler is called with the action that failed and the type of failure, and the failure handler is expected to remove the failed portion of the plan and update the model to avoid the same failure when the failed actions are replanned.

Like the work presented in this thesis, Sage can handle new goals whilst it is in the midst of executing a plan that achieves some other goal(s). However, when a new goal is presented to Sage (by a user), a complete plan is generated which achieves that goal

before execution can continue (in the same way as was described in the first strategy of section 3.3 above). Sage is unable to change its focus of attention from one goal to another whilst planning to achieve those goals. The domain description language used by Sage is more expressive than that used in this thesis, as it is based on UCPOP which supports ADL [Pednault 89]. Sage does not reason about time and so cannot plan to achieve goals with an associated deadline. Likewise, Sage does not provide a domain-independent mechanism that enables it to take into account its context. Finally, unlike the work presented in this thesis, Sage incorporates sensing actions by supporting run-time variables (an idea proposed in earlier work on sensing in planning, [Ambros-Ingerson 87], [Etzioni et al 92]) which allow it to reason about the sensed information.

3.5.2 The Remote Agent architecture

The NMRA (New Millenium Remote Agent - [Pell et al 96a], [Pell et al 96b], [Muscettola et al 98]) architecture is the first AI system to control an actual spacecraft and the first system to integrate closed-loop planning and execution of concurrent temporal plans. The key technology demonstrated is spacecraft autonomy, including onboard planning and plan execution - the spacecraft spends long periods of time without the possibility of communication with ground operations staff, and plans when it needs to communicate back to Earth. It must maintain its safety and achieve high-level goals, when possible, even in the presence of hardware faults and other unexpected events. The spacecraft domain presents many challenges for planning and execution.

1. Many devices and systems must be controlled, leading to multiple threads of complex activity - these concurrent processes must be coordinated to control for negative interactions.
2. Activities may have real-time constraints, such as taking a picture of an asteroid during a narrow window of observability.
3. Plans must express compatibilities among activities, and the plan execution system must synchronise these activities at run-time.
4. The planner and plan execution system must reason about and interact with external agents and processes - these external agents can provide information at plan time, and achieve tasks and provide more information at runtime, but are never fully controllable or predictable.

5. Resources are limited and carefully budgeted - some resources are constant but limited, others are finite and must be budgeted across the entire mission. The planner reasons about resource usage and resource constraints must be enforced as part of plan execution. (Planning is also a limited resource.)
6. The spacecraft domain requires tracking of complex goals through time, thus the execution system must communicate the outcomes of commanded activities that affect those goals to the planning system, and the planner must use this information to update current goals and influence future planning.

It is impossible to plan an entire mission at the lowest level of detail due to the extended duration of missions as well as the unpredictability of actions [Muscettola et al 98]. The Remote Agent architecture therefore performs periodic planning, i.e. planning occurs at infrequent intervals and has a restricted horizon - the generation of a plan is explicitly represented in the plan as a task. A Mission Manager formulates short-term planning problems for the planner using a mission profile which lists all goals that are to be achieved during the mission. To overcome the problem of activities in one planning horizon compromising activities later in the mission, when the Mission Manager extracts goals for the next round of planning it also extracts constraints from the mission profile and presents both goals and constraints to the planner. The Remote Agent architecture has a reactive plan execution system which executes plans by decomposing high-level plan activities into commands to the real-time system. The executive component uses a procedural language ESL [Gat 96], to define alternate methods for decomposing activities. The granularity therefore differs between the planning and execution systems - planning occurs at a more abstract level. In addition, goals cannot usually be achieved with the level of satisfaction required - the planner has to trade off the level of goal satisfaction with respect to the long term "mission success" and within resource limitations. The top-level execution loop is presented in table 3.3.

Plans consist of temporal sequences of activities, or tokens - each activity has an earliest start time, a latest start time, an earliest end time, and latest end time. From the point of view of the executive, a plan is a set of timelines, each of which consists of a linear sequence of tokens. The planner used in the Remote Agent architecture consists of a heuristic search engine operating on a temporal database (provided by the HSTS system,

[Muscettola 94]).

Table 3.3 Top-level execution loop for the NMRA architecture

1. Begin waiting for signals of plan failure - if this occurs at any time in the current sequence, abort the currently executing plan and go to step 2.
2. Begin executing a standby plan to establish a stable state.
3. Invoke the planner to generate a new plan (for the next planning horizon), using the current state as the initial state for planning.
4. Continue executing the current plan while waiting for the new one.
5. Upon receipt of the new plan: merge the new plan into the current plan.
6. Upon reaching a new planning goal, repeat from step 3.

The Remote Agent architecture is domain specific which means that the context of the agent is captured through the use of domain specific search control rules. High-level goals are specified prior to the start of a mission through means of the mission profile - the Remote Agent architecture does not have a component responsible for generating high-level goals. The architecture doesn't choose whether to plan or whether to execute - planning for the next horizon is one of the tasks in the current plan. The planner is able to reason about both time and resources in a far more sophisticated way than the work addressed in this thesis. The executive component has the facility to repair the current plan in the event of unforeseen adverse events occurring, and, if it is unable to execute or repair the curent plan, it aborts the plan and puts the system into a stable safe state. Once a safe state has been entered, the executive component provides the Mission Manager with details of the current state and requests a new plan from the planner. Planning and execution occur simultaneously. In addition, execution occurs in parallel, as many separate devices execute simultaneously - unlike the work addressed in this thesis which assumes only one execution agent and that activities must be executed sequentially.

3.6 Summary

In this chapter we discussed in detail which factors cause an agent's motivations to change in strength as well as which factors cause new goals to be generated and existing goals to be updated. Taking these factors into account, we then discussed how often it is necessary to both update the agent's motivations and generate/update goals. We described an alternative control strategy which could be used in the planning/execution architecture presented in this thesis (see section 3.2.3, figure 3.1) and justified our decision to adopt the control strategy presented in chapter 2, figure 2.1. In section 3.3 we

introduced two alternative control strategies implemented as part of the "Plan to achieve goal" component - these strategies differ in the degree to which they allow planning and execution to be interleaved. The algorithm used to implement the control strategy of the planning/execution system was presented in table 3.1. Finally, we described two other planning systems that enable planning and execution to be interleaved (Sage, and the Remote Agent architecture) and compared and contrasted these systems with the system described in this thesis.

Chapter 4

Choosing Whether to Plan or to Execute

4.1 Introduction

In this chapter we begin by describing the data structures used to represent information required by the planning/execution architecture shown in chapter 2, section 2.2.1, figure 2.1, as well as the domain description language required by a user to specify problems that are to be solved by the system. We then introduce the truck world domain - a domain in which a truck-driver agent collects packages and parcels from various cities and delivers them to other cities. Finally, we describe in detail how the agent chooses whether to plan to achieve a goal or whether to execute an action - this corresponds to the component "Select goal or action" of the planning/execution architecture illustrated chapter 2, section 2.2.1, figure 2.1.

4.2 Representations

4.2.1 Introduction

In this section we describe the data structures used to represent the information required by the planning/execution algorithm illustrated in chapter 2, section 2.2.1, figure 2.1. This architecture extends the planning algorithm used in the planner SNLP [McAllester & Rosenblitt 91] to take into account the requirements described in chapter 1, sections 1.5 and 1.6. Whereas SNLP uses standard STRIPS representations for actions and goals, the planning/execution algorithm described in this thesis requires extensions to the standard STRIPS representations in order to enable the architecture to satisfy the above requirements. In this section we describe which particular requirements of the architecture merit extensions to the standard STRIPS representations. We then present the data structures required by the planning/execution algorithm in the following sections.

Context

The *context* of the agent, captured partly by modelling the motivations of that agent, plays an important role in the planning process in that it enables the agent to generate goals, to prioritise amongst goals as well as to select the best plan to achieve such goals. To support this we need to be able to represent the agent's `motivations`. In addition, the representations used for goals and actions have been extended to include the field `importance`, a value indicating how important it is to achieve the goal or to execute the action. Finally, the action representation has been extended to include the fields `pros` and `cons` - `pros/cons` store a set of values indicating the degree to which the action, when executed, supports/undermines some subset of the agent's motivations (these are used as part of the partial plan evaluation/ranking heuristic which is described in detail in chapter 6, section 6.4).

Planning and acting in real or simulated time

Time passes while the agent both plans to achieve goals and executes actions. Goals that are generated by the "Generate/update goals" component have an associated `deadline` by which they must be achieved and `duration` indicating how long they must remain true. It is assumed that deadlines associated with goals are hard (i.e. if the goal cannot be achieved by that deadline, the agent has effectively failed to achieve that goal). Actions also have an associated `deadline` indicating the latest time by which execution must commence, and `duration` (indicating an estimate of how long it will take to execute the action).

In addition, it may not always be possible to achieve all goals in the time available, in which case it is necessary to edit the plan to remove goals that cannot be achieved, together with their associated actions and constraints (see chapter 6, section 6.3 for further details). To facilitate this, the action representation has the associated field `goals` which contains a list of the goals to which the action contributes.

Interleaving planning and execution

Planning and execution are ongoing activities - because the agent is capable of continually generating new goals, planning is never complete which means that planning and execution must be interleaved. To support this requirement, the "Select goal or action" component decides, once each cycle, whether to plan to achieve a goal/subgoal or

whether to execute an action. The representation of goals and actions has therefore been extended to include the fields `importance`, `effort` and `deadline`, where `importance` and `deadline` have already been described, and `effort` is a value indicating the amount of effort which has currently been expended in planning to achieve the goal or, for actions, the amount of effort which has currently been expended in planning to achieve the set of goals to which the action contributes.

4.2.2 Motivations

The data structure `motivation` is used to represent each motivation. It consists of two fields:

- `name` - this specifies the name of the motivation (for example *hunger*).
- `strength` - this specifies the strength associated with the motivation (a real number). The value `strength` associated with each motivation changes depending upon the external environment. For example, the value `strength` associated with the motivation whose `name` is *hunger* will be low if the agent has just eaten but high if the agent has not eaten for a long time.

4.2.3 Actions

Actions are represented using the data structure `action`, which has the following fields.

- `id` - this is the unique identifier associated with the action.
- `type` - this specifies whether an action or an event is being represented (events are described by [Vere 83] and are not discussed in this thesis).
- `name` - this specifies the action's name and parameters, for example *(puton ?x ?y)*.
- `precond` - the action's preconditions are represented as a set of predicates indicating which facts must be true in the world in order that the action may be executed.
- `add` - a set of predicates representing facts which become true as a consequence of executing the action.
- `delete` - a set of predicates representing facts which were true in the world prior to executing the action but which are no longer true once the action has been executed.
- `pros` - a set of `(name strength)` tuples indicating the degree to which executing the action supports the agent's motivations. `name` indicates the motivation which is supported as a consequence of executing the action, while `strength` is a numerical

value indicating a prediction of the degree to which executing the action will support the motivation. For example, the action *(eat)* may have the `pros` tuple *(hunger 0.9)* which indicates that the action *(eat)*, once executed, supports the motivation with the `name` *hunger* - the value `strength`, 0.9, is a prediction of the degree to which the action *(eat)*, once executed, reduces the value `strength` associated with the motivation *hunger.*

- `cons` - a set of `(name strength)` tuples indicating the degree to which executing the action undermines the agent's motivations. `name` indicates the motivation which is undermined as a consequence of executing the action, while `strength` is a numerical value indicating a prediction of the degree to which executing the action undermines the motivation. For example, the action *(drive city1 city2)* may have the `cons` tuple *(conserve-fuel 0.5)* which indicates that the action *(drive city1 city2)*, once executed, undermines the motivation with the `name` *conserve-fuel* - the value `strength`, 0.5, is a prediction of the degree to which the action *(drive city1 city2)*, once executed, increases the value `strength` associated with the motivation *conserve-fuel.*
- `goals` - a set of (the unique identifiers associated with) goals generated by the "Generate/update goal" component, towards which executing the action contributes. This enables the planner to keep a track of dependencies so that goals with their supporting actions may be removed from the plan.

In addition, each action has an associated data structure known as a `node` (see section 4.2.5) which contains extra information such as the amount of effort expended in achieving the goals (which are represented by the set `goals`) towards which the action contributes, the action's importance, the action's deadline, and the action's duration.

4.2.4 Goals

Goals which are generated by the "Generate/update goals" component and subgoals/ open conditions which are generated by the "Plan to achieve goal" component are represented using the data structure `goal`, which has the following fields.

- `id` - the (unique) identifier associated with the goal. This is unique and newly generated for goals created by the "Generate/update goals" component. However, for subgoals/open conditions that are derived from actions (i.e. derived from the preconditions of an action), the identifier is the same as that of the action from which

the open condition was derived.

- `type` - this is used to indicate whether the goal was created by the "Generate/update goals" component or whether the goal was derived from the preconditions of an action (i.e. `type` can have the values :goal or :subgoal).
- `condition` - a predicate representing the goal/open condition.

Each goal has an associated data structure known as a `node` (see section 4.2.5) which contains extra information such as the amount of effort expended in achieving the goal, the goal's importance, the goal's deadline and the goal's duration.

4.2.5 Nodes

Each action and goal has an associated node which stores information such as how important the action or goal is, the amount of effort expended by the planner in achieving the goals towards which the action or goal contributes, the earliest/latest start time and duration associated with the action or goal.

- `id` - the node's identifier. This is the same as the unique identifier associated with the node's corresponding action or goal.
- `type` - this is one of three values, :goal, :action or :event depending upon whether the node corresponds to an action, goal or event (events are not discussed in this thesis).
- `importance` - this is a value indicating the importance of the node's corresponding action or goal.
- `effort` - if the node corresponds to a goal, the value `effort` indicates the amount of work expended by the planner in achieving that goal (i.e. through selecting actions and constraints that, once executed, achieve the goal). If the node corresponds to a subgoal, action or event, the value `effort` indicates the amount of work expended by the planner in achieving the goals towards which the subgoal, action or event contributes.
- `window` - this indicates the earliest and latest (i.e. the deadline) start times associated with the node. These are the times (earliest and latest) by which execution of the node's corresponding action/goal must commence. (The earliest and latest start times associated with a goal indicate the time window in which the goal must be achieved, whereas the earliest and latest start times associated with an action or event indicate the time window during which execution of the action or event must commence.)

- `duration` - an indication of how long the node's associated goal or action must remain true or takes to execute. If the node corresponds to a goal, the duration indicates how long the goal must remain true, whereas if the node corresponds to an action or event, the duration indicates how long the action or event takes to execute.

4.2.6 Plans

Partial plans are represented using the data structure `plan`:

- `steps` - the set of actions or steps contained in the plan.
- `links` - a set of tuples of the form `(establisher-id goal)` representing goals or subgoals which have been achieved by the "Plan to achieve goal" component (i.e. an action has been selected to achieve the goal or subgoal). `establisher-id` is the unique identifier of the establishing action or step, `goal` (see section 4.2.4 above) represents the goal or subgoal which has been achieved.
- `unsafe` - a set of tuples of the form `(link clobber-id clobber-bind)`. `link`, which is represented as the tuple `(establisher-id goal)`, is an unsafe link which may potentially be clobbered/undermined by the action whose unique identifier is equal to `clobber-id` if any variables are instantiated using the bindings belonging to the set `clobber-bind`. All potential conflicts represented by the set `unsafe` must be resolved as part of the "Achieve goal" process described in chapter 2, section 2.2.6, figure 2.2 and in chapter 5, section 5.3.
- `open` - a set of open conditions or goals or subgoals which are to be achieved. Each open condition is represented using the data structure `goal`.
- `ordering` - a set of tuples of the form `(id1 id2)` representing the ordering of actions and goals - `id1` and `id2` are the unique identifiers associated with actions or goals and indicate that the action/goal with the unique identifier `id1` occurs prior to the action/goal with the unique identifier `id2`.
- `bindings` - a set of elements representing the bindings and non-bindings of variables and constants. A binding is represented using the data structure `varset` (used in the implementation of SNLP and UCPOP) which includes the following fields: `const` - a unique constant; `cd-set` - the set of variables that bind/codesignate with the constant `const`; `ncd-set` - the set of variables that must not bind/codesignate with the constant `const` (and which must therefore also not bind/codesignate with

any of the variables belonging to the set `cd-set`).

- `goals` - a set of goals that have been generated by the "Generate/update goals" component. These are represented using the data structure `goal` (see section 4.2.4 above).
- `nodes` - a set of nodes which correspond to actions and goals. Nodes are used to represent the importance, effort, deadlines and durations associated with actions and goals (see section 4.2.5 above).
- `high-step` - an integer used to generate unique identifiers for actions and goals.

4.3 The Domain Description Language

In this section, the domain description language is described. This constitutes the information that must be supplied by a user (or other components of the agent architecture such as "Update motivations", "Generate/update goals") in order that the planning/execution architecture of chapter 2, figure 2.1 can create plans to achieve goals.

4.3.1 Operator schemas (action templates)

The agent's capabilities are represented using operator schemas - each operator schema represents a particular class of actions. For example, chapter 1, table 1.1 represents four classes of actions: stacking some block *?x* onto some block *?y*; unstacking some block *?x* from some block *?y*; picking up some block *?x* off the table; putting down some block *?x* onto the table (*?x* and *?y* are variables). The variables within each operator schema are instantiated during the process of planning to achieve some goal or subgoal, thereby creating an action instance. Operator schemas have the following fields.

- `name` - this specifies the action's name and parameters, for example *(stack ?x ?y)*.
- `precond` - the operator schema's preconditions are represented as a set of predicates indicating which facts must be true in the world in order that the operator schema may be executed.
- `delete` - a set of predicates representing facts which were true in the world prior to executing the operator schema but which are no longer true once the operator schema has been executed.
- `add` - a set of predicates representing facts which become true as a consequence of executing the operator schema.

- `bindings` - a set of bindings indicating which variables may not codesignate or bind with other variables.

4.3.2 Motivations

The domain designer must specify and initialise the agent's set of motivations. The architecture contains the component "Update motivations" which, once implemented, is responsible for updating the motivations in response to both changes in the environment and changes in the agent's partial plan. In the current implementation, a user is responsible for updating the strength associated with the motivations via an interface.

4.3.3 Goals

The domain designer must initially specify a set of goals for the "Plan to achieve goal" component to achieve. Each goal requires information concerning its condition, importance, deadline and duration. The architecture contains the component "Generate/update goals" which, once implemented, is responsible for generating/updating goals to reflect changes which occur to the environment, the agent's partial plan as well as the motivations. In the current implementation, a user is responsible for creating and updating goals via an interface.

4.3.4 The partial plan

In order to begin planning, the architecture requires a partial plan (to be specified by the domain designer) containing an action representing the initial state of the environment (this has the unique identifier 0, null precondition and delete lists), together with its associated node.

4.3.5 Look-up tables

The planner requires two look-up tables, one containing estimates of the duration (i.e. the time it takes to execute an action) of each action (this enables the planner to instantiate the field `duration` when creating or further instantiating actions - see chapter 5, section 5.2.2 for further details), the other containing an estimate of the degree to which each action supports or undermines the agent's motivations (this enables the planner to instantiate the fields `pros` and `cons` when creating or further instantiating actions - see chapter 5, section 5.2.3 for further details). The look-up tables are accessed by the

action's name and parameters, for example *(stack A B)*. The look-up tables require domain and agent specific knowledge - in an extended agent architecture the information contained in these tables could be learnt using some sort of learning component.

4.3.6 The current time

The domain designer needs to specify the current time.

4.3.7 Discussion

Other planners that reason about time such as DEVISER [Vere 83], Zeno [Penberthy & Weld 94] and IxTeT [Laborie & Ghallab 95] encapsulate information specifying the duration of activities within operator schemas. The disadvantage of this approach is that it may make the assumption that each action instance has the same duration regardless of how that action is instantiated. For example, an operator schema used to represent the activity of flying an aeroplane from an initial location to a destination location may assume that the length of time taken to execute this action is the same regardless of the type of aeroplane. This is clearly not the case, as an aeroplane such as Concorde will fly between locations much quicker than a Boeing 747. In order to overcome this, two operator schemas are required, one dealing with each type of aeroplane. This approach may lead to a proliferation of operator schemas - Zeno has two flying operator schemas, one responsible for flying quickly, another for flying slowly. The approach adopted in this thesis is to keep information concerning the duration of actions within a look-up table as opposed to within operator schemas. It is the responsibility of the planner to estimate the duration of each action instance as part of the process of instantiating operator schemas/ action instances.

4.4 The Truck World Domain

In order to illustrate the workings of each of the components used within the planning/ execution architecture presented in this thesis (see chapter 2, figure 2.1), in this section we describe the truck world domain - a domain in which a truck-driver transports parcels and packages from one city to another.

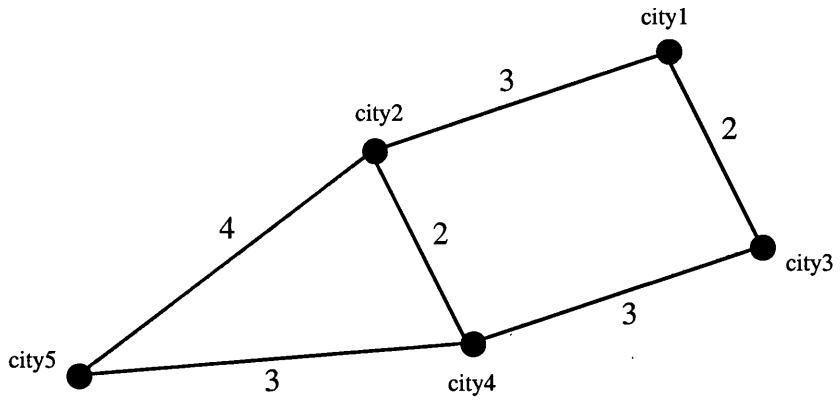


Figure 4.1 The Truck World Domain

Figure 4.1 shows the topology of the truck world domain which consists of five cities connected by roads. The numbers illustrate the units of time it takes to drive from one city to another - for example, to drive from *city2* to *city4* takes 2 units of time. In this domain, it is the duty of a single truck-driver to transport packages or parcels from one city to another by some deadline. At any point in time, the truck-driver may receive an instruction to pick up some package from one city, and to transport that package to another city. The truck-driver is an agent acting within this domain and requires the planning and execution architecture described in this thesis so that it can act to achieve its goals in an intelligent, timely fashion. In order to do this, the planning and execution architecture requires the following information (i.e. this information is described in section 4.3 above as the domain description language) which is described in detail later in this section.

1. A description of the initial state which is encapsulated within an initial plan.
2. A description of the goal state.
3. A set of operator schemas/action templates - these represent the capabilities of the truck-driver.
4. The truck-driver's current motivations.
5. A look-up table specifying the amount of time it takes the truck-driver to execute each action.
6. A look-up table specifying the degree to which executing each action will support or undermine the truck-driver's motivations.
7. The current time.

4.4.1 A description of the initial state of the truck world domain

Table 4.1 describes the initial state of the truck world domain. The *truck* (and truck-driver) and *package* are at *city1*. Various predicates are used to represent static features within the domain (i.e. invariants) such as which cities are connected to which other cities. In this example, it is assumed that the *truck* always has fuel. Changing features in this environment include the *truck* and the *package*, which can both be moved from one city to another. The initial state of the truck world domain is encapsulated within an ini-

Table 4.1 The initial state of the truck world domain

invariants	variants
connects(city1 city2) & connects(city2 city1) & connects(city1 city3) & connects(city3 city1) & connects(city2 city4) & connects(city4 city2) & connects(city2 city5) & connects(city5 city2) & connects(city3 city4) & connects(city4 city3) & connects(city4 city5) & connects(city5 city4) & has-fuel(truck)	*at(truck city1) & at(package city1)*

tial partial plan which is shown in table 4.2 below. The representation of the initial state of table 4.1 is encapsulated within the *add* field of a dummy action whose *name* field is assigned the value *initial*. The initial plan also contains a newly generated goal *at(package city5)* which is described in the next section.

Table 4.2 The initial plan

actions:	id:	name:	add:	
	0	*initial*	*connects(city1 city2) & connects(city2 city1) & connects(city1 city3) & connects(city3 city1) & connects(city2 city4) & connects(city4 city2) & connects(city2 city5) & connects(city5 city2) & connects(city3 city4) & connects(city4 city3) & connects(city4 city5) & connects(city5 city4) & has-fuel(truck) & at(truck city1) & at(package city1)*	
links	*nil*			
open:	*nil*			
unsafe:	*nil*			
ordering:	*nil*			
bindings:	*nil*			
goals:	id:	conditions:	deadline:	importance:
	1	*at(package city5)*	*20*	*6*

4.4.2 A description of the goal

Initially, the desired goal is as illustrated in table 4.3 - the *package* must be delivered to *city5* by 20 units of time. The *importance* associated with this goal is assigned the value 6 and the goal must remain true for 0 units of time. In a complete implementation of the

Table 4.3 The first goal

id:	*1*
condition:	*at(package city5)*
importance:	*6*
deadline:	*20*
duration:	*0*

planning/execution architecture described in chapter 2, figure 2.1, this goal would be generated by the "Generate/update goals" component.

4.4.3 The operator schemas/action templates

Table 4.4 Action templates/operator schemas for the truck world domain

name:	*drive-truck (?truck ?from ?to)*
precondition:	*at (?truck ?from) &* *has-fuel(?truck) &* *connects(?from ?to)*
delete:	*at(?truck ?from)*
add:	*at(?truck ?to)*
bindings:	*(not (?from ?to)) &* *(not(?from ?truck)) &* *(not(?to ?truck))*

name:	*unload(?ob ?truck ?loc)*
precondition:	*at(?truck ?loc) &* *in(?ob ?truck)*
delete:	*in(?ob ?truck)*
add:	*at(?ob ?loc)*
bindings:	*(not(?ob ?truck)) &* *(not(?ob ?loc)) &* *(not(?truck ?loc))*

load(?ob ?truck ?loc)
at(?truck ?loc) & *at(?ob ?loc)*
at(?ob ?loc)
in(?ob ?truck)
(not(?ob ?truck)) & *(not(?ob ?loc)) &* *(not(?truck ?loc))*

Table 4.4 contains three operator schemas/action templates which represent the capabili-

ties of the truck-driver agent - for example, the truck-driver can drive from one city to another provided the truck has fuel and that the cities are connected, and it can load and unload objects such as packages and parcels into or out of the truck at various locations.

4.4.4 The agent's motivations

In this example the agent has three motivations which are shown in table 4.5. The first is concerned with maximising pleasure, the second with conserving fuel and the third with conserving tyres. Each motivation is initialised so that its motivational value is 0.

Table 4.5 The truck-driver's motivations

Motivations			
name:	*pleasure*	*conserve-fuel*	*conserve-tyres*
strength:	*0.0*	*0.0*	*0.0*

4.4.5 Values indicating duration

The planning/execution architecture requires access to a look-up table which stores values indicating the duration associated with actions. The look-up table, shown table 4.6, is accessed using the action's *name* field as the key. The duration associated with the action *drive-truck(truck city2 city1)* is the same as that associated with *drive-truck(truck city1 city2)* - this applies to the other instantiations of the *drive-truck* action. In addition, the

Table 4.6 Values indicating duration

action name	duration
drive-truck(truck city1 city2)	*3*
drive-truck(truck city1 city3)	*2*
drive-truck(truck city2 city4)	*2*
drive-truck(truck city2 city5)	*4*
drive-truck(truck city3 city4)	*3*
drive-truck(truck city4 city5)	*3*
drive-truck	*4*
unload-truck	*1*
load-truck	*1*

duration associated with incomplete instantiations of the action *drive-truck* assume the worst case value 4. The actions *unload-truck* and *load-truck* take the same amount of

time regardless of how they are instantiated.

4.4.6 Values indicating the degree of support

Table 4.7 Values indicating the degree of support of each action

action name	pros		cons	
	name	strength	name	strength
drive-truck(truck city1 city2)	*pleasure*	*0.1*	*conserve-fuel* *conserve-tyres*	*1.2* *1.0*
drive-truck(truck city1 city3)	*pleasure*	*1.9*	*conserve-fuel* *conserve-tyres*	*0.3* *0.1*
drive-truck(truck city2 city4)	*pleasure*	*1.2*	*conserve-fuel* *conserve-tyres*	*0.3* *0.2*
drive-truck(truck city2 city5)	*pleasure*	*0.2*	*conserve-fuel* *conserve-tyres*	*1.0* *0.9*
drive-truck(truck city3 city4)	*pleasure*	*1.2*	*conserve-fuel* *conserve-tyres*	*0.3* *0.4*
drive-truck(truck city4 city5)	*pleasure*	*1.8*	*conserve-fuel* *conserve-tyres*	*0.3* *0.4*
drive-truck	*pleasure*	*0.0*	*conserve-fuel* *conserve-tyres*	*0.0* *0.0*
unload-truck	*pleasure*	*0.0*	*conserve-fuel* *conserve-tyres*	*0.0* *0.0*
load-truck	*pleasure*	*0.0*	*conserve-fuel* *conserve-tyres*	*0.0* *0.0*

A look-up table is required which stores values indicating the degree to which actions in the truck world domain support the truck-driver's motivations. This table, shown in table 4.7, is accessed using the action's *name* field as the key. For example, the action *drive-truck(truck city1city2)* supports the motivation *pleasure* (the strength, 0.1, indicates the degree to which the truck-driver finds the route between *city1* and *city2* pleasurable to drive along) and undermines the motivations *conserve-fuel* (the value 1.2 means that the truck consumes a lot of fuel when driving between *city1* and *city2*) and *conserve-tyres* (the value 1.0 means that the road between *city1* and *city2* is bad for the truck's tyres). The degree to which the action *drive-truck(truck city2 city1)* supports or undermines the truck-driver's motivations is the same as that associated with *drive-truck(truck city1 city2)* - this applies to the other instantiations of the *drive-truck* action. Incomplete instantiations of the actions *drive-truck*, *unload-truck* and *load-truck* have no effect upon

the truck-driver's motivations.

4.4.7 Summary

The truck world domain will be used in the remainder of this thesis to illustrate the workings of each of the components. Because the components "Update motivations" and "Generate/update goals" have not been implemented, it is not possible to give a complete demonstration of the overall performance of the planning/execution system in this domain. For example, future work could involve extending this domain to demonstrate how using up fuel and wearing down the truck's tyres alters the strength of the agent's motivations which in turn causes goals to be generated that involve obtaining fuel and replacing the tyres. At present such mechanisms are not implemented.

4.5 Choosing to Plan or to Execute

In the following sections we describe in detail how the agent chooses whether to plan to achieve a goal/subgoal or whether to execute an action - this corresponds to the component "Select goal or action" of the planning/execution architecture illustrated in chapter 2, section 2.2.1, figure 2.1. The first part of the process involves determining which, if any, actions are ready to be executed: an action is ready to be executed if its preconditions are established in the current state and if the current time is before or equal to the action's execution deadline. Once the set of actions that are ready to be executed has been determined, the set of open conditions (consisting of both goals and subgoals) and ready to be executed actions are evaluated to determine the "best" goal/subgoal or action. Goals/subgoals and actions are evaluated on the basis of values indicating their *importance*, *effort* and *deadline* (see sections 4.2.3 , 4.2.4 and 4.2.5 which describe how actions and goals/subgoals are represented).

1. *importance* is a value indicating the importance of the action/goal/subgoal - the higher the value, the more important the action/goal/subgoal. Each goal generated by the "Generate/update goals" component has assigned to it a value indicating its *importance*. The *importance* associated with an action or subgoal however, is determined by the *importance* associated with the goals towards which the action or subgoal contributes. Actions/goals/subgoals with a high value of *importance* are favoured.

2. *effort* is a value indicating the amount of effort which has already been expended by the "Plan to achieve goal" component - the higher the value, the greater the amount of effort that has already been expended. The value *effort* associated with a goal generated by the "Generate/update goals" component is the amount of effort expended by the "Plan to achieve goal" component in achieving that goal. The value *effort* associated with an action or subgoal is determined by the effort expended in achieving the goals to which the action/open condition contributes (this will be explained further in section 4.5.3 below). Actions/goals/subgoals with a high value of *effort* are favoured.

3. *deadline* is a value indicating the time by which the action/goal/subgoal must be executed/true (an action must be executed by this deadline whereas a goal or subgoal must be established by this time) - the lower the value, the earlier the deadline. Actions/open condition with a low value of *deadline* are favoured.

Once evaluation has taken place, the best action/goal/subgoal is returned - if it is an action, that action is executed, while if it is a goal or subgoal, that goal or subgoal is achieved. In the following sections we describe the parameters *importance*, *effort* and *deadline* in more detail.

4.5.1 Importance

Earlier, in chapter 2, sections 2.2.3 and 2.2.4, we saw that motivations cause goals to be generated - by achieving such goals an agent is able to satisfy the motivations that caused those goals to be generated. If the strength of a motivation such as *hunger* is high, the goal, *have-food*, might be generated. Once that goal is satisfied, the strength associated with *hunger* is reduced. In addition, the strength associated with the motivations of an agent directly determines the *importance* of any goals generated in order to mitigate those motivations. If a motivation is high in strength, any goal generated to satisfy that motivation will also be high in *importance*. The external environment constantly changes as a result of either the agent acting to satisfy its goals (this also includes physical internal changes that take place within the planning/execution agent) or due to the activities of other autonomous agents or physical processes. In addition, the agent's internal plan changes as a consequence of the agent planning to achieve some goal. Both changes cause the motivations of the agent to change in strength which in turn may affect the *importance* of the goals generated previously to mitigate those motivations. The *impor-*

tance of a goal may therefore change while the agent plans and acts.

For example, an agent might receive an e-mail message informing them that a talk is to be given during the afternoon of the following day. The e-mail message contains the title of the talk together with the time, venue and the name of the person delivering the talk. Because the title sounds relevant to the agent's current research interests, the agent may decide to attend this talk. Attending the talk can be viewed as being a goal generated by the "Generate/update goals" component in response to the agent receiving the new information, which has a fairly high value of importance (determined by the estimated relevance of the talk to the agent's research interests). Subsequently, the agent may receive a further e-mail communication which includes an abstract describing in more detail what the talk will cover. After reading this e-mail, the agent may realise that the talk is not as relevant to their current research interests as the previous message led them to believe. In view of this newly acquired information it may no longer be as important that the agent attends the talk.

Another example involves an agent perceiving that a kitchen tap is leaking, which leads the agent to generate a goal to repair the tap. Initially the importance associated with this goal might not be high. However, if the leak suddenly gets a lot worse, the importance of repairing the tap increases significantly as a consequence of a severe leak might involve costly repairs.

The importance of the agent's goals might therefore change whilst the agent is planning and acting within its environment. The design of a mechanism which demonstrates how changes in the agent's motivations directly affect the importance of the agent's goals is beyond the scope of this work. However, it is assumed that the importance of the agent's goals can change in response to the agent's motivations. For the purpose of this thesis however, it is assumed that the *importance* associated with goals generated by the "Generate/update goals" component is fixed.

Finally, because goals are assigned a value indicating their importance, it is also possible to assign a value indicating the importance of each action (and its preconditions or subgoals) that is selected to achieve such goals. The value indicating the importance of an action and its preconditions or subgoals is determined by summing the values indicating the importance associated with each goal to which the action and its preconditions contributes. In section 4.5.3 we describe how actions and subgoals are assigned a value indicating their importance. In the "Select goal or action" procedure, actions/goals/sub-

goals with a high associated value indicating their *importance* are preferred.

4.5.2 Effort

In addition to having a value indicating *importance*, each goal that is generated by the "Generate/update goals" component has a value indicating the amount of *effort* which has currently been expended by the planner in achieving that goal. Each time an action is selected by the "Plan to achieve goal" component to contribute towards the achievement of a goal *g (*generated by the "Generate/update goals" component), the value *effort* associated with *g* is incremented. As with *importance*, it is also possible to calculate a value indicating the *effort* associated with each action and subgoal within the partial plan (this process is described in section 4.5.3 below). Actions/goals/subgoals with a high associated value for *effort* are preferred. (It could be argued however, that once the value *effort* associated with a goal has exceeded a certain threshold, the selection strategy should no longer continue to try achieving that goal as the "Plan to achieve goal" component has invested too much time in achieving the goal.)

4.5.3 The importance and effort of actions and subgoals

The role of the "Generate/update goals" component is to generate and update goals in response to the agent's current motivations, predicted future motivations, the current state of the world and predicted future states of the world (see chapter 2, section 2.2.4). As mentioned in previous sections (see section 4.5.1 and 4.5.2), associated with each goal generated by this component is a value indicating its *importance* (this value is determined by the strength associated with each motivation that led to the generation of the goal) and a value indicating the amount of *effort* expended by the "Plan to achieve goal" component in achieving that goal (this value is initially 0). Each goal generated by the "Generate/update goals" component is subsequently presented to the "Plan to achieve goal" component which generates a sequence of actions that, once executed, achieves the goal. It is therefore possible to calculate values indicating the *importance* and *effort* associated with each action and subgoal (i.e. the preconditions of actions) within the partial plan.

For example, the "Generate/update goals" process generates two goals, g_1 and g_2, which have the values 6 and 4 respectively indicating their *importance*. Initially, both goals have the value 0 indicating *effort* as no planning has yet taken place. The "Select

goal or action" process chooses to achieve the goal g_1. A newly created action a_1 is then selected by the "Plan to achieve goal" process to achieve the goal g_1 - the value indicating the *importance* of executing a_1 is equal to the value indicating the *importance* associated with the goal g_1 (the value 6). The value *effort* associated with g_1 is incremented to 1 by the "Plan to achieve goal" procedure to reflect the effort expended by that procedure in achieving g_1 - the value *effort* assigned to the newly generated action a_1 is equal to the newly generated value *effort* associated with g_1 (the value 1). Likewise, the values *importance* and *effort* associated with each subgoal that is derived from the preconditions of a_1 are equal to the values *importance* and *effort* associated with the goal g_1 (the values 6 and 1 respectively). In fact, all actions that contribute towards the achievement only of g_1 (for example the actions that are selected to achieve the preconditions of a_1 and their preconditions) assume the same values of *importance* and *effort* (the values 6 and 1) as those associated with g_1.

Later, the "Select goal or action" procedure chooses to achieve the goal g_2. The action a_1 is selected by the "Plan to achieve goal" process to achieve the goal g_2. As a consequence of this decision a_1 becomes more important, as executing a_1 now contributes to the achievement of two goals, g_1 and g_2. The new value indicating the increased *importance* of a_2 is the sum of the values indicating the *importance* associated with the two goals g_1 and g_2 (i.e. the value 10). In addition, the value *importance* associated with each subgoal derived from the preconditions associated with a_1 and the value *importance* associated with all actions and subgoals that contribute towards the achievement of the goals g_1 and g_2 also increases. Likewise, the value *effort* associated with the goal g_2 is incremented to 1 to reflect the extra effort expended by the "Plan to achieve goal" component in achieving this goal. The new value *effort* associated with a_1 and with the subgoals that are derived from the preconditions of a_1 (and all actions that contribute towards the achievement of the goals g_1 and g_2) is the sum of the values indicating the amount of effort expended in achieving g_1 and g_2 (the value 2). The process of assigning values indicating the importance and effort of actions and subgoals is described in further detail in chapter 5, sections 5.2.5 and 5.2.6).

If the importance associated with goals that are generated by the "Generate/update goals" component changes whilst such goals are being achieved by the "Plan to achieve goal" component, such changes propagate changes in the value of importance associated with actions and subgoals within the partial plan. A mechanism has been developed to

propagate such changes in the value of importance and associated with actions.

4.5.4 Importance and urgency

In this section we consider how importance and urgency differ. Urgency is directly related to the passing of time. As the time approaches the point by which a sequence of actions must be executed in order to achieve a goal by some deadline, urgency increases. For example, if the intention is to submit a paper to a conference, the goal of having written the paper is less urgent if the deadline is far in the future. As time approaches the submission deadline, it becomes more and more urgent to achieve the goal of having written the paper. This notion of urgency is rather similar to an increase in importance. Urgency, like importance, is a factor that causes the agent to focus more or less on a goal. Importance however, is independent of time. Urgency is represented by reasoning about the deadlines associated with actions and open conditions which is discussed in the following section.

4.5.5 Deadlines

Goals that are generated by the "Generate/update goals" process also have an associated deadline. Using ordering constraints, it is possible to estimate deadlines for each action and subgoal within a partial plan. The process of estimating the deadline for actions and their associated open conditions (derived from their preconditions) is described in more detail in chapter 6, section 6.2. Actions/goals/subgoals with imminent deadlines are preferred.

4.5.6 Choosing whether to plan or execute - the algorithm

In this section we describe the algorithm used by the "Select goal or action" component to determine whether a goal/subgoal should be achieved or whether an action should be executed. This process requires the set of outstanding goals belonging within a partial plan as well as the set of actions that are ready to be executed.

An action is ready to be executed if it is possibly first, its preconditions are true in the current state and the current time is prior or equal to its execution deadline. It may be the case that there are no actions that are ready for execution, in which case the algorithm chooses to achieve the best outstanding goal or subgoal.

Once the set of actions that are ready to be executed has been determined, each open

condition (i.e. goals or subgoals) and action (which is ready to be executed) is evaluated on the basis of its associated values *importance*, *effort* and deadline. The evaluation function prefers (i.e. assigns higher values to) actions/goals/subgoals with high values of *importance* and *effort* and with imminent deadlines. The evaluation function first determines the latest deadline associated with the set of ready-to-execute actions and outstanding goals. For each ready-to-execute action and outstanding goal, the evaluation function first subtracts the deadline associated with the ready-to-execute action or outstanding goal from the value indicating the latest deadline, and then adds the result to the values *importance* and *effort*.

Note: more than one open condition may have the same value as a result of evaluation. This is because one or more open conditions may be derived from the preconditions of a single action which means they have the same values for *importance*, *effort* and deadline. In this case, the first of such open conditions belonging within the set of open conditions is returned. The algorithm is described in table 4.8.

Table 4.8 Choosing whether to plan or to execute

1. Let *gls* be the set of outstanding goals (containing goals and subgoals) belonging to the partial plan *p*.
2. Let *readyacts* be the set of actions that are ready to be executed. (An action is ready to execute if it is possibly first, its preconditions are true in the current state, and the current time is prior or equal to the action's deadline. The algorithm only returns a set of actions provided that the preconditions of all possibly first actions are true in the current state.)
3. Let *best* be the best goal/subgoal or action - each goal/subgoal or action belonging to the sets *gls* and *readyacts* is evaluated on the basis of its associated values *importance*, *effort* and deadline.
   (a) If *best* is one of the actions that are ready to be executed, return two values: the value *:action*; (the unique identifier associated with) the best action.
   (b) Else, - *best* represents one or more subgoals (subgoals are derived from the preconditions of actions which means more than one subgoal may have the same associated *importance*, *effort* and deadline), return two values: the value *:goal* and the first subgoal belonging to the set of open conditions which corresponds to *best* (i.e. the first subgoal whose unique identifier is equal to the unique identifier associated with *best*).
   (c) Else, there are no outstanding goals/subgoals and no actions that are ready to execute. In this case, the system waits until new goals are generated.

4.6 Example

4.6.1 An example

In chapter 5, section 5.4.1, table 5.9 we illustrate how the planning/execution architecture generates a partial plan to achieve the goal *at(package city5)* by its deadline of 20

units (this goal is shown in section 4.4.2, table 4.3). During the process of executing the sequence of actions to achieve this goal, the partial plan shown in chapter 5, section 5.4.2, table 5.10 is generated. This partial plan contains one action that is ready to execute, *drive-truck(truck city4 city5)*, (see chapter 5, section 5.4.2, table 5.14 - it can be seen that the preconditions of this action are true in the current state which is shown in chapter 5, section 5.4.2, table 5.11) and one newly generated goal, *at(parcel city5)*, which is shown in chapter 5, section 5.4.2, table 5.12. The "Select goal or action" component takes into account the importance, effort and deadline associated with the action and the goal. Table 4.9 shows values for each of these fields. The algorithm described in

Table 4.9 The goal and action

	Goal
id:	*7*
condition:	*at(parcel city5)*
importance:	*20*
effort:	*0*
deadline	*16*

	Action
id:	*4*
name:	*drive-truck(truck city4 city5)*
importance:	*6*
effort:	*14*
deadline:	*16*

section 4.5.6 assigns the value 20 to both the goal and the action - because goals take time to achieve (i.e. they require planning which may result in a sequence of more than one action which takes time to execute), the "Select goal or action" component decides to achieve the goal *at(parcel city5)*.

4.7 Discussion

The algorithm described above is used in the implementation of the "Select goal or action" component presented in this thesis and is not necessarily the best algorithm. A better algorithm might take into account an estimate of the amount of time required to achieve each outstanding goal by their deadline, and use this estimate to determine whether a goal should be achieved or whether an action should be executed.

The purpose of the "Select goal or action" component is to enable an agent to reflectively evaluate what course of action to take (i.e. whether to plan or to execute) and to utilise its resources accordingly.

4.8 Summary

In this chapter we described the data structures used to represent information required by the planning/execution architecture shown in chapter 2, section 2.2.1, figure 2.1, in order to support the requirements outlined in chapter 1, sections 1.5 and 1.6. We also described the domain description language required by a user in order to specify problems that are to be solved by the planning/execution system. We then introduce the truck world domain - a domain in which a truck-driver agent collects packages and parcels from various cities and delivers them to other cities. The truck-driver requires the planning/execution system described in this thesis in order to perform tasks within the truck world domain in a timely manner. Finally, we described in detail how the agent chooses whether to plan to achieve a goal or whether to execute an action - this corresponds to the component "Select goal or action" of the planning/execution architecture illustrated chapter 2, section 2.2.1, figure 2.1 - and present an example of the truck-driver agent choosing whether to achieve a goal or execute an action.

Chapter 5

Planning to Achieve a Goal - Part 1

5.1 Introduction

In this and the following chapter we describe the main focus of the research in this thesis, the "Plan to achieve goal" process shown in chapter 2, section 2.2.1, figure 2.1. This process, illustrated in more detail in chapter 2, section 2.2.6, figure 2.2 and summarised in table 5.1, takes a goal or subgoal chosen by the "Select goal or action" component and generates several new partial plans which achieve that goal or subgoal.

First, the "Achieve goal" process takes the best outstanding goal/subgoal (chosen by the "Select goal or action" component of chapter 2, section 2.2.1, figure 2.1) and generates a new partial plan which achieves that goal/subgoal by selecting a new or existing action (a set of operator schemas/action templates is used to enable a new action to be created) and by posting temporal and binding constraints. In addition, to ensure that the achievement of the goal/subgoal does not conflict with goals/subgoals which have already been achieved, further temporal and binding constraints may be posted (this process is known as conflict resolution). The "Estimate deadlines" component then estimates the deadlines associated with each action and subgoal (subgoals are derived from the preconditions of actions) within the newly generated partial plan. If this process fails it means there is insufficient time available to achieve all of the goals within that partial plan, in which case, the "Edit the partial plan" process removes a goal together with its associated actions and constraints. Once the partial plan has been edited, the "Estimate deadlines" process reestimates the deadlines associated with actions and subgoals - again, if there is insufficient time available, the "Edit the partial plan" process removes another goal together with its associated actions and constraints. This process is repeated until the "Estimate deadlines" procedure is able to successfully assign deadlines to the remaining actions and subgoals (this means there is sufficient time available to achieve

all of the goals in the partial plan). A number of alternative partial plans will be generated by the "Achieve goal" process, representing the alternative ways there are of achieving the goal/subgoal. Once deadlines have been successfully assigned to the actions and subgoals within each partial plan, the "Evaluate partial plans" process evaluates/ranks each partial plan with respect to the current motivations of the agent (this determines the degree to which each partial plan supports/undermines the agent's motivations). Finally, each newly generated partial plan is sorted and merged (on the basis of its ranking) with the ordered search space of partial plans.

The algorithm used within the "Plan to achieve goal" component has been designed to support the following requirements of the overall planning/execution architecture.

- **Context.** The context of the agent (captured partly by modelling the motivations of that agent) plays an important role in enabling the agent to generate goals, to prioritise amongst actions and goals as well as to select the best partial plan in which such goals are achieved. The implementation of this requirement is supported by the "Achieve goal" component which assigns values indicating the importance associated with actions and their preconditions, and the degree to which actions support or undermine the agent's motivations. In addition, the "Evaluate partial plans" process evaluates and ranks each partial plan by calculating the degree to which a plan supports or undermines the agent's motivations.
- **Planning and acting in real or simulated time.** Time passes while the agent executes actions. Goals that are generated by the "Generate/update goals" component have an associated deadline by which they must be achieved, and duration indicating how long they must remain true. It is assumed that deadlines associated with goals are hard (i.e. if the goal cannot be achieved by its deadline, the agent has effectively failed to achieve that goal). Actions also have duration. It is important that the agent ensures that it can achieve goals by their deadline, if possible. If there is insufficient time available to achieve all of the goals within the partial plan by their deadlines, it is necessary to create extra time by abandoning the achievement of one or more goals (i.e. by editing the partial plan to remove such goals). To support these requirements, the "Achieve goal" process both estimates how long it takes to execute each action (i.e. it assigns a value indicating the duration of each action) and keeps track of which goals each action contributes towards (to facilitate plan editing). The "Estimate deadlines" process estimates the deadlines associated with actions and subgoals belonging within

each partial plan. Finally, the "Edit the partial plan" process edits the plan by removing one or more goals and their associated actions and constraints.

Table 5.1 Planning to achieve a goal/subgoal

The "Select goal or action" process selects the best goal/subgoal which is then passed to the "Plan to achieve goal" component.

1. A set of new partial plans are generated in which the goal/subgoal is achieved. Such plans achieve the goal/subgoal by using new actions (step addition) or by using existing actions (simple establishment). Conflict resolution is performed on all newly generated partial plans. The algorithms used are an extension of the goal achievement procedures used in SNLP.
2. The deadlines for actions and goals within each newly generated partial plan are estimated.
   (a) If the deadline estimation process fails this indicates that there is not enough time to achieve all of the goals within the plan. The plan is therefore edited to remove one goal together with its associated actions and constraints. Once editing is complete the deadlines for the actions and subgoals within that plan are reestimated (i.e. back to stage 2).
   (b) Else, if the deadline estimation process succeeds, the partial plan is evaluated with respect to the agent's motivations.
3. The set of newly generated plans together with their values (see 2. (b)) are sorted and merged with the (ordered) search space of partial plans.

- **Interleaving planning and execution.** Planning and execution are ongoing activities - because the agent is capable of continually generating new goals, planning is never complete which means that planning and execution must be interleaved. To support this requirement, the "Select goal or action" component of chapter 2, section 2.2.1, figure 2.1, decides, once each cycle, whether to plan to achieve a goal/subgoal or whether to execute an action. In order to do this, goals/subgoals and actions are evaluated on the basis of values indicating their importance, deadlines and effort (for goals, the value *effort* represents the amount of effort previously expended by the planner in achieving those goals, while for actions and their associated preconditions, the value *effort* represents the amount of effort previously expended by the planner in achieving the set of goals to which each action and its preconditions contributes). The "Plan to achieve goal" component is responsible for assigning values indicating the importance, effort and deadline to each action and its preconditions - the "Achieve goal" process assigns values indicating the importance, effort and duration associated with each action and its preconditions, while the "Estimate deadlines" process assigns deadlines to each action and its preconditions (using the values indicating the duration of each action).

In this and the following chapter we describe the four processes that comprise the "Plan to achieve goal" component in further detail. The remainder of this chapter will focus upon the "Achieve goal" procedure while the following chapter, chapter 6 will

describe the remaining three procedures, "Estimate deadlines", "Edit the partial plan" and "Evaluate the partial plan".

5.2 Achieving a Goal

5.2.1 Introduction

In the remainder of this chapter we describe in detail the process "Achieve goal" shown in chapter 2, section 2.2.6, figure 2.2, which takes as input a goal or subgoal (chosen by the "Select goal or action" process) and generates a number of new partial plans which achieve that goal or subgoal. In order to implement this process, the goal achievement algorithm used in the nonlinear planner SNLP (McAllester & Rosenblitt 91] has been extended to take into account the three requirements listed in the previous section: context; planning and acting in real or simulated time; interleaving planning and execution. In chapter 4, section 4.2 we described how the basic STRIPS representations which are used by SNLP have been extended to meet these requirements. In particular, the action representation has extra fields indicating its *duration*, *importance*, *effort*, *deadline*, *pros* and *cons* (these latter two fields contain values that indicate the degree to which executing that action supports or undermines the agent's motivations), see chapter 4, sections 4.2.3 and 4.2.5 for a description of each field. The extensions made to the goal achievement algorithm used by SNLP involve instantiating (i.e. assigning values to) each of these fields as part of the process of achieving a goal/subgoal.

The goal achievement algorithm used in SNLP consists of two stages: selecting a new or selecting an existing action to achieve the goal (known as step addition and simple establishment respectively); and ensuring that there are no conflicts (known as conflict resolution). The algorithm therefore consists of three main processes.

1. Step addition - this involves achieving the goal or subgoal by creating a new action (by selecting and instantiating an operator schema/action template) and adding that action together with new binding and temporal constraints to the partial plan. (The goal achievement algorithm therefore has access to a set of operator schemas/ action templates which collectively represent the agent's execution capabilities.)
2. Simple establishment - new temporal and binding constraints are posted to ensure the goal or subgoal is achieved by an action currently within the partial plan.
3. Conflict resolution - new temporal and binding constraints are posted to ensure that

the action selected to achieve the goal/subgoal does not conflict with goals/subgoals that have already been achieved and that actions currently within the partial plan do not conflict with the newly achieved goal/subgoal.

The goal achievement algorithm (in particular, the step addition and simple establishment processes) used in SNLP has been extended to incorporate the following features.

1. Actions have duration (i.e. they take time to execute) - one of the tasks of the "Achieve goal" component is to assign a value indicating the estimated duration associated with each action. This information is required by the "Estimate deadlines" process in order to assign deadlines to each action and subgoal. In turn, the deadlines of actions and subgoals are required by the "Select goal or action" component.
2. It is necessary to be able to determine the degree to which each action supports the agent's motivations. The "Achieve goal" component assigns values to each action which indicate the estimated degree to which that action, once executed, supports or undermines the agent's motivations. These values are required by the "Evaluate partial plans" component.
3. The "Achieve goal" component determines which goals each action contributes towards. This information is required by the "Achieve goal" component to calculate values indicating the importance and effort associated with actions and subgoals (see below). In addition, this information is required by the "Edit the partial plan" process to enable goals and their associated actions and constraints to be removed from a partial plan.
4. The "Achieve goal" component estimates the importance associated with actions and subgoals. These values are required by the "Select goal or action" component.
5. The "Achieve goal" component calculates the effort invested in planning to achieve each goal as well as the effort associated with actions and subgoals. These values are required by the "Select goal or action" component.

In the following sections, we describe in detail the extensions made to incorporate the above features.

5.2.2 Estimating duration

To determine the latest time by which actions should be executed in order to achieve one or more goals by their deadlines, it is necessary to estimate the duration of each action (i.e. to estimate how long it will take the agent to execute each action). An estimate of an action's duration is made initially when that action is created (during the step addition process). When any variables belonging to that action are further instantiated (during the simple establishment process) the duration associated with that action is reestimated. (Note: variables belonging to an action may also be further instantiated during conflict resolution, which means that the duration associated with that action should be reestimated as part of the conflict resolution process. However, this is not currently implemented.)

In some cases, if the action is not fully instantiated when it is created and added to the partial plan, it may not be possible to accurately determine how long it will take to execute that action. For example, the operator schema/action template *(travel ?x ?y)* represents the activity of travelling from location *?x* to location *?y*, where *?x* and *?y* are variables - *(travel ?x ?y)* is the *name* of the operator schema (see chapter 4, section 4.2.3 for a detailed description of the representation used for actions). *(travel ?x ?y)* has the precondition *(at ?x)* meaning that prior to executing *(travel ?x ?y)* the agent must be at some initial location *?x*, and the postcondition *(and (not (at ?x)) (at ?y))* meaning that once *(travel ?x ?y)* has been executed the agent is no longer at location *?x* but is at some other location *?y*. When the "Achieve goal" process achieves the goal or subgoal *(at shop)* using step addition, a new action *(travel ?x shop)* is created using the operator schema/ action template *(travel ?x ?y)* with the variable *?y* instantiated to *shop*. The variable *?x* remains uninstantiated until the partial plan is further refined - which means that at this initial stage it is not possible to accurately estimate the duration of the action. This problem is overcome by using domain knowledge in the form of a domain-designer supplied look-up table (see chapter 4, section 4.3.5 for a description of the look-up table) to assign a worst estimate (i.e. the longest period of time) of the time it will take to execute the action. For example, if the domain is a small town, we may know that it will take at most 20 minutes to travel the longest distance between two locations within that town. In this case, all incomplete instantiations of the action template *(travel ?x ?y)* such as *(travel ?x shop)* will be assigned the duration 20 minutes. (This is in contrast to DEVISER [Vere 83] where an action whose duration has not yet been determined is assigned a default

duration of 0.)

The look-up table stores estimates of the duration for different action instantiations - different variable bindings will affect the value assigned to the field *duration* associated with each action. For example, as we saw earlier, the value *duration* associated with the operator schema/action template *(travel ?x ?y)* will depend upon the variable bindings for *?x* and *?y* - the action *(travel home shop)* will have a different associated *duration* than the action *(travel post-office shop)*. The *duration* associated with each existing action must therefore be reestimated whenever that action is further instantiated (following simple establishment). In our earlier example, the *duration* associated with the action *(travel ?x shop)* was a worst estimate of 20 minutes as initially we did not know how the variable *?x* would be instantiated. The look-up table will contain estimates for different instantiations of *?x* - for example, the instantiation *(travel garage shop)* may have the estimated duration 15 minutes while the instantiation *(travel home shop)* may have the estimated duration 5 minutes. Once the action *(travel ?x shop)* is further instantiated following simple establishment to become the action *(travel garage shop)*, the value *duration* is reestimated to be 15 minutes using the look-up table (the predicate *(travel garage shop)* - the action's *name* - is used as a key for the look-up table to determine the value *duration*).

The algorithm required for this extension is described in table 5.2 and is used as part of both the step addition and simple establishment procedures. This algorithm assumes that the *name* field associated with actions contains parameters (which may be variables or constants) (for example *(stack ?x ?y)*).

Table 5.2 Assigning durations to actions

1. Let *g* be the goal/subgoal belonging to a partial plan *p* which is to be achieved.
2. Let *establisher* be a new or existing action (step addition and simple establishment respectively) chosen to achieve *g*. This action contains a predicate belonging to its associated set *add* which unifies with *g* with respect to the bindings belonging to the partial plan *p*.
3. Let *new-bindings* be the new set of bindings created during step addition/simple establishment (when the *add* predicate of the action *establisher* unifies successfully with the goal/subgoal *g* with respect to the *bindings* currently within the partial plan *p*, the resulting bindings are added to the set *bindings* currently belonging to *p*).
4. Instantiate the *name* field associated with the action *establisher* using the bindings *new-bindings* generated in stage 3. above.
5. The newly instantiated *name* field is used as a key to search for the value *duration* to be assigned to the action *establisher* (which is stored in a hashtable).

5.2.3 Estimating the degree of support

When an action is selected to achieve a goal or subgoal, it is necessary to estimate the degree to which that action, when executed, will support the agent's motivations. The degree to which each action supports the agent's motivations forms part of the "Evaluate partial plans" heuristic (described in more detail in chapter 6, section 6.4) which is used to evaluate and rank partial plans.

When creating a new action (as part of the step addition process), the fields *pros* and *cons* must be assigned (see chapter 4, section 4.2.3 for a descripton of the representation used for actions). These indicate an estimate of the degree to which executing the new action will support the agent's motivations (*pros*) and the degree to which executing the new action will undermine the agent's motivations (*cons*). Currently, values for these fields are determined by examining a domain-designer supplied look-up table (see chapter 4, section 4.3.5), which contains *pros* and *cons* values corresponding to the *name* field associated with each new action (the *name* of the action is the key which is used to search the look-up table for corresponding values for *pros* and *cons*). For example, for an action with the *name (pickup ?object)*, the *pros* and *cons* entry corresponding to the key *(pickup ?object)* is retrieved from the look-up table.

Table 5.3 Assigning *pros* and *cons* to actions

1. Let *g* be the goal/subgoal belonging to a partial plan *p* which is to be achieved.
2. Let *establisher* be a new or existing action (step addition and simple establishment respectively) chosen to achieve *g*. This action contains a predicate belonging to its associated set *add* which unifies with *g* with respect to the bindings belonging to the partial plan *p*.
3. Let *new-bindings* be the new set of bindings created during step addition/simple establishment (when the *add* predicate of the action *establisher* unifies successfully with the goal/subgoal *g* with respect to the *bindings* currently within the partial plan *p*, the resulting bindings are added to the set *bindings* currently belonging to *p*).
4. Instantiate the *name* field associated with the action *establisher* using the bindings *new-bindings* generated in stage 3. above.
5. The newly instantiated *name* field is used as a key to search for the values *pros* and *cons* to be assigned to the action *establisher* (which is stored in a hashtable).

In section 5.2.2 above we saw how the value *duration* assigned to an action depended upon how that action was instantiated - the action *(travel ?x shop)* was estimated to take 20 minutes to execute. Once that action was further instantiated (binding the variable *?x* to the value *garage*) to become *(travel garage shop)*, it was assigned a different estimate of the value *duration* - the value 15 minutes. In the same way, the values *pros* and *cons* assigned to an action depend upon how that action is instantiated. For

example, the action *(dine ?restaurant)* which involves eating in some restaurant where *?restaurant* is a variable, may support or undermine an agent's motivations differently depending upon in which restaurant the agent may be eating. A look-up table stores estimates of the values for *pros* and *cons* for different variable instantiations. For example, the look-up table will store an estimate of the degree to which executing *(dine ?restaurant)* supports/undermines the agent's motivations. When the action is further instantiated as a consequence of simple establishment, the values *pros* and *cons* are reestimated to reflect the instantiation - for example, if the variable *?restaurant* is later bound to the value *tandoori_kitchen*, the values *pros* and *cons* are updated (the predicate *(dine tandoori_kitchen)* - the action's *name* - is used as a key).

The algorithm required for this extension is described in table 5.3 and is used as part of both the step addition and simple establishment procedures. Again, this algorithm assumes that the *name* field associated with actions contains parameters (which may be variables or constants) (for example *(puton ?x ?y)*).

5.2.4 Maintaining a record of the goals to which an action contributes

An important part of both the step addition and simple establishment algorithms is concerned with maintaining a record of which goals each action contributes towards. This both enables the values *importance* and *effort* associated with actions to be determined and, should there be insufficient time available to achieve one or more goals, facilitates the removal of goals and their associated actions and constraints (this is the responsibility of the "Edit the partial plan" component). In this section we describe some of the mechanisms used to keep track of the relationship between actions and goals. These procedures play a key role in both step addition and simple establishment.

Figure 5.1 illustrates a partial plan consisting of actions that have been selected to achieve the goals *g1*, *g2* and *g3* which have been created by the "Generate/update goals" process. There are six actions, *a1*,...*a6*. The arrows between actions indicate partial orderings - for example, the action *a4* occurs prior to the action *a1*. The action *a1* was selected by step addition to achieve the goal *g1* (i.e. one of the effects of executing *a1* is to establish the goal *g1*), while the action *a4* was selected by step addition to achieve the subgoal/precondition *p11* associated with *a1*. The action *a3* was selected by step addition to achieve the goal *g3*, while the action *a6* was selected by step addition to achieve the subgoal/precondition *p31* associated with *a3*. Finally, the action *a2* was selected by

step addition to achieve the goal *g2*, the action *a5* was selected by step addition to achieve the subgoal/precondition *p21* associated with *a2*, while the action *a6* was selected by simple establishment to achieve the subgoal/precondition *p22* associated with *a2*. Each action must maintain a record of the goals to which it contributes. For example, the actions *a1* and *a4* both only contribute towards the achievement of the goal *g1*, the action *a3* only contributes towards the achievement of the goal *g3*, actions *a2* and *a5* both only contribute towards the achievement of the goal *g2*, while the action *a6* contributes towards both the achievement of the goals *g2* and *g3*. Should there be insufficient time available to achieve the goal *g1*, the plan may be edited to remove the goal *g1*, together with its associated actions *a1* and *a4*.

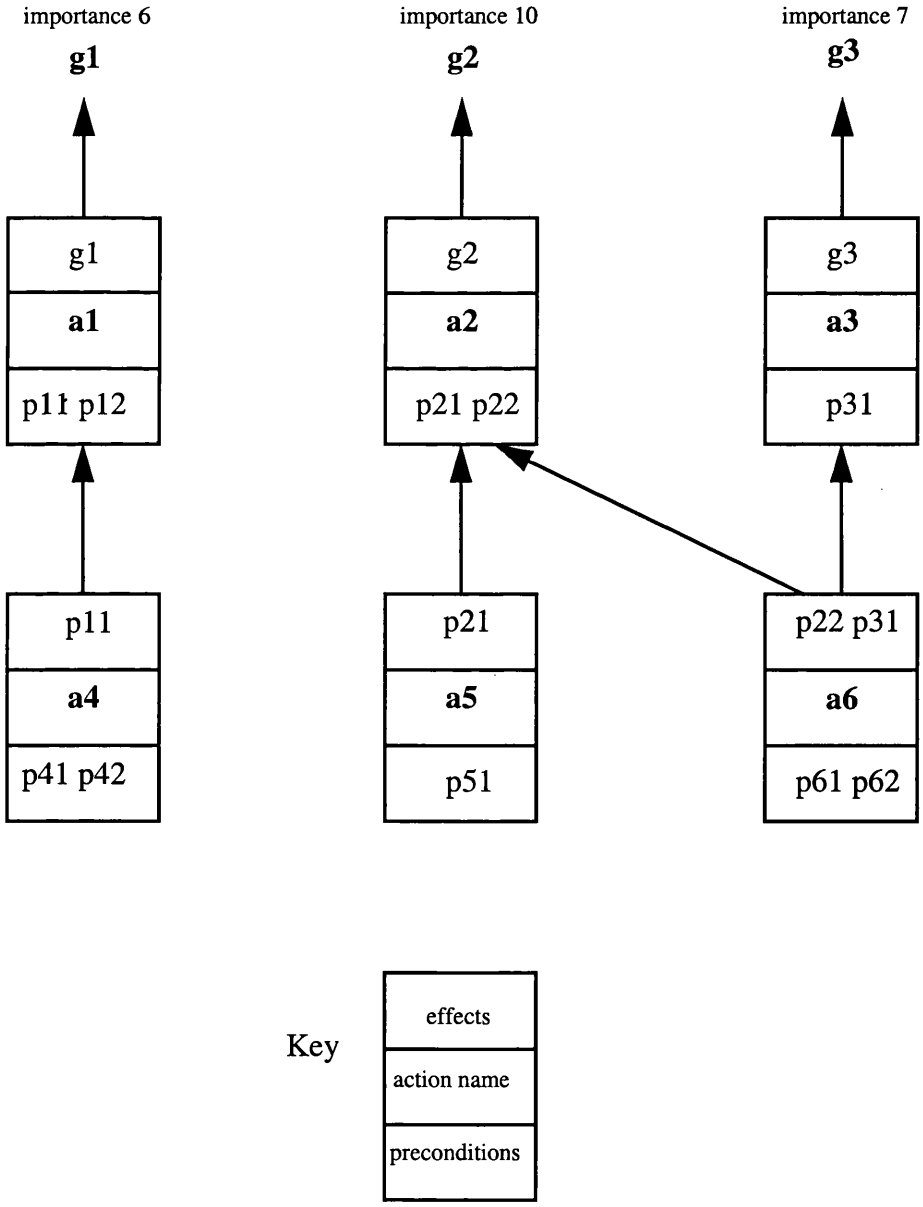


Figure 5.1 A Partial Plan

The "Achieve goal" process extends both the step addition and simple establishment

procedures used by SNLP in order to maintain a record of the goals to which each action contributes. Sections 5.2.7 and 5.2.8 describe this process in more detail.

5.2.5 Calculating the *importance* of actions

Maintaining a record of the goals to which each action contributes also enables us to calculate the *importance* of executing each action, as well as the amount of *effort* associated with each action. When goals are generated by the "Generate/update goals" procedure, they are assigned a value which indicates how important it is to achieve them. In figure 5.1, *g1* has the associated *importance* 6, *g2*, the value 10, and *g3* the value 7. In order to calculate how important it is to execute an action, it is necessary to take into account the values indicating the *importance* of achieving each goal to which that action contributes. The value indicating the *importance* of executing an action is obtained by summing the values indicating the *importance* of each goal to which that action contributes. For example, in figure 5.1, the value indicating the *importance* of action *a1* is 6, i.e. the same value as that assigned to the goal *g1* which is the only goal to which *a1* contributes. The value indicating the *importance* assigned to the action *a6* however, is 17, the value obtained by adding 10 and 7 (the values indicating the *importance* of the goals *g2* and *g3* respectively - the goals to which *a6* contributes).

The "Achieve goal" process extends both the step addition and simple establishment procedures used by SNLP in order to calculate the importance of actions and their preconditions. Sections 5.2.7 and 5.2.8 describe this process in more detail.

5.2.6 Calculating *effort*

Each goal has an associated value indicating the amount of *effort* which has been expended by the "Achieve goal" process in achieving that goal. For example, in figure 5.1, the amount of effort expended by the planner in achieving the goal *g1* is 2 - indicating that two goal achievement cycles have been invoked: the action *a1* was selected by step addition to achieve the goal *g1*; the action *a4* was selected by step addition to achieve the subgoal/precondition *p11* of the action *a1* (as we saw earlier, the actions *a1* and *a4* only contribute towards the achievement of the goal *g1*). The amount of effort expended by the planner in achieving the goal *g3* is also 2: the action *a3* was selected by step addition to achieve the goal *g3*; the action *a6* was selected by step addition to achieve the subgoal/precondition *p31* of the action *a3*. Finally, the amount of effort

expended by the planner in achieving the goal *g2* is 3: the action *a2* was selected by step addition to achieve the goal *g2*; the action *a5* was selected by step addition to achieve the subgoal/precondition *p21* of the action *a2*; the action *a6* was selected by simple establishment to achieve the subgoal/precondition *p22* of the action *a2*.

In order to determine the value *effort* associated with each action and its preconditions, it is necessary to take into account the amount of effort expended in achieving each goal to which the action contributes - the value *effort* assigned to an action is determined by summing the values *effort* associated with each goal to which the action contributes. For example, the value *effort* assigned to both the actions *a1* and *a4* in figure 5.1 is 2 as both actions contribute only towards the goal *g1*. The value *effort* assigned to the action *a3* is 2 as *a3* only contributes towards the goal *g3*. The value *effort* assigned to the actions *a2* and *a5* is 3 as both actions only contribute towards the goal *g2*. However, the value *effort* assigned to the action *a6* is 5 as *a6* contributes towards two goals, *g2* and *g3*.

The "Achieve goal" process extends both the step addition and simple establishment procedures used by SNLP in order to calculate the effort associated with goals, actions and their preconditions. Sections 5.2.7 and 5.2.8 describe this process in more detail.

5.2.7 Step addition

The step addition process has been extended as illustrated in table 5.4. Note that the algorithm presented in table 5.4 lists extensions to the step addition process used by SNLP (i.e. the SNLP step addition procedures are not described in this algorithm). In the following sections we describe some of these extensions in greater detail.

Maintaining a record of the goals to which an action contributes

When creating a new action to achieve a goal/subgoal *g*, a part of the process involves determining which goals the new action contributes towards (see section 5.2.4 above and stage 3 in table 5.4). Once constraints have been posted to ensure that a newly created action *newact* achieves a goal/subgoal *g*, the new action *newact* contributes towards or establishes *g* as well as all goals to which achieving *g* contributes. For example, figure 5.2 illustrates the partial plan that arises through adding a new action *a7* to the partial plan featured in figure 5.1 in order to achieve the subgoal *p61* (a precondition belonging to the action *a6*). In figure 5.2 we see that achieving the subgoal *p61* will contribute towards the achievement of the goals *g2* and *g3* (executing the action *a6* will contribute

towards the achievement of *g2* and *g3* so, as *p61* is a precondition of *a6*, achieving *p61* also contributes towards the achievement of *g2* and *g3*). This means that the new action

Table 5.4 Extensions to the step addition procedure

1. Let *g* be a goal/subgoal belonging to a partial plan *p* which is to be achieved by step addition (this is chosen by the "Select goal or action" procedure).
2. Let *bindings* be the set of new bindings. These are created by first successfully unifying an *add* predicate belonging to an operator schema/action template *templ* with the goal/subgoal *g* (with respect to the bindings currently belonging to *p*), and then adding the resulting bindings to the plan bindings currently belonging to *p*.
3. Let *newgoals* be the set of goals to which achieving the goal/subgoal *g* contributes (see section 5.2.7, "Maintaining a record of the goals to which an action contributes").
4. Update the set of *goals* belonging within the partial plan *p*. The value *effort* associated with each goal belonging to the set *newgoals* needs updating to reflect the fact that the planner has expended extra effort in achieving these goals (see section 5.2.7, "Updating the value *effort* associated with goals").
5. Create a new action *newact* (using the operator schema/action template *templ*) to achieve *g* - using the following procedures.
   (a) Create the fields *name*, *precond*, *delete* and *add* associated with *newact* (by using *bindings* to instantiate the corresponding fields belonging to the action template *templ*) - this is part of the SNLP step addition process.
   (b) Use *bindings* to estimate the value *duration* associated with *newact* (as described in section 5.2.2).
   (c) Use *bindings* to estimate the values *pros* and *cons* associated with *newact* (as described in section 5.2.3).
   (d) The *goals* field associated with the newly created action *newact* is assigned the value *newgoals*.
   (e) The value *importance* associated with the newly created action *newact*, is determined by adding together the values indicating the *importance* associated with each goal belonging to the set *newgoals* (see section 5.2.7, "Estimating the importance associated with the newly created action").
   (f) The value *effort* associated with the newly created action *newact*, is determined by adding together the newly generated values *effort* associated with each goal (see stage 4.) belonging to the set *newgoals* (see section 5.2.7, "Assigning the value *effort* to the newly created action").
6. Update the set of actions belonging within the partial plan *p*. First, add the newly created action *newact* to the set of actions belonging to p. Then, the value *effort* associated with each action that contributes towards the achievement of any of the goals belonging to the set *newgoals* needs to be updated to reflect the fact that the "Achieve goal" process has expended extra effort in achieving these goals (this is described in section 5.2.7, "Reestimating the value *effort* associated with existing actions").

a7, when executed, also contributes towards achieving the goals *g2* and *g3* because the goal *p61* is achieved by executing *a7*. The field *goals* associated with the new action *a7* is therefore assigned a set containing the goals *g2* and *g3*. For further discussion, the set of goals to which a newly created action contributes is referred to as the set *newgoals*.

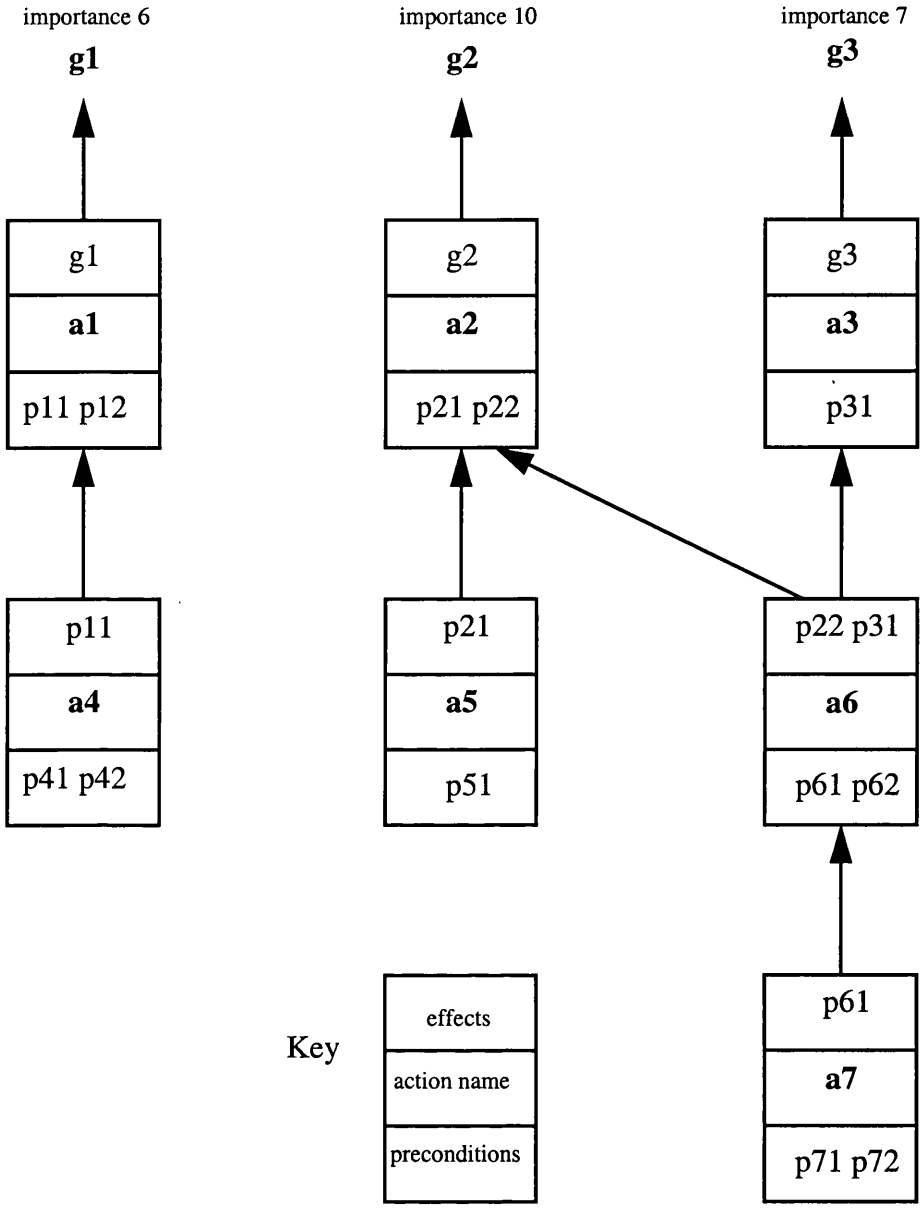


Figure 5.2 A Partial Plan following Step Addition

Updating the value *effort* associated with goals

Once a new action has been selected to achieve a goal/subgoal *g*, the value *effort* associated with each goal to which achieving *g* contributes (i.e. each goal belonging to the set *newgoals*), must be incremented to reflect the extra effort expended by the "Achieve goal" component in achieving those goals). For example, figure 5.1 shows that prior to adding the newly created action *a7* to the partial plan, the value indicating the *effort* expended in achieving the goal *g2* was 3 while the value indicating the *effort* expended in achieving *g3* was 2. Following step addition, the value indicating the *effort* expended in achieving *g2* is 4 while the value indicating the *effort* expended in achieving *g3* is 3.

Estimating the importance associated with the newly created action

The value *importance* that is assigned to the newly created action *newact* is the sum of the values indicating the *importance* of achieving each goal to which *newact* contributes. In the example, the value indicating the *importance* of executing *a7* is 17 - the value obtained by adding 10 (the value indicating the *importance* of achieving *g2*) and 7 (the value indicating the *importance* of achieving *g3*).

Assigning the value *effort* to the newly created action

The value *effort* that is assigned to the newly created action *newact* is the sum of the values indicating the *effort* expended in achieving each goal to which *newact* contributes (i.e. the sum of the values indicating the *effort* expended in achieving each goal belonging to the set *newgoals*). For example, in figure 5.2, the newly created action *a7* contributes towards the achievement of the goals *g2* and *g3*. The value *effort* assigned to *a7* is 7 - the value obtained by adding 4 (the *effort* expended in achieving *g2*) and 3 (the *effort* expended in achieving *g3*).

Reestimating the value *effort* associated with existing actions

Following step addition, the value indicating the *effort* expended in achieving each goal belonging to the set *newgoals* is incremented to reflect the extra work undertaken by the "Achieve goal" component. This means that the value indicating the *effort* associated with each existing action that already contributes towards one or more goals belonging to the set *newgoals* must be reestimated. For example, prior to step addition, in figure 5.1 the value *effort* associated with both *a2* and *a5* is 3 - following step addition (see figure 5.2), the *effort* associated with both *a2* and *a5* is now 4 (the value indicating the *effort* expended in achieving *g2*, the goal to which they both contribute, has been incremented) which reflects the fact that a new action *a7* has been added to the plan (*a7* contributes towards the goals *g2* and *g3*). Likewise, the value indicating the *effort* associated with *a3* is now 3 (the value indicating the *effort* expended in achieving *g3*, the goal to which *a3* contributes, has been incremented) and the value indicating the *effort* associated with *a6* is now 7 (obtained by adding 4, the new value *effort* associated with *g2*, to 3, the new value *effort* associated with *g3*).

5.2.8 Simple establishment

The simple establishment process has been extended as shown in table 5.5. The algorithm presented in table 5.5 lists extensions to the simple establishment process used by SNLP (i.e. the SNLP simple establishment procedures are not described in this table). In the following sections we describe some of these extensions in greater detail.

Table 5.5 Extensions to the simple establishment procedure

1. Let *g* be a goal/subgoal belonging to a partial plan *p* which is to be achieved by simple establishment (*g* is chosen by the "Select goal or action" procedure).
2. Let *bindings* be the set of new bindings. These are created by first successfully unifying an *add* predicate associated with an action *act* (which belongs to *p*) with the goal/subgoal *g* (with respect to the bindings currently belonging to *p*), and then adding the resulting bindings to the plan bindings currently belonging to *p*.
3. Let *newgoals* be the set of goals to which achieving the goal/subgoal *g* contributes (see "Maintaining a record of the goals to which each action contributes" in section 5.2.8).
4. Let *newestablishers* be the set of actions that, as a consequence of simple establishment, newly contribute towards the set of goals *newgoals* (see "Determining the actions which newly establish goals" in section 5.2.8).
5. Update the set of *goals* belonging within *p*. The value *effort* associated with each goal belonging to the set *newgoals* needs updating to reflect the fact that the planner has expended extra effort in achieving these goals (see "Updating the value *effort* associated with goals" in section 5.2.8).
6. Update the set of actions belonging within the partial plan *p* - using the following procedures.
   (a) Use *bindings* to further instantiate the fields *name*, *precond*, *delete* and *add* associated with each action - this is part of the SNLP simple establishment process.
   (b) Use *bindings* to reestimate the value *duration* associated with each action (as described in section 5.2.2).
   (c) Use *bindings* to reestimate the values *pros* and *cons* associated with each action (as described in section 5.2.3).
   (d) The *goals* field associated with each action belonging to the set *newestablishers* is updated to reflect the fact that these actions also now contribute towards the goals belonging to the set *newgoals* (see "Updating the set of goals associated with actions" in section 5.2.8).
   (e) The value *importance* associated with each action belonging to the set *newestablishers* is reevaluated to reflect the fact that these actions now also contribute towards the achievement of each goal belonging to the set *newgoals* (see "Reevaluating the importance of actions" in section 5.2.8).
   (f) The value *effort* associated with each action belonging to the set *newestablishers* is recalculated to reflect the fact that these actions now also contribute towards the achievement of each goal belonging to the set *newgoals* (see "Reevaluating the value *effort* associated with actions" in section 5.2.8).
   (g) The value *effort* associated with each action that contributes towards the achievement of one or more of the goals belonging to the set *newgoals* (but which is not a member of the set *newestablishers*) needs to be updated to reflect the fact that the planner has expended extra effort in achieving these goals (see "Reevaluating the value *effort* associated with actions" in section 5.2.8).

Maintaining a record of the goals to which each action contributes

As with step addition, an important part of the simple establishment algorithm is concerned with maintaining a record of which goals (created by the "Generate/update goals" component) each action contributes towards. This enables values indicating the *importance* and *effort* associated with actions to be recalculated and facilitates the removal of goals and their associated constraints should there be insufficient time (this is the responsibility of the "Edit the partial plan" procedure).

When selecting an existing action *act* to achieve a goal/subgoal *g* (simple establishment), a part of the process involves determining which new/extra goals the existing action *act* will contribute towards. Once constraints have been posted to ensure that the existing action *act* achieves the goal/subgoal *g*, the action *act* newly contributes towards or establishes *g* as well as all goals to which achieving *g* contributes. For example, figure 5.3 illustrates the partial plan that arises through adding constraints to the partial plan of figure 5.2 so that the existing action *a2* (belonging to the partial plan featured in figure 5.2) achieves the subgoal *p12* (a precondition belonging to the action *a1*). In figure 5.2 we see that prior to adding the constraints, the action *a2* contributed solely towards the achievement of the goal *g2*. Following simple establishment, in figure 5.3 it can be seen that the action *a2* newly contributes towards achieving the goal *g1* (executing the action *a2* will achieve the precondition *p12* which contributes towards the achievement of *g1*). Determining the set of goals to which the existing action *act* newly contributes is a part of the simple establishment process (see table 5.5, stage 3). For further discussion, the set of goals to which the existing establisher/action *act* newly contributes is referred to as the set *newgoals*.

Determining the actions which newly establish goals

In addition to the existing action *act* newly contributing towards the achievement of the set of goals *newgoals* (as described in the previous section) each existing action that contributes towards the achievement of the preconditions of *act* now also newly contributes towards the set of goals *newgoals*. For example, as we saw in figure 5.3, following simple establishment, the existing action *a2* newly contributes towards the goal *g1*. In addition, the actions *a5*, *a6* and *a7*, as well as *a2*, newly contribute towards the goal *g1* (*a5* establishes the precondition *p21* of the action *a2*, *a6* establishes the precondition *p22* of *a2* and *a7* establishes the precondition *p61* of the action *a6*). An important part of the

simple establishment process involves determining the set of actions that newly contribute towards the goals belonging to the set *newgoals*. For further discussion we refer to this set of actions (which includes the establisher/action *act*) as the set *newestablishers*. In our example, the set *newestablishers* contains the actions *a2*, *a5*, *a6* and *a7*.

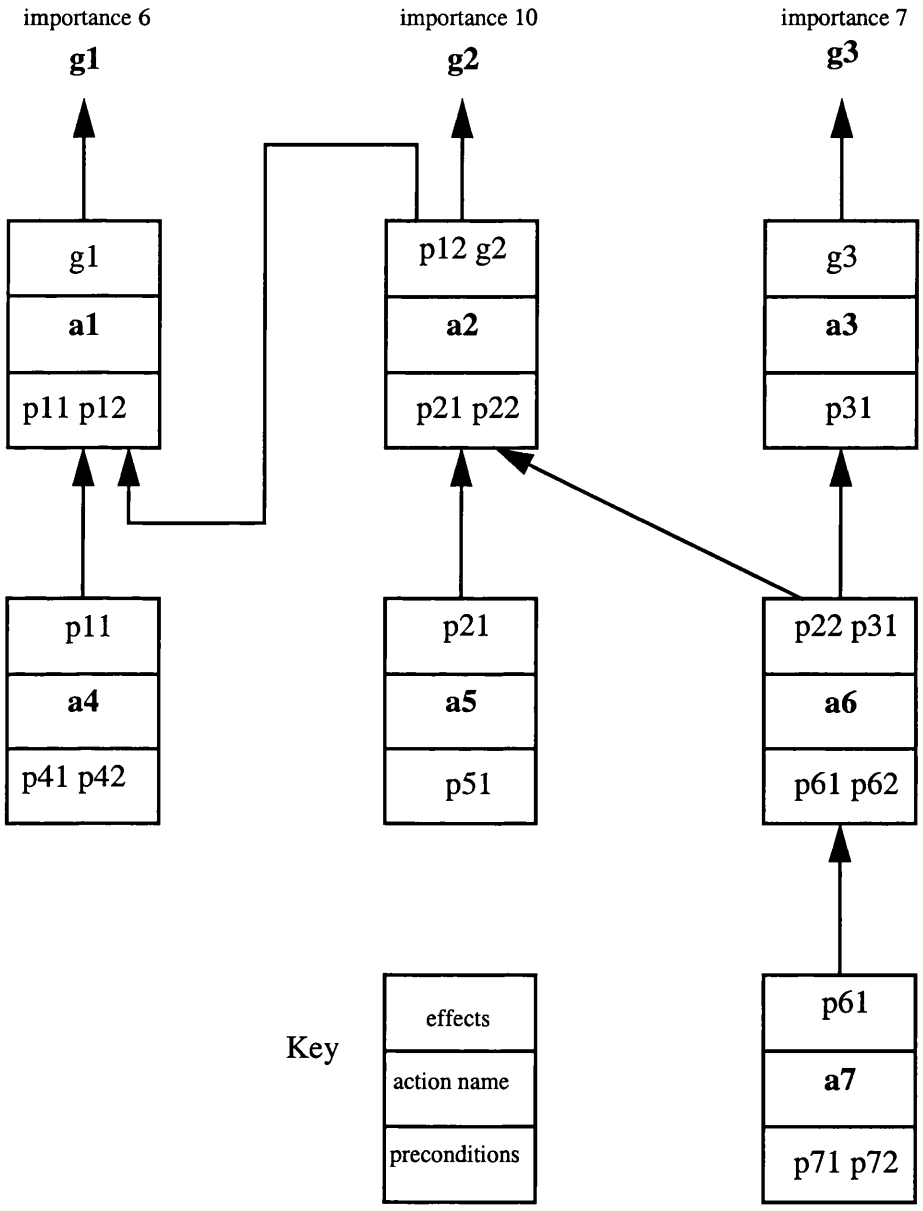


Figure 5.3 A Partial Plan following Simple Establishment

Updating the value *effort* associated with goals

Once an existing action *act* has been selected to achieve a goal/subgoal *g*, the value *effort* associated with each goal to which achieving *g* contributes (i.e. each goal belonging to the set *newgoals*), must be incremented to reflect the extra effort expended by the "Achieve goal" component). For example, figure 5.2 shows that prior to adding con-

straints to ensure the existing action *a2* will achieve the precondition *p12*, the effort expended in achieving the goal *g1* is 2. Following simple establishment, the effort expended in achieving *g1* is now 3.

Updating the set of goals associated with each action

The *goals* field associated with each new establisher (i.e. each action belonging to the set *newestablishers*) needs to be updated to reflect the fact that the new establisher/action now newly contributes towards the achievement of the set of goals *newgoals*. The new value assigned to *goals* is the union of the original set *goals* with the set *newgoals*. For example, in figure 5.2 the action *a2* originally contributed towards the goal *g2* (i.e. the set *goals* associated with *a2* contained the goal *g2*). Following simple establishment, *a2* newly contributes towards the goal *g1* (*newgoals* is a set containing the goal *g1*). The field *goals* assigned to *a2* following simple establishment is a set containing the goals *g1* and *g2* (this value is also assigned to the field *goals* associated with the action *a5*).

Reevaluating the importance of actions

When the simple establishment process posts constraints to ensure that an existing action *act* will achieve a goal *g*, the *importance* associated with *act*, as well as the *importance* associated with all actions that newly contribute towards or establish *g* (i.e. the *importance* associated with each action belonging to the set *newestablishers*) changes and so must be reevaluated. The example in the previous section may be used to illustrated this change. Prior to simple establishment (i.e. figure 5.2) the actions *a2* and *a5* contribute towards the achievement of the goal *g2* while the actions *a6* and *a7* contribute towards the achievement of the goals *g2* and *g3*. If *g2* has the importance value 10 and *g3* has the importance value 7, then actions *a2* and *a5* have the importance value 10 while actions *a6* and *a7* have the importance value 17. Following simple establishment, actions *a2*, *a5*, *a6* and *a7* also establish or newly contribute towards the achievement of the goal *g1*. Their associated value indicating their importance therefore changes - in fact it increases to reflect the fact that they now also establish or contribute towards the achievement of the goal *g1*. If *g1* has the importance 6 then the actions *a2* and *a5* now have the importance value 16 - the total obtained by adding the importance associated with the goals *g1* and *g2*. Actions *a6* and *a7* now have the importance value 23 - the total obtained by adding the importance values associated with the goals *g1*, *g2* and *g3*. The simple establish-

ment process has been extended to reevaluate the importance associated with all actions that newly establish or contribute towards the achievement of the goal *g1*.

Reevaluating the value *effort* associated with actions

When a goal or subgoal *g* is achieved by simple establishment, the value *effort* associated with all goals to which *g* contributes (i.e. all goals belonging to the set *newgoals*) is updated (incremented). For example, in figure 5.3 we see that achieving the subgoal *p12* contributes towards the achievement of the goal *g1*. Following simple establishment the value *effort* associated with *g1* is incremented to reflect the fact that extra effort has been expended in planning to achieve this goal. (Prior to simple establishment, the value *effort* associated with *g1* was 2 - once simple establishment has taken place, the value *effort* associated with *g1* is incremented to become 3).

In addition, the value *effort* associated with each action that newly (i.e. following simple establishment) contributes (i.e. each action belonging to the set *newestablishers*) towards the achievement of (or which newly establishes) *g*, and the value *effort* associated with each action that already (i.e. prior to simple establishment) contributes towards the achievement of one or more goals belonging to the set *newgoals* (these are action that do not belong to the set *newestablishers*), needs to be updated. For example, prior to simple establishment (i.e. figure 5.2) the actions *a2* and *a5* contribute towards the achievement of the goal *g2* while the actions *a6* and *a7* contribute towards the achievement of the goals *g2* and *g3*. *g2* has the *effort* value 4 prior to simple establishment while *g3* has the *effort* value 3, which means that actions *a2* and *a5* have the *effort* value 4 while actions *a6* and *a7* have the *effort* value 7 (in section 5.2.6 we saw that the value *effort* associated with an action is the total obtained by adding the *effort* values associated with each goal to which the action contributes). Following simple establishment, actions *a2*, *a5*, *a6* and *a7* also establish or newly contribute towards the achievement of the goal *g1* (these actions belong to the set *newestablishers*). Their associated value for *effort* therefore changes - it increases to reflect the fact that extra effort has been expended by the planner in achieving the goal *g1*. Following simple establishment, *g1* has the value 3 for *effort* which means the actions *a2* and *a5* now have the value 7 for *effort* - the total obtained by adding the values for *effort* associated with the goals *g1* and *g2*. Actions *a6* and *a7* now have the value 10 for *effort* - the total obtained by adding the values for *effort* associated with the goals *g1*, *g2* and *g3*. In addition, the actions *a1* and *a4* already con-

tributed towards the achievement of the goal *g1* prior to simple establishment and had the value 2 for *effort*. The value *effort* associated with *a1* and *a4* must also be recalculated following simple establishment - the resulting value for *effort* associated with actions *a1* and *a4* is 3 (both actions contribute solely towards the achievement of the goal *g1*, whose value *effort* was incremented during simple establishment).

5.3 Conflict Resolution

The algorithms used to test for unsafe links and to resolve conflicts are those used by the planner SNLP - no further extensions have been implemented. In the following sections we describe two possible extensions.

5.3.1 Updating the value *effort* associated with goals

A goal/subgoal *g* is achieved by calling step addition and simple establishment procedures. A part of the "Achieve goal" procedure involves resolving any potential conflicts by posting constraints to protect any unsafe links within each newly generated plan (this is known as conflict resolution). Currently, the extra work involved in protecting such links is not reflected in the value *effort* associated with the goal/subgoal *g's* corresponding goals (these are the goals to which achieving *g* contributes - i.e. the set *newgoals*). The conflict resolution procedure could be extended so that the value *effort* associated with each of the goal/subgoal *g's* goals (i.e. each goal belonging to the set *newgoals*) is incremented for each unsafe link requiring protection.

5.3.2 Updating the values *pros, cons* and *duration*

During the separation process, new bindings (in the form of both codesignation and non-codesignation constraints) are added to the partial plan. These bindings may be used to further instantiate the *name* field associated with each action which may mean that each action's associated *pros*, *cons* and *duration* fields should be updated to reflect this further instantiation. This feature is not currently implemented.

5.4 A Truck World Example

5.4.1 Achieving a goal

In this section we describe how the planning/execution architecture achieves a goal in the

truck world domain described in chapter 4, section 4.4. In order to achieve a goal, the following information is required which was described in detail in chapter 4, section 4.4.

1. A description of the initial state which is encapsulated within an initial plan (see chapter 4, section 4.4.1, tables 4.1 and 4.2)
2. A description of the goal state (see chapter 4, section 4.4.2, table 4.3).
3. A set of operator schemas/action templates - these represent the capabilities of the truck-driver (see chapter 4, section 4.4.3, table 4.4).
4. A look-up table specifying the amount of time it takes the truck-driver to execute each action (see chapter 4, section 4.4.5, table 4.6).
5. A look-up table specifying the degree to which executing each action will support or undermine the truck-driver's motivations (see chapter 4, section 4.4.6, table 4.7).

Table 5.6 Actions used to achieve the goal *at(package city5)*

id:	3	6
name:	*load-truck(package truck city1)*	*drive-truck(truck city1 city3)*
precondition:	*at(truck city1) & at(package city1)*	*at (truck city1) & has-fuel(truck) & connects(city1 city3)*
delete:	*at(package city1)*	*at(truck city1)*
add:	*in(package truck)*	*at(truck city3)*
goals:	*(1)*	*(1)*
pros:	*nil*	*(pleasure 1.9)*
cons:	*nil*	*(conserve-fuel 0.3) & (conserve-tyres 0.1)*
duration:	*1*	*2*
importance:	*6*	*6*
effort:	*14*	*14*

Given the initial plan shown in chapter 4, section 4.4.1, table 4.2, and the goal *at(package city5)* shown in chapter 4, section 4.4.2, table 4.3, the "Select goal or action" component chooses to achieve the goal *at(package city5)* as there are no executable actions in the initial plan and no other goals. Tables 5.6, 5.7 and 5.8 illustrate the actions which are selected (using step addition, simple establishment and conflict resolution) to achieve the goal *at(package city5)* by its deadline 20 (see chapter 4, section 4.4.2 table 4.3), assuming the current time is 0, while table 5.9 shows the resulting partial plan which contains

these actions. The "Achieve goal" component requires the initial plan which is shown in

Table 5.7 Actions used to achieve the goal *at(package city5)*

id:	5	4
name:	*drive-truck(truck city3 city4)*	*drive-truck(truck city4 city5)*
precondition:	*at(truck city3) & has-fuel(truck) & connects(city3 city4)*	*at(truck city4) & has-fuel(truck) & connects(city4 city5)*
delete:	*at(truck city3)*	*at(truck city4)*
add:	*at(truck city4)*	*at(truck city5)*
goals:	*(1)*	*(1)*
pros:	*(pleasure 1.2)*	*(pleasure 1.8)*
cons:	*(conserve-fuel 0.3) & (conserve-tyres 0.4)*	*(conserve-fuel 0.3) & (conserve-tyres 0.4)*
duration:	*3*	*3*
importance:	*6*	*6*
effort:	*14*	*14*

chapter 4, section 4.4.1, table 4.2 as well as the operator schemas/action templates which are shown in chapter 4, section 4.4.3, table 4.4. The step addition and simple establish-

Table 5.8 Actions used to achieve the goal *at(package city5)*

id:	2
name:	*unload-truck(package truck city5)*
precondition:	*at(truck city5) & in(package truck)*
delete:	*in(package truck)*
add:	*at(package truck)*
goals:	*(1)*
pros:	*nil*
cons:	*nil*
duration:	*1*
importance:	*6*
effort:	*14*

ment procedures require the two look-up tables shown in chapter 4, section 4.4.5, table 4.6 and chapter 4, section 4.4.6, table 4.7 which store values estimating how long it takes

to execute various actions, and values indicating the degree to which each action may support or undermine the agent's motivations, respectively. These tables, which are

Table 5.9 Plan which achieves the goal *at(package city5)*

actions:	id:	name:		
	3 *6* *5* *4* *2*	*load-truck(package truck city1)* *drive-truck(truck city1 city3)* *drive-truck(truck city3 city4)* *drive-truck(truck city4 city5* *unload-truck(package truck city5)*		
links	establisher:	condition:		consumer:
	6 *5* *4* *3* *2* *0* *0* *0* *0* *0* *0* *0* *0*	*at(truck city3)* *at(truck city4)* *at(truck city5)* *in(package truck)* *at(package city5)* *at(package city1)* *at(truck city1)* *connects(city4 city5)* *has-fuel(truck)* *connects(city3 city4)* *has-fuel(truck)* *connects(city1 city3)* *has-fuel(truck)* *at(truck city1)*		*5* *4* *2* *2* *1* *3* *3* *4* *4* *5* *5* *6* *6* *6*
open:	*nil*			
unsafe:	*nil*			
ordering:	*load-truck(package truck city1) -> drive-truck(truck city1 city3) -> drive-truck(truck city3 city4) -> drive-truck(truck city4 city5 -> unload-truck(package truck city5)* *(3 -> 6 -> 5 -> 4 -> 2 -> 1)*			
bindings:	*<varset {(?from6 ?loc3 city1)} = city1 not(?to6 ?truck6 ?ob3 ?truck3)>* *<varset {(?truck5 truck ?truck2 ?truck3 ?truck4 ?truck6)} = truck not(?from5 ?to5 ?ob2 ?loc2 ?loc3 ?ob3 ?to4 ?from4 ?to6 ?from6)>* *<varset {(city3 ?from5 ?to6)} = city3 not(?truck5 ?to5 ?truck6 ?from6)>* *<varset {(city4 ?from4 ?to5)} = city4 not(?truck4 ?to4 ?truck5 ?from5)>* *<varset {(?loc2 city5 ?to4)} = city5 not(?ob2 ?truck2 ?truck4 ?from4)>* *<varset {(?ob2 package ?ob3)} = package not(?truck2 ?loc2 ?loc3 ?truck3)>*			
goals:	id:	conditions:	deadline:	importance:
	1	*at(package city5)*	*20*	*6*

accessed using an action's *name* field, are used in order to assign values to the fields *pros*, *cons* and *duration* associated with each action as shown in tables 5.6, 5.7 and 5.8. In addition, the step addition and simple establishment procedures assign values to the fields *importance* and *effort* associated with each action. In this example, only the goal

at(package city5) has been achieved which has the value 6 indicating its importance. All actions in the plan (see tables 5.6, 5.7 and 5.8) contribute only towards this goal (the field *goals* assigned to each action contains the unique identifier, 1, associated with the goal *at(package city5)*) and are therefore assigned the value 6 indicating their importance. The value *effort* indicates how many "Achieve goal" cycles have occurred in order to achieve the goal *at(package city5)* - i.e. this value indicates the number of goals and subgoals that have been achieved (see the plan links shown in table 5.9) using either step addition or simple establishment. The sequence of actions selected in order to achieve *at(package city5)* are in the order *load(package truck city1)*, *drive-truck(truck city1 city3)*, *drive-truck(truck city3 city4)*, *drive-truck(truck city4 city5)* and *unload(package truck city5)* as shown in table 5.9.

5.4.2 Another example of achieving a goal

Table 5.10 Plan 2 - following execution of *drive-truck(truck city3 city4)*

actions:	id:	name:		
	0 *4* *2*	*initial* *drive-truck(truck city4 city5)* *unload-truck(package truck city5)*		
links	establisher:	condition:		consumer:
	0 *4* *0* *2* *0* *0*	*at(truck city4)* *at(truck city5)* *in(package truck)* *at(package city5)* *connects(city4 city5)* *has-fuel(truck)*		*4* *2* *2* *1* *4* *4*
open:	*nil*			
unsafe:	*nil*			
ordering:	*drive-truck(truck city4 city5 -> unload-truck(package truck city5)* *(4 -> 2 -> 1)*			
bindings:	*<varset {(truck ?truck2 ?truck4)} = truck not(?ob2 ?loc2 ?to4 ?from4)>* *<varset {(city4 ?from4)} = city4 not(?truck4 ?to4)>* *<varset {(?loc2 city5 ?to4)} = city5 not(?ob2 ?truck2 ?truck4 ?from4)>* *<varset {(?ob2 package)} = package not(?truck2 ?loc2)>*			
goals:	id:	conditions:	deadline:	importance:
	1 *7*	*at(package city5)* *at(parcel city5)*	*20* *16*	*6* *20*

Once the actions *load-truck(package truck city1)*, *drive-truck(truck city1 city3)* and *drive-truck(truck city3 city4)*, (see tables 5.6 and 5.7) which belong to the plan illustrated

in table 5.9 have been executed, the resulting partial plan is shown in table 5.10. Execu-

Table 5.11 Initial action for Plan 2

id:	name:	add:
0	*initial*	*connects(city1 city2) & connects(city2 city1) & connects(city1 city3) & connects(city3 city1) & connects(city2 city4) & connects(city4 city2) & connects(city2 city5) & connects(city5 city2) & connects(city3 city4) & connects(city4 city3) & connects(city4 city5) & connects(city5 city4) & has-fuel(truck) & at(truck city4) & in(package truck) & at(parcel city2)*

tion has resulted in the expected outcome which means that the current time is now 6. The current state which was keyed in by a user, and which is encapsulated within the action *initial*, is shown in table 5.11. Following execution, the truck-driver receives a request to satisfy a new goal, *at(parcel city5)* by the deadline 16 units, shown in table 5.12.

Table 5.12 The second goal

id:	*7*
condition:	*at(parcel city5)*
importance:	*20*
deadline:	*16*
duration:	*0*

In addition, the truck-driver is informed that the *parcel* is currently located at *city2* - this extra information is incorporated into the truck-driver's model of the current environment, *at(parcel city2),* as shown in the action *initial* shown in table 5.11. The "Select goal or action" takes the plan illustrated in table 5.10 which contains the new goal, *at(parcel city5)* (see table 5.12), and chooses to plan to achieve this new goal as opposed to executing the action *drive-truck(truck city4 city5).* Tables 5.13, 5.14 and 5.15 illustrate the actions which are selected (using step addition, simple establishment and conflict resolution) to achieve the goal *at(parcel city5)* by its deadline 16 (see table 5.12), assuming the current time is 6, while table 5.16 shows the resulting partial plan which contains

these actions. The "Achieve goal" component requires the initial plan shown in table

Table 5.13 Actions for Plan 3

id:	10	9
name:	*drive-truck(truck city4 city2)*	*load-truck(parcel truck city2)*
precondition:	*at (truck city4) & has-fuel(truck) & connects(city4 city2)*	*at(truck city2) & at(parcel city2)*
delete:	*at(truck city4)*	*at(parcel city2)*
add:	*at(truck city2)*	*in(parcel truck)*
goals:	*(1 7)*	*(7)*
pros:	*(pleasure 1.2)*	*nil*
cons:	*(conserve-fuel 0.3) & (conserve-tyres 0.2)*	*nil*
duration:	*2*	*1*
importance:	*26*	*20*
effort:	*35*	*14*

5.10, as well as the operator schemas/action templates which are shown in chapter 4, section 4.4.3, table 4.4. Again, the step addition and simple establishment procedures

Table 5.14 Actions for Plan 3

id:	11	4
name:	*drive-truck(truck city2 city4)*	*drive-truck(truck city4 city5)*
precondition:	*at(truck city2) & has-fuel(truck) & connects(city2 city4)*	*at(truck city4) & has-fuel(truck) & connects(city4 city5)*
delete:	*at(truck city2)*	*at(truck city4)*
add:	*at(truck city4)*	*at(truck city5)*
goals:	*(1 7)*	*(1 7)*
pros:	*(pleasure 1.2)*	*(pleasure 1.8)*
cons:	*(conserve-fuel 0.3) & (conserve-tyres 0.2)*	*(conserve-fuel 0.3) & (conserve-tyres 0.4)*
duration:	*2*	*3*
importance:	*26*	*26*
effort:	*35*	*35*

require the two look-up tables shown in chapter 4, tables 4.6 and 4.7 which store values

estimating how long it takes to execute various actions, and values indicating the degree to which each action may support or undermine the agent's motivations, respectively. In

Table 5.15 Actions for Plan 3

id:	8	2
name:	*unload-truck(parcel truck city5)*	*unload-truck(package truck city5)*
precondition:	*at(truck city5) & in(parcel truck)*	*at(truck city5) & in(package truck)*
delete:	*in(parcel truck)*	*in(package truck)*
add:	*at(parcel city5)*	*at(package city5)*
goals:	*(7)*	*(1)*
pros:	*nil*	*nil*
cons:	*nil*	*nil*
duration:	*1*	*1*
importance:	*20*	*6*
effort:	*14*	*21*

addition, the step addition and simple establishment procedures must assign values to the fields *importance* and *effort* associated with each action. In this example, two goals, the goal *at(package city5)* and the goal *at(parcel city5)*, have been achieved which have the values 6 and 20 respectively indicating their importance. The action *drive-truck(truck city4 city5)* now contributes to the new goal *at(parcel city5)* in addition to the original goal *at(package city5)* - the *goals*, *importance* and *effort* fields associated with this action are therefore updated to become the values *(1 7)*, *26* and *35* respectively (see table 5.14). All actions that contribute towards the preconditions of the action *drive-truck(truck city4 city5)*, namely *drive-truck(truck city4 city2)* and *drive-truck(truck city2 city4)*, also contribute towards the achievement of the two goals *at(package city5)* and *at(parcel city5)* which means their associated *goals*, *importance* and *effort* fields are assigned the same values as shown above, namely *(1 7)*, *26* and *35* respectively. The new actions *load-truck(parcel truck city2)* and *unload-truck(parcel truck city5)* (see tables 5.13 and 5.15) only contribute towards the achievement of the new goal *at(parcel city5)* which means their associated *goals*, *importance* and *effort* fields are assigned the values *(7)*, *20* and *14*

(this value means that 14 goal achievement cycles were performed to achieve each goal/

Table 5.16 Plan 3 which achieves the goal *at(parcel city5)*

actions:	id:	name:			
	10 *9* *11* *4* *8* *2*	*drive-truck(truck city4 city2)* *load-truck(parcel truck city2)* *drive-truck(truck city2 city4)* *drive-truck(truck city4 city5)* *unload-truck(parcel truck city5)* *unload-truck(package truck city5)*			
links	establisher:	condition:			consumer:
	11 *10* *9* *8* *2* *4* *0* *0* *0* *0* *4* *0* *0* *10* *0* *0* *0*	*at(truck city4)* *at(truck city2)* *in(parcel truck)* *at(parcel city5)* *at(package city5)* *at(truck city5)* *at(parcel city2)* *connects(city4 city2)* *has-fuel(truck)* *at(truck city4)* *at(truck city5)* *connects(city2 city4)* *has-fuel(truck)* *at(truck city2)* *in(package truck)* *connects(city4 city5)* *has-fuel(truck)*			*4* *9* *8* *7* *1* *2* *9* *10* *10* *10* *8* *11* *11* *11* *2* *4* *4*
open:	*nil*				
unsafe:	*nil*				
ordering:	*drive-truck(truck city4 city2) -> load-truck(parcel truck city2) ->* *drive-truck(truck city4 city2) -> drive-truck(truck city4 city5) ->* *unload-truck(parcel truck city5),* *unload-truck(package truck city5)* *(10 -> 9 -> 11 -> 4 -> 8 ->7) & (4 -> 2 -> 1)*				
bindings:	*<varset {(?from11 city2 ?loc9 ?to10)} = city2 not(?to11 ?truck11 ?truck9 ?ob9 ?truck10 ?from10)>* *<varset {(?truck4 ?truck2 truck ?truck9 ?truck8 ?truck10 ?truck11)} = truck not(?from4 ?to4 ?loc2 ?ob2 ?ob9 ?loc9 ?loc8 ?ob8 ?to10 ?from10 ?to11 ?from11)>* *<varset {(?from4 city4 ?from10 ?to11)} = city4 not(?to4 ?truck4 ?truck10 ?to10 ?truck11 ?from11)>* *<varset {(?ob8 parcel ?ob9)} = parcel not(?truck8 ?loc8 ?loc9 ?truck9)>* *<varset {(?to4 city5 ?loc2 ?loc8)} = city5 not(?from4 ?truck4 ?truck2 ?ob2 ?truck8 ?ob8)>* *<varset {(?ob2 package)} = package not(?truck2 ?loc2)>*				
goals:	id:	conditions:	deadline:	effort:	importance:
	1 *7*	*at(package city5)* *at(parcel city5)*	*20* *16*	*21* *14*	*6* *20*

subgoal contributing towards the new goal *at(parcel city5)*). The action *unload-*

truck(package truck city5) remains unchanged.

5.5 Summary

In this chapter we described in detail the "Achieve goal" component which is a part of the "Plan to achieve goal" process. The aims and motivations of the work presented in this thesis were to develop a planning/execution architecture to be used by a self-motivated autonomous agent in order to achieve its goals. A key component of this architecture is the "Achieve goal" or planning component, which, given a goal or subgoal, finds an action that will achieve that goal or subgoal. In this chapter we described how the goal achievement process used by SNLP was extended to deal with three features required by an autonomous agent: the ability to take into account the context of that agent; the ability to interleave planning and execution; as well as the ability to plan and act in real/simulated time.

Chapter 6

Planning to Achieve a Goal - Part 2

6.1 Introduction

In this chapter we describe the remaining components of the "Plan to achieve goal" process shown in chapter 2, section 2.2.6, figure 2.2: the "Estimate deadlines", "Edit the partial plan" and "Evaluate partial plans" components.

6.2 Estimating the Deadlines of Actions

6.2.1 Introduction

This section introduces an algorithm developed to estimate the deadlines for actions and their associated subgoals or preconditions. The "Plan to achieve goal" component described in this and the previous chapter (illustrated in chapter 2, section 2.2.6, figure 2.2) extends the classical nonlinear planning algorithm by introducing time - goals have deadlines and actions have duration. By introducing deadlines and duration, it is necessary for an agent to be able to reason about whether or not it can meet those deadlines. Part of meeting a deadline involves ensuring that planning and execution take place early enough - for a goal to be achieved by its deadline, all actions that contribute towards the achievement of that goal must be executed by their own (earlier) deadlines which are determined by their position within the overall sequence of actions as well as by their duration. We define the deadline of an action to be the latest possible time by which execution of that action should commence in order to ensure that the goal(s) towards which the action contributes is achieved by its/their associated deadline(s). By implication, the preconditions of an action must be true by the action's deadline. This enables deadlines to be assigned to subgoals as well as to actions within the partial plan. A mechanism is required which estimates the deadlines of actions and subgoals for the following reasons.

1. To enable the "Select goal or action" process (see chapter 4, section 4.5) to select which goal/subgoal or action is to be achieved/executed.
2. To enable the agent to reason about whether or not there is sufficient time available to achieve its goals prior to their deadlines. If not, the plan is edited to remove a goal and its associated actions and constraints (this is the responsibility of the "Edit the partial plan" component) to create sufficient time for the agent to achieve its remaining goals.

Using information about the deadlines of goals (these are goals which are created by the "Generate/update goals" procedure), the ordering (encapsulated using temporal constraints) and duration of actions (estimated by the "Achieve goal" process described in chapter 5, section 5.2), it is possible to estimate deadlines for actions and their associated subgoals (or preconditions). Each time a new temporal constraint is added to a partial plan (by the "Achieve goal" procedures step addition, simple establishment and conflict resolution), the ordering between actions is further constrained. This means that each time a new temporal constraint is added to a partial plan, more information is acquired concerning the ordering of actions within that partial plan which in turn means that the deadline associated with each action can be estimated more accurately. It is only when actions are totally ordered that deadlines can accurately be assigned to those actions.

In estimating the deadlines of actions a pessimistic approach is adopted - when actions are only partially ordered with respect to each other, those actions are assigned the earliest likely deadline. For example, three actions *a1*, *a2* and *a3* each with a duration of 3 minutes (i.e. it takes 3 minutes to execute each action) remain unordered with respect to each other. These actions were selected to achieve the goal *g* which has the deadline 5pm. We estimate the deadline for each action to be 4.51pm (5pm - (3mins + 3mins + 3mins)). This is because in the absence of further information, any of the three actions could be executed first. This means that the deadlines estimated for actions within a partial plan are likely to be too early in the early stages of plan refinement. However, this approach prevents the situation occurring whereby a goal cannot be achieved because the deadlines that were estimated for its contributory actions were too late. If there is insufficient time to achieve all of the goals, the deadline estimation process fails. The deadline estimation procedure is not a reliable indicator as to whether or not there is sufficient time available to achieve all of the goals in a plan. In some situations it estimates deadlines for all actions even though there is insufficient time available. However,

as a plan becomes more refined the deadline estimation process more reliably fails when there is insufficient time. When the deadline estimation procedure *does* fail however, it means that there really isn't enough time available to achieve all of the goals.

6.2.2 An algorithm that enables deadlines to be assigned to actions

In this section we describe the algorithm used to estimate deadlines for actions and sub-goals within a partial plan (see table 6.1). This algorithm assumes that the deadlines associated with goals created by the "Generate/update goals" component are hard or *fixed* and that if the "Plan to achieve goal" cannot achieve a goal by its deadline, it has failed to achieve that goal. In addition, the algorithm assumes that all actions are executed sequentially by some agent with the properties described in chapter 1, section 1.5 - it is assumed that such an agent cannot execute more than one action simultaneously.

The algorithm considers each goal created by the "Generate/update goals" component in turn as these have *fixed* deadlines. When estimating the deadlines for each of the actions within the partial plan, the algorithm begins by taking the goal with the earliest deadline, the goal *g*. The deadlines for all actions that *are* or that are *possibly* constrained to be executed prior to *g* are estimated. A part of this process involves determining whether it is possible to execute these actions within the time interval associated with *g* - the time interval associated with the goal *g* (which has the earliest deadline) begins at the current time *NOW* and ends at a time equal to the deadline associated with *g*. If it is not possible to achieve all actions that *are* constrained to be executed prior to *g* in this time interval, the value :fail is returned. If it is not possible to achieve one or more of the actions that are *possibly* constrained to be executed prior to *g* in this time interval, the deadlines associated with such actions are assigned within the next time interval - the time interval associated with the goal with the next earliest deadline, the goal *g+1*. The algorithm then takes the goal with the next earliest deadline, the goal *g+1*, and estimates the deadlines for all remaining actions (i.e. actions whose deadlines have not yet been estimated including any actions that were *possibly* constrained to be executed prior to the previous goal *g* whose deadlines could not be estimated) that *are* or that are *possibly* constrained to be executed before that goal. The time interval associated with the goal *g+1* begins at a time equal to the previous goal *g*'s latest point (this time is determined by adding the duration associated with *g* to the deadline associated with *g*) and ends at a

time equal to the deadline associated with the current goal *g+1*. The algorithm continues

Table 6.1 Estimating the deadlines associated with actions their preconditions

1. Let the set *unestacts* contain all actions whose deadlines have not yet been estimated. This set is initialised to contain all actions belonging within a partial plan *p*.
2. Let *begin* be the current time (*begin* marks the beginning of the current time interval which ends with the deadline of the first goal to be considered).
3. For each goal *g*, (beginning with the goal with the earliest deadline):
   (a) Let the set *defprior* contain all actions which both belong to the set *unestacts* and which are constrained to be executed prior to *g*.
   (b) Let the set *possprior* contain all actions which both belong to the set *unestacts* and which *may* be executed prior to *g* (these are actions that are currently unordered with respect to *g*).
   (c) If the sets *defprior* and *possprior* are empty, continue with the goal with the next earliest deadline belonging to the partial plan (i.e. the goal *g+1*) from step 3. above.
   (d) Else, calculate where possible the deadlines of all actions belonging to the sets *defprior* and *possprior* as follows.
      i. Subtract the sum of the durations of all actions belonging to the set *defprior* from the deadline associated with *g*.
      ii. If the result is prior to the time *begin*, there is insufficient time available to execute all of the actions constrained to be executed prior to *g*. Return two values, :fail and a set containing the goal *g*.
      iii. Else, let *deadline* be the value obtained by subtracting the sum of the durations of all actions within the sets *defprior* and *possprior* from the deadline associated with *g*.
         A. If *deadline* is later than or equal to the time *begin*, provisionally set the deadlines of all actions belonging to the sets *defprior* and *possprior* to be this value. Using the value *deadline*, calculate the deadlines of all actions belonging to the sets *defprior* and *possprior* (see Algorithm A outlined in table 6.2). Once the deadline of an action has been calculated remove the action from the set *unestacts*.
         B. Else, if *deadline* is earlier than the time *begin*, let the provisional deadline associated with each action belonging to the sets *defprior* and *possprior* be the time *begin*.
            I. Using the time *begin* and the set *defprior*, calculate the deadlines for all actions belonging to the set *defprior* (see Algorithm A outlined in table 6.2). Once the deadline of an action has been estimated, remove that action from the set *unestacts*.
            II. Using the time *begin* and the sets *defprior* and *possprior*, calculate (if possible) the deadlines for all actions belonging to the set *possprior* (see Algorithm B in table 6.3). Once the deadline of an action has been estimated, remove the action from the set *unestacts*. (There may not be enough time to execute all of the actions belonging to the set *possprior* prior to the deadline associated with *g*. If this is the case, the deadlines of those actions that cannot be executed prior to *g* will be estimated within the next time interval - the time interval associated with the goal *g+1*.)
   (e) Set *begin* to be the value obtained by adding the duration associated with *g* to the deadline associated with *g* (this marks the beginning of the next time interval).
   (f) Continue from step 3. above with the goal *g+1* - i.e. the goal with the next earliest deadline.

in this manner until the deadlines for all actions have been estimated with respect to each of the goals.

The algorithm terminates under the following conditions: there is insufficient time available to achieve some goal by its deadline - in this case the values :fail and a set containing only the goal which cannot be achieved is returned; the deadlines have been estimated for all actions belonging within the plan - in this case a plan containing a set of actions with newly estimated deadlines is returned. In the following two sections the algorithms used to estimate the deadlines of actions are described in detail - the first is used in stages 3. (d) iii. A. and 3. (d) iii. B. I. in table 6.1 while the second is used in stages 3. (d) iii. B. II. in table 6.1.

Algorithm A

This algorithm (see table 6.2) is used to estimate the deadlines of the set of actions *actions* belonging to a partial plan *p* that are definitely constrained or that are possibly constrained to be executed prior to some goal *g* (see stages 3. (d) iii. A. and 3. (d) iii. B. I. in table 6.1). This algorithm assumes that there is sufficient time available to achieve

Table 6.2 Algorithm A

1. Let *actions* be the set of actions belonging to a partial plan *p* whose deadlines are to be estimated. These actions are definitely constrained or are possibly constrained to be executed prior to some goal *g*.
2. Let *deadline* be a provisional deadline by which execution of the set *actions* must commence in order to achieve *g* by its deadline.
3. For each action *act* belonging to the set *actions*.
   (a) Provisionally set the deadline of *act* to be the value *deadline*.
   (b) Are any of the actions in the set *actions* constrained to be executed prior to *act* (this is determined by examining the temporal contraints belonging within the partial plan *p*)?
      i. If yes, add the sum of the durations associated with each action (in the set *actions*) constrained to be executed prior to *act* to the provisional deadline associated with *act* (i.e. the value *deadline*). The result is the estimated deadline for the action *act*.
      ii. Else, if no, the deadline of the action *act* is the value *deadline*.

the goal *g* by its deadline by executing the sequence of actions *actions*. Each action is provisionally assigned the value *deadline* which is the time by which the sequence of actions *actions* must commence execution in order to achieve the goal *g* by its deadline (i.e. it assumes initially that each action is possibly the first in the sequence of actions). In order to estimate the deadline associated with an action *act* belonging to *actions*, the sum of the durations of each action (belonging to the set *actions*) that is constrained to be executed prior to *act* (this is determined by examining the temporal constraints belonging

to the partial plan *p*), is added to the value *deadline* (the provisional deadline associated with *act*). The algorithm adopts a pessimistic approach in that the estimated deadlines of actions and their associated subgoals are likely to be too early if the partial plan is in an early stage of refinement (the ordering of actions will not be specified which means it is assumed that each action is possibly first in the sequence of actions). This algorithm also assumes that the durations of all of the actions within the partial plan are known.

Algorithm B

This algorithm (see table 6.3) is used to estimate deadlines for the set of actions *possprior* that are possibly constrained to be executed prior to some goal *g* (see stage 3. (d) iii. B. II. in table 6.1). The algorithm assumes firstly that there is sufficient time available to achieve all actions that are definitely constrained to be executed prior to *g* (the set *defprior*) and secondly that there isn't enough time available to execute all of the actions belonging to the set *possprior* prior to the deadline associated with *g*. In addition, this algorithm also assumes that the durations of all of the actions within the partial plan are known and that actions must be executed sequentially.

Table 6.3 Algorithm B

1. Let *defprior* be the set of actions belonging to a partial plan *p* that are definitely constrained to be executed prior to some goal *g*.
2. Let *possprior* be the set of actions belonging to a partial plan *p* whose deadlines are possibly constrained to be executed prior to some goal *g*. These are the actions whose deadlines are to be estimated.
3. Let *deadline* be a provisional deadline by which execution of the sets *defprior* and *possprior* must commence in order to possibly achieve *g* by its deadline.
4. For each action *act* belonging to the set *possprior*.
   (a) Provisionally set the deadline of *act* to be the value *deadline*.
   (b) Add the sum of the durations associated with *act* and with each action belonging either to the set *defprior* or to the set *possprior* which is constrained to be executed prior to *act* (this is determined by examining the temporal contraints belonging within the partial plan *p*), to the provisional deadline *deadline*. The result becomes the new provisional deadline associated with the action *act*.
      i. If the result is later than the deadline associated with *g*, there is insufficient time available to execute *act* before the deadline associated with *g*. It is impossible to assign a deadline to *act*. Continue with the next action (i.e. from stage 4. above).
      ii. Else, if the result is before or equal to the deadline associated with *g*, the estimated deadline associated with the action *act* is the provisional deadline obtained in stage 4. (b) above.

Each action (in the set *possprior*) is provisionally assigned the value *deadline* which is the beginning of the time interval during which the set of actions *defprior* is to be exe-

cuted to achieve *g* and during which some of the actions belonging to the set *possprior* are to be executed, if possible. In order to estimate the deadline associated with an action *act* belonging to the set *possprior*, the sum of both the duration of *act* and the durations of each action (belonging to either the set *defprior* or to the set *possprior*) that is constrained to be executed prior to *act* (this is determined by examining the temporal constraints belonging to the partial plan *p*), is added to the value *deadline* (the provisional deadline associated with *act*). If the result is later than the deadline associated with *g*, it is impossible to assign a deadline to the action *act* as it is impossible to execute *act* prior to the deadline associated with *g*. If the result is earlier or equal to the deadline associated with *g*, then this result becomes the estimated deadline associated with *act*. The algorithm adopts a pessimistic approach in that the estimated deadlines of actions and their associated subgoals are likely to be too early if the partial plan is in an early stage of refinement (the ordering of actions will not be specified which means it is assumed that each action is possibly first in the sequence of actions).

6.2.3 Problems with the algorithm

There is a problem concerning the amount of reasoning required to determine whether or not there is sufficient time available to execute all of the actions within the partial plan, taking into account the deadlines associated with goals. The time available to the agent falls into various free time intervals: the first time interval begins with the current time and ends with the time marking the deadline associated with the earliest goal; the second time interval begins with the deadline associated with the earliest goal and ends with the deadline associated with the next earliest goal, etc. Because actions take up "blocks" of time to execute, it is not easy to fit actions into time intervals in an optimal manner. For example, if we have two actions, where one takes 30 minutes to execute whilst the other takes 10 minutes to execute, and we have a time interval of 35 minutes we cannot execute both actions within this time interval. If we choose to execute the first action within this time interval we potentially "waste" 5 minutes of the time interval, while if we choose to execute the second action within this time interval we could "waste" 25 minutes. Indeed, to estimate whether or not we can fit all actions into existing time intervals, we might need to consider every possible alternative ordering of actions. For a large number of actions and time intervals this will be extremely expensive. The algorithm we adopt to estimate deadlines is more likely to indicate failure as the number of temporal

constraints increases (this reduces the number of alternative orderings). The prime objective of the "Estimate deadlines" component is to estimate the earliest possible deadline associated with each action and its preconditions. The "Estimate deadlines" component will indicate failure when there is insufficient time to execute all actions within the constraints imposed by goals and their deadlines. However, it is not guaranteed to indicate failure in all circumstances, especially those in which actions remain unordered with respect to each other.

6.2.4 An example

Figure 5.3 in chapter 5, section 5.2.8, illustrates a partial plan containing three goals, *g1*, *g2* and *g3*, with seven actions, *a1*, *a2*, *a3*, *a4*, *a5*, *a6* and *a7*. Arrows indicate ordering

Table 6.4 Ordering constraints

goals/actions	definitely prior actions	possibly prior actions
g1	*a1 a2 a4 a5 a6 a7*	*a3*
g2	*a2 a5 a6 a7*	*a1 a3 a4*
g3	*a3 a6 a7*	*a1 a2 a4 a5*
a1	*a2 a4 a5 a6 a7*	*a3*
a2	*a5 a6 a7*	*a3 a4*
a3	*a6 a7*	*a1 a2 a4 a5*
a4		*a2 a3 a5 a6 a7*
a5		*a3 a4 a6 a7*
a6	*a7*	*a4 a5*
a7		*a4 a5*

constraints between actions - for example the action *a7* is prior to *a6* which is prior to *a3*, etc.,. In order to illustrate the deadline estimation procedure described above, we will describe how deadlines are assigned to each of the actions in this plan. Table 6.4 shows how the actions and goals are ordered with respect to each other - for example, chapter 5, section 5.2.8, figure 5.3 shows that the actions *a5*, *a6* and *a7* are ordered to occur prior to *a2*, while the actions *a3* and *a4* are possibly prior to *a2* (i.e. there are no constraints specifying the ordering of *a2* and *a3*, or of *a2* and *a4*). Table 6.5 illustrates the deadlines

associated with the goals *g1*, *g2* and *g3* while table 6.6 illustrates the duration of (i.e. the

Table 6.5 Goals and their deadlines

goals	deadlines
g1	25
g2	18
g3	11

estimated amount of time it will take to execute) each action. Below we illustrate, using

Table 6.6 Actions and their durations

actions	duration
a1	5
a2	3
a3	4
a4	2
a5	3
a6	4
a7	2

the algorithms described above in table 6.1, how deadlines are assigned to each action.

1. The set of actions *unestacts* whose deadlines are not yet estimated includes all of the actions *a1*-*a7* (see table 6.1, stage 1.).
2. The time *begin* marking the beginning of the current time interval is 0.
3. Take the goal with the earliest deadline, *g3*. The task is to estimate the deadlines for all actions that are both definitely prior to (the set *defprior*, containing actions *a3*, *a6* and *a7* - see table 6.1, stage 3. (a)) and (if possible) possibly prior to (the set *possprior*, containing actions *a1*, *a2*, *a4* and *a5* - see table 6.1, stage 3. (b)) *g3*.
   (a) Estimate the earliest deadline required to execute each of the actions that are definitely constrained to occur prior to *g3* (*a3*, *a6* and *a7*) - see table 6.1, stage 3. (d) i. This deadline is 1 (obtained by subtracting the sum of the durations of actions *a3*, *a6* and *a7* from the deadline associated with *g3*). The result 1 is later than *begin*, the current time 0.

$$11 - (4 + 4 + 2) = 1$$

   (b) Estimate the earliest deadline required to execute the actions that are both defi-

nitely constrained to occur prior to *g3* (actions *a3*, *a6* and *a7*) and that may possibly occur prior to *g3* (*a1*, *a2*, *a4* and *a5*). This deadline is -12 (obtained by subtracting the sum of the durations associated with each action from the deadline associated with *g3*) - see table 6.1, stage 3. (d) iii. The result, -12, is earlier than the current time 0.

$$11 - (4 + 4 + 2 + 5 + 3 + 2 + 3) = -12$$

i. Let the provisional deadline associated with each action belonging to the sets *defprior* and *possprior* be the time *begin* (i.e. 0) - see table 6.1, stage 3. (d) iii. B.

ii. Estimate the deadlines for the actions belonging to the set *defprior* (i.e. that are definitely constrained to occur prior to *g3*) - see table 6.1, stage 3. (d) iii. B. I. and Algorithm A in table 6.2.

   A. Estimate the deadline for *a3*.

      I. The actions belonging to the set *unestacts* which are definitely constrained to occur prior to *a3* (see table 6.4) are *a6* and *a7*. The deadline associated with *a3* is obtained by adding the durations associated with *a6* and *a7* to the provisional deadline 0 and is the value 6.

$$0 + 4 + 2 = 6$$

   B. Estimate the deadline for *a6*.

      I. The action belonging to the set *unestacts* which is definitely constrained to occur prior to *a6* (see table 6.4) is *a7*. The deadline associated with *a6* is obtained by adding the duration associated with *a7* to the provisional deadline 0 which gives the value 2.

$$0 + 2 = 2$$

   C. Estimate the deadline for *a7*.

      I. There are no actions belonging to the set *unestacts* that are definitely constrained to occur prior to *a7* (see table 6.4). The deadline associated with *a7* is therefore 0.

iii. Estimate the deadlines for the actions belonging to the set *possprior* (i.e. that may possibly occur prior to *g3*) - see table 6.1 stage 3. (d) iii. B. II. and Algorithm B in table 6.3.

A. Estimate the deadline for *a1*.

I. The actions belonging to the set *unestacts* which are definitely constrained to occur prior to *a1* (see table 6.4) are *a2*, *a4*, *a5*, *a6* and *a7*. The deadline associated with *a1* is obtained by adding the durations associated with *a2*, *a4*, *a5*, *a6*, *a7*, as well as *a1*, to the provisional deadline 0 and is the value 19. This is later than the deadline associated with *g3* so it is not possible to assign a deadline to *a1* at this point.

$$0 + 3 + 2 + 3 + 4 + 2 + 5 = 19$$

B. Estimate the deadline for *a2*.

I. The actions belonging to the set *unestacts* which are definitely constrained to occur prior to *a2* (see table 6.4) are *a5*, *a6* and *a7*. The deadline associated with *a2* is obtained by adding the durations associated with *a5*, *a6*, *a7*, as well as *a2*, to the provisional deadline 0 which gives the value 12. This is later than the deadline associated with *g3* so it is not possible to assign a deadline to *a2* at this point.

$$0 + 3 + 4 + 2 + 3 = 12$$

C. Estimate the deadline for *a4*.

I. There are no actions belonging to the set *unestact* that are definitely constrained to occur prior to *a4* (see table 6.4). The deadline assigned to *a4* is therefore 0.

D. Estimate the deadline for *a5*.

I. There are no actions belonging to the set *unestact* that are definitely constrained to occur prior to *a5* (see table 6.4). The deadline assigned to *a5* is therefore 0.

iv. Remove the actions whose deadlines have been estimated (i.e. the actions *a3*, *a6*, *a7*, *a4* and *a5*) from the set *unestacts*. *unestacts* now includes only the actions *a1* and *a2* - see table 6.1, stage 3. (d) iii. B. II.

v. Set the time *begin* (which now marks the beginning of the next current time interval) to 11 (this is obtained by adding the duration associated with *g3*, 0, to the deadline associated with *g3*,11) - see table 6.1, stage 3. (e).

4. Take the goal with the next earliest deadline, *g2*. The task is to estimate the dead-

lines for all actions remaining within the set *unestacts* that are definitely prior to (the set *defprior*, containing the action *a2*) and (if possible) possibly prior to (the set *possprior*, containing the action *a1*) *g2* - see table 6.1, stage 3.

(a) Estimate the earliest deadline required to execute all actions which are definitely constrained to occur prior to *g2* (in this case the action *a2*). This deadline is 15 (obtained by subtracting the sum of the durations of actions that are definitely constrained to occur prior to *g2* - i.e. *a2* - from the deadline associated with *g2*). The result 15 is later than the time marking the beginning of the time interval, 11 (see table 6.1, stage 3. (d) i.).

$$18 + 3 = 15$$

(b) Estimate the earliest deadline required to execute the actions that are both definitely constrained to occur prior to *g2* (the action *a2*) and possibly constrained to occur prior to *g2* (the action *a1*). This deadline is 10 (obtained by subtracting the sum of the durations associated with each action from the deadline associated with *g2*). The result 10 is earlier than the time marking the beginning of the time interval, 11 (see table 6.1, stage 3. (d) iii.).

$$18 - (3 + 5) = 10$$

i. Let the provisional deadline associated with each action belonging to the sets *defprior* and *possprior* be the time *begin* (i.e. 11) - see table 6.1, stage 3. (d) iii. B.

ii. Estimate the deadlines for the actions belonging to the set *unestacts* (i.e. the actions *a1* and *a2*) that are definitely constrained to occur prior to *g2* - in this case the action *a2* (see table 6.1, stage 3. (d) iii. B. I. and Algorithm A in table 6.2).

A. Estimate the deadline for *a2*.

I. There are no actions belonging to the set *unestacts* which are definitely constrained to occur prior to *a2* (see table 6.4). The deadline assigned to *a2* is therefore 11.

iii. Estimate the deadlines for the actions belonging to the set *unestacts* that may possibly occur prior to *g2* (i.e. the action *a1*) - see table 6.1, stage 3. (d) iii. B. II. and Algorithm B in table 6.3.

A. Estimate the deadline for *a1*.

I. The action belonging to the set *unestacts* which is definitely constrained to occur prior to *a1* (see table 6.4) is *a2*. The deadline assigned to *a1* is obtained by adding the duration associated with *a2*, and with *a1*, to the provisional deadline 11 and is the value 19. This is later than the deadline associated with *g2* so it is not possible to assign a deadline to *a1* at this point.

$$11 + 3 + 5 = 19$$

(c) Remove the actions whose deadlines have been estimated (i.e. the action *a2*) from the set *unestacts*. *unestacts* now contains only the action *a1* - see table 6.1, stage 3. (d) iii. B. II.

(d) Set the time *begin* (which now marks the beginning of the next current time interval) to 18 (this is obtained by adding the duration associated with *g2*, 0, to the deadline associated with *g2*,18) - see table 6.1, stage 3. (e).

5. Take the goal with the next earliest deadline, *g1*. The task is to estimate the deadlines for all actions belonging to the set *unestacts* that are definitely prior to (the action *a1*) and (if possible) possibly prior to (in this case there are actions that are possibly prior) to *g1* - see table 6.1, stage 3.

(a) Estimate the earliest deadline required to execute all actions that are definitely constrained to occur prior to *g1* (in this case the action *a1*). This deadline is 20 (obtained by subtracting the sum of the durations of actions that are definitely constrained to occur prior to *g1* - i.e. the action *a1* - from the deadline associated with *g1*). The result 20 is later than the time marking the beginning of the time interval, 18 (see table 6.1, stage 3. (d) i.).

$$25 - 5 = 20$$

(b) Estimate the earliest deadline required to execute the actions that are both definitely constrained to occur prior to *g1* (the action *a1*) and that may possibly occur prior to *g1* (this is an empty set). This deadline is 20 (obtained by subtracting the sum of the durations associated with each action from the deadline associated with *g1*). The result (the value *deadline*), 20 is later than the time marking the beginning of the time interval, 18 (see table 6.1, stage 3. (d) iii.).

$$25 - 5 = 20$$

i. Using the value *deadline* (the provisional deadline, 20), estimate the dead-

lines for the actions belonging to the set *unestacts* that are definitely constrained (i.e. the action *a1*) to occur prior to *g1* - see table 6.1, stage 3. (d) iii. A. and Algorithm A in table 6.2.

A. Estimate the deadline for *a1*.

I. There are no actions belonging to the set *unestacts* which are definitely constrained to occur prior to *a1* (see table 6.4). The deadline assigned to *a1* is therefore 20.

(c) Remove the actions whose deadlines have been estimated (i.e. the action *a1*) from the set *unestacts*. *unestacts* is now empty which means that deadlines have been assigned to each action in the partial plan.

Table 6.7 illustrates the deadlines assigned to the actions *a1-a7* by the deadline estimation process.

Table 6.7 Deadlines assigned to actions

actions	estimated deadline
a1	20
a2	11
a3	6
a4	0
a5	0
a6	2
a7	0

6.2.5 A truck-world domain example

In chapter 5, section 5.4.1, table 5.9 illustrates a partial plan which was generated to achieve the goal *at(package city5)* (see chapter 4, section 4.4.2, table 4.3) by its deadline 20. The "Estimate deadlines" component is responsible for assigning deadlines to each of the actions within this plan, which are shown in chapter 5, section 5.4.1, tables 5.6, 5.7 and 5.8. In order to do this, the "Estimate deadlines components requires the following information.

1. The goals belonging within the partial plan (in particular, their associated deadlines and duration), see the partial plan field *goals* in chapter 5, section 5.4.1, table 5.9.

2. The set of ordering constraints, see the partial plan field *ordering* in chapter 5, section 5.4.1, table 5.9.
3. The actions belonging within the partial plan (in particular, their associated duration), see the partial plan field *actions* in chapter 5, section 5.4.1, table 5.9.
4. The current time.

The deadlines assigned to each action belonging to the plan shown in chapter 5, section 5.4.1, table 5.9, by the "Estimate deadlines" component where the current time is 0, is illustrated in table 6.8. This example is simple as there is only one goal, *at(package city5)*, prior to which each action is constrained.

Table 6.8 Deadlines are assigned to actions

id:	type:	name:	duration:	deadline:
3	*:action*	*load-truck(package truck city1)*	*1*	*10*
6	*:action*	*drive-truck(truck city1 city3)*	*2*	*11*
5	*:action*	*drive-truck(truck city3 city4)*	*3*	*13*
4	*:action*	*drive-truck(truck city4 city5)*	*3*	*16*
2	*:action*	*unload-truck(package truck city5)*	*1*	*19*
1	*:goal*	*at(package city5)*	*0*	*20*

6.2.6 Another truck-world domain example

Table 6.9 shows the deadlines that have been assigned by the "Estimate deadlines" component to the partial plan shown in chapter 5, section 5.4.2, table 5.16 with the current time 6. The actions belonging to this partial plan can be seen in chapter 5, section 5.4.2, tables 5.13, 5.14 and 5.15. By examining the ordering constraints *ordering* belonging to

Table 6.9 Deadlines are assigned to actions

id:	type:	name:	duration:	deadline:
10	*:action*	*drive-truck(truck city4 city2)*	*2*	*6*
9	*:action*	*load-truck(parcel truck city2)*	*1*	*8*
11	*:action*	*drive-truck(truck city2 city4)*	*2*	*9*
4	*:action*	*drive-truck(truck city4 city5)*	*3*	*11*
8	*:action*	*unload-truck(parcel truck city5)*	*1*	*14*
2	*:action*	*unload-truck(package truck city5)*	*1*	*14*
1	*:goal*	*at(package city5)*	*0*	*20*
7	*:goal*	*at(parcel city5)*	*0*	*16*

the partial plan shown in chapter 5, section 5.4.2, table 5.16, it can be seen that the actions *unload-truck(parcel truck city5)* and *unload-truck(package truck city5)* remain unordered with respect to each other, and that *unload-truck(parcel truck city5)* remains unordered with respect to the goal *at(package truck city5)* and *unload-truck(package truck city5)* remains unordered with respect to the goal *at(parcel truck city5)*. Because of this, each action is assigned a deadline that is too early - in particular this deadline assignment is based on the assumption that the action *unload-truck(package truck city5)* might later be constrained to occur both prior to the action *unload-truck(parcel truck city5)* and prior to the goal with the earlier deadine *at(parcel city5)*.

6.2.7 Implementing DEVISER window compression routines

An alternative way of implementing a procedure to estimate the deadlines of actions is to use the window compression routines developed as part of DEVISER [Vere 83]. Each time a new temporal constraint is added to a partial plan, DEVISER calls various window compression routines which compress the execution windows (an execution window consists of an earliest start time, a latest start time, an earliest finishing time and a latest finishing time, together with a duration) associated with actions. If, during the window compression propagation routines, an execution window is compressed such that the earliest execution time is later than the latest execution time (or the latest execution time is earlier than the earliest execution time), DEVISER fails - a plan cannot be found. Once compressed however, a window can never be decompressed. In the planning/execution architecture described in this thesis, a part of the planning process involves removing actions and goals if there is insufficient time available to achieve them (this is the responsibility of the "Edit the partial plans" component, see section 6.3). Once a goal and its associated constraints have been removed, the deadline estimation process must reestimate the deadlines associated with each action. This means we need to backtrack - i.e. previously compressed windows need to be decompressed. To accommodate this requirement, instead of compressing windows each time a temporal constraint is added during plan refinement, we first refine the plan, and run the window compression routines from scratch, taking each temporal constraint in turn. This is more costly, as it involves much duplication. In addition, the latest start times associated with actions (when assigned by the DEVISER window compression routines) are often too late, if three actions remain unordered with respect to each other and each have the duration 3

minutes, the latest start time assigned to each action will be 3 minutes earlier than the deadline associated with the goal to which they contribute - this may be too late. DEVISER assumes that actions may be executed in parallel. One possible solution might be to estimate the deadline by taking the mid point between the earliest start time and the latest start time.

6.2.8 Other related work

[Smith et al 00] identified a number of planners that extend the partial order, causal link (POCL) planning algorithm to reason about time. This approach, described by [Smith et al 00] as the Constraint-Based Interval (CBI) approach, has been used in many planning systems such as DEVISER [Vere 83], TRAINS-95 [Ferguson et al 96], Zeno [Penberthy & Weld 94], IxTeT [Ghallab & Laruelle 94], HSTS, [Muscettola 94] and *parc*PLAN ([Lever & Richards 94], [El-Kholy & Richards 96]). These planning systems have combined ideas from POCL planning with an interval representation for actions and propositions (first introduced by [Allen 84]) using constraint-satisfaction techniques to manage the relationship between intervals.

Such planners, because of their ability to reason about time, could have been used as the basis of the "Plan to achieve goals" component described in this and the previous chapter. However, as with DEVISER, one limitation with the CBI approach is that once temporal constraints have been posted, it is not easy to backtrack or unpost those constraints. If new constraints are incompatible with current constraints, such planners may fail. In the planning/execution architecture described in the thesis, should there be insufficient time available to achieve some goal, we require that goal together with its associated actions and constraints to be removed from the partial plan. A CBI planning system would therefore have to be modified in order to meet this requirement.

6.3 Editing a Partial Plan

6.3.1 Introduction

When a human generates and executes a plan to achieve their aims or goals, they might find that there will be insufficient time available to achieve all of those goals. Instead of abandoning their plan, they may choose instead to abandon achieving one of their goals or aims, thereby freeing time which can be used to fulfil their other aims. One of the

objectives of this research is to model this behaviour by giving the planning/execution architecture the facility to edit partial plans. Plan editing is therefore an essential part of the "Plan to achieve goal" process (see chapter 2, section 2.2.1, figure 2.1). Once either a

Table 6.10 The partial plan editing algorithm

1. Let *redundantacts* be the set of actions that contribute only towards the achievement of the goal *g*. (This set is determined by examining the set *goals* associated with each action - chapter 4, section 4.2.3 describes the representation used for actions. If the goal *g* is the only member of the set *goals*, then the action contributes only towards the achievement of *g*.)
2. Let *contributoracts* be the set of actions that contribute towards the achievement of the goal *g* in addition to other goals. An action contributes towards the achievement of the goal *g* if *g* is a member (but not the only member) of the action's set *goals*.
3. The set of actions belonging to the plan is updated as follows.
   (a) All actions which contribute only towards the achievement of the goal *g* (i.e. actions which belong to the set *redundantacts*) must be removed from the partial plan.
   (b) All actions belonging to the set *contributoracts* must be updated to reflect the fact that such actions no longer contribute to the goal *g*. This involves removing the goal *g* from the action's set *goals*, and updating the values *importance* and *effort* (both decrease in value) associated with the action.
4. The set of temporal constraints is updated. All temporal constraints associated with both the goal *g* and with actions belonging to the set *redundantacts* must be removed from the partial plan.
5. The set of open conditions/unachieved goals is updated. All unachieved goals/subgoals that are derived either from the goal *g* or from actions which belong to the set *redundantacts* must be removed from the partial plan.
6. The set of achieved goals/subgoals (persistence constraints) is updated. All persistence constraints containing (as an establisher or consumer) either the goal *g* or actions belonging to the set *redundantacts* must be removed from the partial plan.
7. The set of binding constraints is updated. All codesignation and noncodesignation constraints containing variables associated with actions belonging to the set *redundantacts* must be removed from the partial plan.
8. The set of goals is updated by removing the goal *g*.
9. Once actions contributing to the goal *g* are removed from a partial plan extra free time becomes available. This means that the deadlines of the actions remaining within the partial plan must be re-estimated.

goal or a subgoal has been achieved, the "Estimate deadlines" component (described in the previous section) attempts to assign deadlines to each of the actions within the resulting partial plans. If there is insufficient time available to achieve all of the goals within a plan, the "Edit the partial plan" procedure is called to remove each goal that cannot be achieved together with its associated actions and constraints. In addition, when an action is executed, the actual execution time may be longer than the estimated execution time (this is described in detail in chapter 7, section 7.3). In this situation, the "Estimate deadlines" process is called to reassign deadlines to the remaining actions in the plan. It may

be that as a consequence of the extra time required for execution, there is insufficient time available to achieve all of the goals belonging within the partial plan. In this case, the "Edit the partial plan" procedure edits the partial plan. To summarise, a partial plan must be edited under the circumstances outlined below.

1. There may be insufficient time available to achieve one or more goals by their associated deadlines. This scenario occurs when the "Estimate deadlines" component (described in section 6.2) fails to estimate deadlines for each of the actions within a partial plan. In this case, by removing the actions (together with their associated unachieved subgoals, codesignation, temporal and persistence constraints) which contribute only towards the achievement of the goals that cannot be achieved within the time available, it may be possible to create sufficient time to achieve the remaining goals by their associated deadlines.
2. If an action takes longer to execute than anticipated, there may no longer be sufficient time available to achieve the goals to which that action contributes. In this case, the "Edit the partial plan" procedure is called to remove such goals together with their associated actions and constraints.

The algorithm used to implement the "Edit the partial plan" component is shown in

Table 6.11 A partial plan prior to editing

actions	*id*	*importance*	*effort*	*goals*
	a1	6	3	*g1*
	a2	16	7	*g1 g2*
	a3	7	3	*g3*
	a4	6	3	*g1*
	a5	16	7	*g1 g2*
	a6	23	10	*g1 g2 g3*
	a7	23	10	*g1 g2 g3*
temporal constraints	*(a1 g1) (a4 a1) (a2 a1) (a2 g2) (a5 a2) (a6 a2) (a7 a6) (a6 a3) (a3 g3)*			
open conditions	*p41 p42 p51 p62 p71 p72*			
persistence constraints	*(a1 g1 g1) (a2 g2 g2) (a3 g3 g3) (a4 p11 a1) (a2 p12 a1) (a7 p61 a6) (a5 p21 a2) (a6 p22 a2) (a6 p31 a3)*			
goals	*g1 g2 g3*			

table 6.10.

6.3.2 An example

Chapter 5, section 5.2.8, figure 5.3 illustrates a partial plan containing three goals, *g1*, *g2* and *g3*. In this section we demonstrate how this partial plan is edited to remove the goal *g1* (we assume that the "Estimate deadlines" process has indicated that there is insufficient time available to achieve *g1* by its associated deadline - this would be the case if *g1* had the associated deadline 20 in the deadline estimation example of section 6.2.4 above). Table 6.11 illustrates the components of the partial plan prior to editing. As we can see from chapter 5, section 5.2.8, figure 5.3 the actions which only contribute towards the achievement of *g1* (the set *redundantacts*) are *a1* and *a4*. Actions which contribute to the goal *g1* in addition to other goals (the set *contributoracts*) include *a2*, *a5*, *a6* and *a7*. Table 6.12 illustrates the components of the partial plan once editing has taken place. The goal *g1* and the actions *a1* and *a4* have been removed from the plan, as have their associated temporal constraints, open conditions and persistence constraints (and binding constraints which we do not include in tables 6.11 and 6.12). The fields

Table 6.12 The partial plan following editing

actions	*id*	*importance*	*effort*	*goals*
	a2	10	4	*g2*
	a3	7	3	*g3*
	a5	10	4	*g2*
	a6	17	7	*g2 g3*
	a7	17	7	*g2 g3*
temporal constraints	*(a2 g2) (a5 a2) (a6 a2) (a7 a6) (a6 a3) (a3 g3)*			
open conditions	*p51 p62 p71 p72*			
persistence constraints	*(a2 g2 g2) (a3 g3 g3) (a7 p61 a6) (a5 p21 a2) (a6 p22 a2) (a6 p31 a3)*			
goals	*g2 g3*			

importance, *effort* and *goals* associated with actions *a2*, *a5*, *a6* and *a7* have been updated to reflect the fact that these actions no longer contribute towards *g1*.

6.3.3 A truck world domain example

When the "Select goal or action" component chooses to execute the action *load-truck(parcel truck city2)* shown in chapter 7, section 7.3.1, table 7.8, the "Execute

action" component updates the partial plan to reflect the outcome of execution, resulting in the partial plan shown in chapter 7, section 7.3.2, table 7.11. However, when executed, the action, *load-truck(parcel truck city2)* takes longer to execute than anticipated which means that the actual time following execution is later than the predicted time. The "Estimate deadlines" procedure is therefore called to reassign deadlines to each of the actions belonging within the updated partial plan of chapter 7, section 7.3.2, table 7.11. Because there is insufficient time available to achieve the goal *at(parcel city5)* by its deadline 16, the "Edit the partial plan" component edits the updated partial plan of chapter 7, section 7.3.2, table 7.11, to remove the goal *at(parcel city5)* together with its associated actions and constraints. The edited partial plan is shown in chapter 7, section 7.3.2, table 7.12.

6.3.4 Discussion

The "Edit the partial plan" component is designed (in conjunction with the "Estimate deadlines" component) to emulate the way humans abandon the achievement of some of their goals if they discover they do not have enough time to meet all of their objectives. Editing partial plans as part of the "Plan to achieve goal" procedure leads to two problems.

1. Various temporal constraint propagation techniques that undertake temporal reasoning are not easy to use when partial plans are edited as it is difficult to backtrack if goals are removed.
2. Editing partial plans may lead to duplicate partial plans in the search space. Various checks must be made to ensure this does not happen. Once one or more new partial plans have been generated by the "Plan to achieve goal" component, each new plan should be checked to ensure they are not duplicated. Alternatively, the search space could be implemented as a directed graph structure.

6.4 Evaluating Partial Plans

6.4.1 Introduction

Each time a goal or subgoal is achieved (by the "Achieve goal" procedure), several new partial plans are generated - one or more new partial plan for each new or for each existing action capable of achieving that goal or subgoal. The "Estimate deadlines" component then assigns deadlines to the actions belonging to each newly generated partial plan.

If it is impossible to assign deadlines to the actions within a plan (because there is insufficient time available to achieve the plan's goals), the "Edit the partial plan" component edits the partial plan and another attempt is made by the "Estimate deadlines" component to reassign deadlines. This cycle continues until the plan has been edited sufficiently in order that deadlines can successfully be assigned to each action belonging to that plan. To summarise, each time a goal or subgoal is achieved, several new partial plans are generated. In order to select the best partial plan for subsequent refinement, each newly generated partial plan must be evaluated prior to being added to an ordered search space of partial plans. Because the search space is ordered, the best partial plan for subsequent refinement is the head plan within the search space of partial plans. In this section we describe in detail the "Evaluate partial plans" process which uses the agent's motivations to evaluate each newly generated partial plan so that it is possible to select the "best" partial plan from among those within the search space. Each partial plan is assessed in terms of the extent to which each of its actions supports or undermines the agent's current motivations.

6.4.2 Evaluating partial plans

In chapter 2, section 2.2.3 and chapter 3, section 3.2.1 we described how the motivations of an agent are directly affected both by physical changes occurring within the agent's environment, either brought about by the agent or by other agents/physical processes, and by changes that are made to the agent's plan (as a consequence of planning to achieve some goal or subgoal). The planning/execution architecture illustrated in chapter 2, section 2.2.1, figure 2.1 updates an agent's motivations to reflect such changes each time the agent executes an action. The agent's motivations therefore partly represent the context of the agent within its environment. When an agent executes an action, one of the consequences is that the agent's motivations are updated to reflect the fact that the agent has brought about changes to its environment. For example, the motivation *hunger* will decrease in strength as a consequence of an agent eating some food. This means there is a difference between the agent's current motivations and how those motivations will be once the agent has executed some action. It can therefore be argued that one way of determining the degree to which the actions within a partial plan support the agent's motivations is to predict the future motivations of the agent (i.e. the future motivations that arise once those actions have been executed). In this thesis we assume that it is pos-

sible to crudely predict the effect that executing an action will have upon the agent's motivations. This is the responsibility of the "Evaluate partial plans" component which attempts to predict the effect that executing the sequence of actions belonging within each partial plan will have upon the agent's motivations. This component is able to predict the agent's future motivations by using the current motivations as well as the fields *pros* and *cons* associated with each action (see chapter 4, section 4.2.3).

For example, one way of achieving the goal of having some food, is to buy food at the local shop. When selecting an action that enables the agent to get to the shop, one of the options might be to walk down a secluded alleyway. When considering this option, the agent may predict that executing this action will undermine the motivation *safety*, i.e. the agent's safety might be compromised by executing this action. In fact, when a human agent plans to achieve such a goal, when imagining the action of walking down the secluded alleyway, they might actually experience a small increase in their level of fear, even though they are currently within a safe environment. An alternative means of travelling to the shop, such as driving by car, may not undermine the motivation *safety* but may have some other problem associated with it - the agent may not wish to pollute the environment by driving unnecessarily, or it may be hard/expensive to find a parking place. Likewise, when thinking about the option of travelling by coach between London and Edinburgh, a human agent might imagine the boredom of such a lengthy journey spent in such a confined space. A trade-off may have to be made, on one hand travelling to Edinburgh by coach is extremely dull and not very pleasant, on the other hand it supports the human agent's desire to save money. It is this notion of trade-off that we wish to capture by evaluating actions within a plan to determine the extent to which they support the agent's motivations. It is not just different actions that may affect the agent's motivations differently. The way those actions are instantiated might also have different effects on the agent's motivations. For example, if an agent is deciding to go out to eat, the different places at which the agent might eat will support its motivations to a differing extent. Eating at a very expensive restaurant may undermine the agent's motivation to save money, but support the agent's motivation for eating good food. Eating at a fast-food restaurant however, may support the agent's motivation to save money but undermine the agent's motivation for eating good food.

6.4.3 Examples of the degree to which actions support motivations

An agent has to achieve the goal of having some food. The different possibilities available to the agent are listed below along with the degree to which they may support the agent's motivations.

1. Go to a shop to buy some food, prepare the food and cook a meal. This may support the motivations associated with saving money, eating healthy food and eating enjoyable food. In addition, this may undermine the motivations associated with idleness and enjoyment (i.e. the agent may not wish to invest the effort involved in going shopping and cooking, and the agent may not enjoy cooking).
2. Go to a shop to buy some ready cooked food, heat up that food in the oven and eat the food. This may support the motivations associated with saving money to a lesser degree than option 1, eating healthy food to a lesser degree and eating enjoyable food to a lesser degree. In addition this may also support the motivation associated with idleness and enjoyment (i.e. the agent still has to go shopping but does not have to cook).
3. Go to a takeaway to buy ready cooked food. This may support the motivations associated with saving money to an even lesser degree than option 2, eating healthy food and eating enjoyable food to a greater extent than option 2 but to a lesser extent than option 1. In addition, this may support the motivation associated with idleness to a greater extent than option 2 (the agent doesn't even have to heat up the food) and enjoyment.
4. Go to a fast food place. This may support the motivations associated with saving money to a greater degree than option 1 but to a lesser degree than options 2 and 3, eating healthy food and eating enjoyable food to a lesser degree than options 1, 2 and 3. In addition, this may support the motivations associated with idleness (the agent doesn't have to cook) and enjoyment.
5. Go to a restaurant. This may undermine the motivations associated with saving money but support the motivations associated with eating healthy food and eating enjoyable food to a greater extent than options 1, 2, 3 and 4 above. This may also support the motivation associated with idleness and enjoyment.

The latter two options, going to a fast food place (4) and going to a restaurant (5) are different instantiations of an operator schema/action template which represents the activ-

ity of going out to eat. This illustrates how different instantiations of operator schemas/ action templates support or undermine the agent's motivations to varying degrees (see also chapter 5, section 5.2.3). As discussed in chapter 5, section 5.2.3 a look-up table is used during the step addition/simple establishment process, which indicates how different instantiations of various operator schemas/action templates support or undermine the agent's motivations.

Another example involves the goal of being in Edinburgh by 2pm. The options available to achieve this goal are listed below.

1. Fly by aeroplane - this may undermine the motivation associated with saving money, but support the motivations concerned with saving time and arriving at the destination feeling reasonably alert.
2. Travel by train - this may support the motivation associated with saving money, saving time and feeling reasonably alert (but to a lesser degree than option 1).
3. Drive - this may support the motivation associated with saving money (to a greater degree than option 2) but undermine the motivations associated with saving time and arriving feeling reasonably alert.
4. Travel by coach - this may support the motivation associated with saving money (to a greater degree than option 3), undermine the motivation concerned with saving time, and support (but to a lesser degree than options 1 and 2) the motivation associated with feeling reasonably alert.

6.4.4 The algorithm used to evaluate partial plans

When evaluating partial plans, the "Evaluate partial plans" component crudely predicts the agent's future motivations - i.e. the motivations that will arise as a consequence of executing the actions within each partial plan. In order to do this, it requires the current set of motivations together with information indicating the extent to which each action supports or undermines the agent's motivations. The representation used for actions includes two fields, *pros* and *cons,* where *pros* indicates the degree to which executing an action will support the agent's motivations, and *cons* indicates the degree to which executing that action will undermine the agent's motivations (see chapter 4, section 4.2.3). *pros* and *cons* both contain a set of tuples where each tuple contains the *name* associated with a motivation (this is used to uniquely identify a particular motivation), together with a value indicating the degree to which executing the action will support (if the tuple

belongs to the set *pros*) or undermine (if the tuple belongs to the set *cons*) the motivation identified by *name*.

The algorithm used by the "Evaluate partial plans" component (see table 6.13) deter-

Table 6.13 Evaluating partial plans

1. Let *support* be a value reflecting the degree of support that a partial plan *p* has for the set of motivations *motivations*. Initialise *support* to be the value 0.
2. For each motivation *motivation* belonging to the set *motivations*.
   (a) Let *newstrength* be a value indicating the predicted future strength associated with the motivation *motivation*. Initialise *newstrength* to be the value *strength* associated with the motivation *motivation*.
   (b) For each action *act* belonging to the partial plan *p*.
      i. Let *pro* be a measurement of the degree to which the action *act* supports the motivation *motivation*.
         A. If the action *act* contains a motivation *promot* belonging within its associated set *pros*, where the *name* associated with *promot* is equal to the *name* associated with the motivation *motivation*, *pro* is assigned the value *promot*.
         B. Else, the action *act* does not support the motivation *motivation* so *pro* is assigned the value nil.
      ii. Let *con* be a measurement of the degree to which the action *act* undermines the motivation *motivation*.
         A. If the action *act* contains a motivation *conmot* belonging within its associated set *cons*, where the *name* associated with *conmot* is equal to the *name* associated with the motivation *motivation*, *con* is assigned the value *conmot*.
         B. Else, the action *act* does not undermine the motivation *motivation* so *con* is assigned the value nil.
      iii. If *pro* has a value (i.e. *pro* is not nil),
         A. Let *newstrength* be the value obtained by subtracting the value *strength* associated with *pro* from the original value of *newstrength*, i.e.
            newstrength = newstrength – strength. This new value *newstrength* indicates a prediction of the future value *strength* associated with the motivation *motivation* after the action *act* has been executed. Continue with step 2. (b) above.
      iv. Else if *con* has a value (i.e. *con* is not nil),
         A. Let *newstrength* be the value obtained by adding the value *strength* associated with *con* to the original value of *newstrength*, i.e.
            newstrength = newstrength + strength. This new value *newstrength* indicates a prediction of the future value *strength* associated with the motivation *motivation* after the action *act* has been executed. Continue with step 2. (b) above.
      v. Else, continue with step 2. (b) above.
   (c) Set the value *support* to equal to the total obtained by adding the value *newstrength* (this value represents a prediction of the future strength of the motivation *motivation* - i.e. once each action belonging within the plan *p* has been executed) to the original value *support*, i.e. support = support + newstrength. Continue with stage 2. (a) above.
3. Return a tuple containing the partial plan *p* together with the value *support*.

mines the degree to which the actions within a partial plan support the agent's motivations by creating a cumulative score for each motivation - this score indicates the degree to which executing each action will support/undermine that motivation. These scores are

then totalled - the result is a value indicating the degree to which the partial plan supports the agent's motivations.

For each motivation in turn, the algorithm examines each action to determine whether it supports or undermines that motivation. If the action supports the motivation (i.e. if the set *pros* contains a tuple with the same *name* as the motivation), the value indicating the degree to which executing the action will support the motivation is subtracted from the score indicating the current strength associated with the agent's motivation. If the action undermines the motivation (i.e. the motivation belongs to the action's set *cons*), the value indicating the degree to which executing the action will undermine the motivation is added to the score indicating the current strength associated with the motivation. If the action neither supports nor undermines the motivation, the score remains unchanged. In this way, each motivation will have a cumulative score representing the degree to which the actions within the partial plan support or undermine that motivation. (This method is used to predict the future strength associated with each motivation as a consequence of executing the actions in the partial plan.) Finally, the cumulative scores associated with each motivation are totalled - the result is the measurement of the degree to which the partial plan supports the agent's motivations (a tuple containing the partial plan together with this result is returned by the "Evaluate partial plans" procedure"). Partial plans with low results are preferred (i.e. they best support the agent's motivations.

As well as determining the degree to which a partial plan supports the agent's motivations, several versions of the "Evaluate partial plans" heuristic have been implemented which take into account the following domain-independent information: the number of actions; the total duration of the partial plan (i.e. the estimated time it takes to execute the sequence of actions belonging to the partial plan); the number of outstanding goals (i.e. these are goals/subgoals which have not yet been achieved by the "Achieve goal" component); the number of achieved goals or subgoals (these have been achieved by the "Achieve goal" component).

6.4.5 A truck world example

In chapter 4, we demonstrated how a truck-driver agent was presented with the goal of delivering a *package* from *city1* to *city5* by 20 units of time, *at(package city5)* - see chapter 4, section 4.4.2, table 4.3. By examining the topology of the truck world domain (see chapter 4, section 4.4, figure 4.1) it can be seen that the truck-driver could use three alter-

native routes to achieve this goal.

1. Drive from *city1* to *city2* to *city5*.
2. Drive from *city1* to *city2* to *city4* to *city5*.
3. Drive from *city1* to *city3* to *city4* to *city5*.

A standard plan evaluation heuristic might take into account the number of steps in a

Table 6.14 Plans to achieve *at(package city5)*

action name:	pros		cons	
	name	strength	name	strength
Plan 1				
drive-truck(truck city1 city2)	*pleasure*	*0.1*	*conserve-fuel* *conserve-tyres*	*1.2* *1.0*
drive-truck(truck city2 city5)	*pleasure*	*0.2*	*conserve-fuel* *conserve-tyres*	*1.0* *0.9*
Plan 2				
drive-truck(truck city1 city2)	*pleasure*	*0.1*	*conserve-fuel* *conserve-tyres*	*1.2* *1.0*
drive-truck(truck city2 city4)	*pleasure*	*1.2*	*conserve-fuel* *conserve-tyres*	*0.3* *0.2*
drive-truck(truck city4 city5)	*pleasure*	*1.8*	*conserve-fuel* *conserve-tyres*	*0.3* *0.4*
Plan3				
drive-truck(truck city1 city3)	*pleasure*	*1.9*	*conserve-fuel* *conserve-tyres*	*0.3* *0.1*
drive-truck(truck city3 city4)	*pleasure*	*1.2*	*conserve-fuel* *conserve-tyres*	*0.3* *0.4*
drive-truck(truck city4 city5)	*pleasure*	*1.8*	*conserve-fuel* *conserve-tyres*	*0.3* *0.4*

partial plan - in this example, if such a method was used the first plan would be selected. However, the heuristic adopted by the planning/execution architecture assesses the degree to which each partial plan supports or undermines the agent's motivations in addition to taking into account the number of steps and the number of outstanding goals. Table 6.14 illustrates the actions used for each route and the degree of support they lend to the agent's motivations - the look-up table shown in chapter 4, section 4.4.6, table 4.7 shows how the values indicating the degree of support were obtained. Using the algorithm shown in table 6.13, and the current strength associated with the truck-driver's

motivations (see chapter 4, section 4.4.4, table 4.5), it can be seen that the first plan supports the truck-driver's motivations with the value 3.8, the second plan with the value 0.3 and the third plan with the value -3.1. The best partial plan is that with the lowest value indicating its support for the agent's motivations. The third partial plan is therefore chosen by the partial plan evaluation heuristic when planning to achieve the goal *at(package city5)* as shown in chapter 5, section 5.4.1 table 5.9, even though it has a high number of plan steps and takes longer to execute than the first partial plan.

6.4.6 Discussion

The current method adopted to evaluate partial plans with respect to the agent's motivations uses the current motivations and set of actions to predict the future motivations (i.e. the motivations that will arise as a consequence of executing each action). One limitation of this approach is that it treats each motivation as being equal in importance. In practice, motivations may not be equal - it may be more important for an agent to support (i.e. to maintain a low value of) one motivation than another. For example, if an agent creates a plan to attend an interview, if the agent wishes to perform well, it may be more important that they ensure that having travelled to the interview they remain alert than that they save money, in the process of executing the plan to achieve this goal. Actions that favour remaining alert (for example actions that allow the agent to sleep, such as travelling by train), may be preferred over those that allow the agent to save money (driving to the interview is cheaper but more tiring). The relative importance of each motivation depends upon the context of the plan. For example, if the agent was planning to reach a holiday destination, saving money might be more important than feeling alert. This feature is not currently implemented and requires further examination.

Finally, the following additional criteria may be used to determine good partial plans. Firstly, a good partial plan might achieve a large number of important goals. A plan which achieves a small number of goals of high importance may be better than one that achieves a large number of less important goals. Again, this feature is not currently implemented and requires work. Secondly, a good partial plan may be one containing a small number of actions that take very little time to execute - i.e. plans that have a lot of free time. However, this criteria very much depends upon the agent - one agent (a lazy agent) might prefer to have as much free time as possible, while another might prefer to be as busy as possible (such an agent might favour plans containing lots of actions and

might not be concerned as to how long those actions take to execute). Future work might focus upon how an agent's motivations influence its choice of "Evaluate partial plans" heuristic - different agent's might prefer to use different evaluation heuristics. This illustrates why it is difficult to implement good domain and agent-independent heuristics to guide the choice of partial plans.

6.5 Summary

In this chapter we described in detail three of the "Plan to achieve goal" components (shown in chapter 2, section 2.2.6, figure 2.2): "Estimate deadlines"; "Edit the partial plan"; Evaluate partial plans. In section 6.2 we described how deadlines are assigned to actions and subgoals belonging within a partial plan - in the current implementation such deadlines are pessimistic (i.e. are too early in the early stages of plan refinement). We also discussed problems that arise in using various temporal constraint propagation techniques to reason about time. In particular, the requirement that goals should be abandoned if there is insufficient time available to achieve them, means partial plans are edited. Temporal constraint propagation techniques require more sophisticated means of backtracking to deal with this problem. In section 6.3 the "Edit the partial plan" component was presented - this enables the planning/execution architecture to abandon the achievement of one or more goals, thereby allowing sufficient time to achieve the remaining goals. When plans are edited, there is a possibility that the search space might contain duplicate partial plans. Finally, in section 6.4, we described the "Evaluate partial plans" heuristic - one of the main problems with domain-independent planning is being able to define a "good" plan. It is our belief that "good" plans are context dependent - i.e. a plan considered to be "good" by one agent may be considered "bad" by another. This problem is overcome in part by evaluating partial plans with respect to the agent's motivations.

Chapter 7

Execution and Recovery

7.1 Introduction

In this chapter we describe what happens when an action is executed. If the "Select goal or action" component (shown in chapter 2, section 2.2.1, figure 2.1) chooses to execute an action, that action is passed to the "Execute action" procedure which is responsible for updating the partial plan to reflect the outcome of execution. In chapter 4, section 4.5 we described how an action may only be selected for execution if it is fully instantiated, its preconditions are true, and it is possibly first within the partial order of actions. When the action is executed, changes occur within the environment.

Figure 7.1 is a more detailed diagram which illustrates what happens if the "Select goal or action" component chooses to execute an action. Once the action has been executed, a user keys in the resulting state of the environment as well as the time. In addition to modelling the changes that occur as a consequence of executing the action, the user may also model changes that are brought about by the activities of other agents or physical processes. (Note that in order to perceive changes that occur within its environment a fully implemented agent would require sophisticated sensors as well as the ability to translate raw sensor data into symbols. This is beyond the scope of the work presented in this thesis which is why such information is user-supplied.)

The "Execute action" component ensures that the agent's internal partial plan is updated to reflect the outcome of execution - this involves updating the agent's representation of its environment and removing the action which has been executed together with certain temporal, binding and achieved goals/subgoals.

The "Recover" component checks to see if the outcome of execution is as expected, and, if not (one or more goals or subgoals which have been achieved may have been

undone/clobbered or the action may take longer to execute than expected), repairs the partial plan.

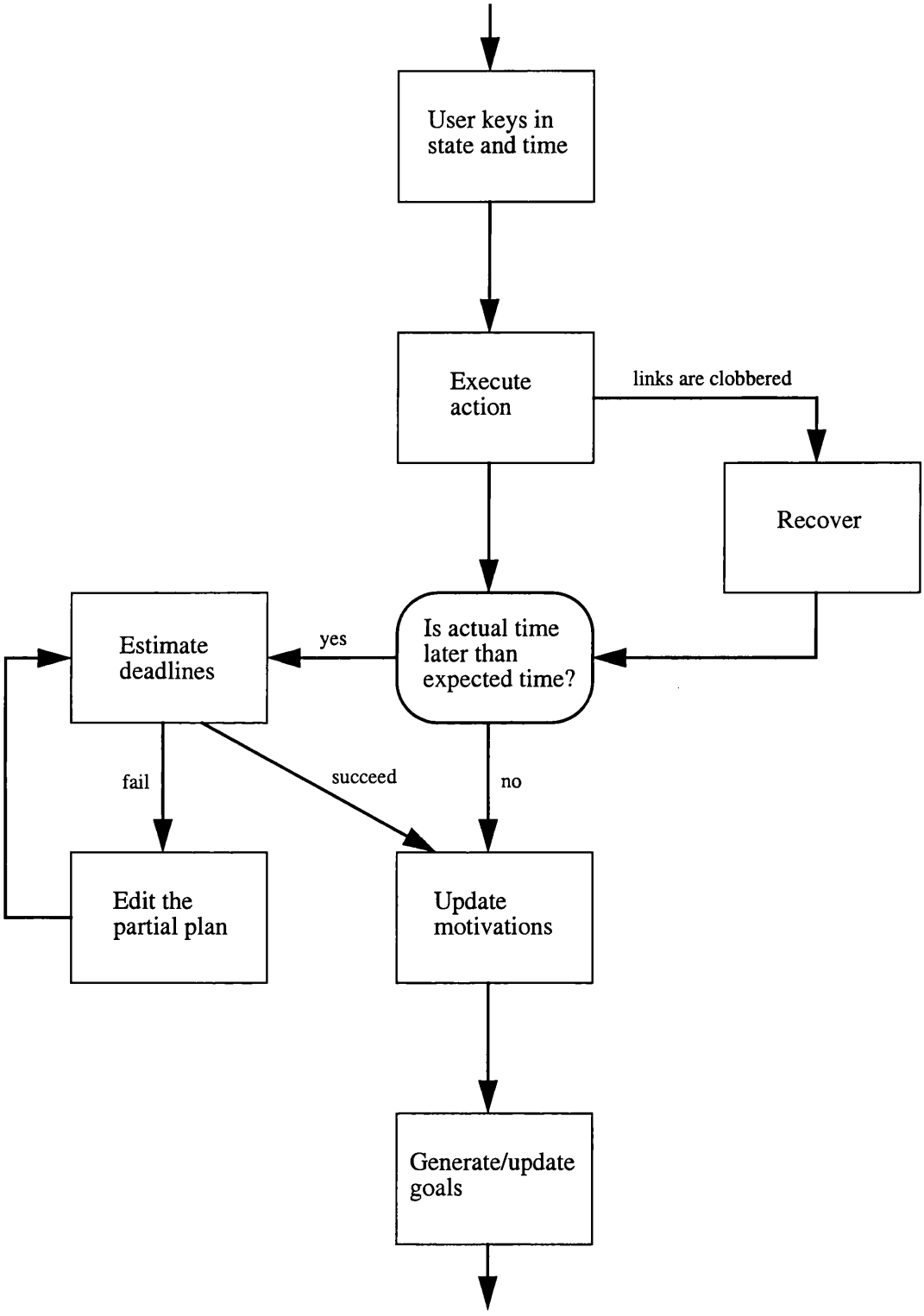


Figure 7.1 Executing an Action

Once the partial plan's model of the environment is updated to reflect the outcome of execution, we effectively have a new "initial state" for planning purposes. It is therefore

necessary to prune the search space of partial plans by deleting all other previously generated partial plans. The changes that occur within the environment as a consequence of

Table 7.1 Executing an action

1. Let *exact* be the action which is to be executed.
2. Let *newstate* be the actual state of the world following execution (this is keyed in by a user). *newstate* represents all changes that have occurred in the world following execution, including those that arise as a consequence of executing *exact* (which may result in an unforeseen outcome) as well as those that arise as a consequence of the activities of other agents or physical processes (see section 7.2.1).
3. Let *posttime* be the actual time following execution (this is keyed in by a user).
4. Let *latestft* be the estimated time by which execution of the action *exact* should be complete. (*latestft* is calculated by adding the *duration* associated with *exact* to the execution deadline associated with *exact*.)
5. Let *cloblinks* be the set of plan links which have been clobbered following execution of the action *exact*. (Plan links may be clobbered if the world changes in unforeseen ways following execution.)
6. Let *achgoals* be the set of goals which have been achieved following execution.
7. The partial plan *p* is updated by the following procedures.
   (a) The set of actions belonging to the partial plan *p* is updated using *newstate* (see section 7.2.3).
   (b) The set of persistence constraints or links is updated using the sets *cloblinks* and *achgoals* (see section 7.2.4 below).
   (c) The set of ordering constraints is updated using the set *achgoals* (see section 7.2.4 below).
   (d) The set of binding constraints is updated (see section 7.2.4 below).
   (e) The set of open conditions may need updating if the set *cloblinks* is not empty (i.e. if one or more links have been clobbered following execution of the action *exact*). The clobbered links must be converted into a set of open conditions which are appended to the set of open conditions already within the plan *p* (see section 7.3 below).
   (f) The set of goals is updated if the set *achgoals* is not empty (i.e. if one or more goals have been achieved following execution).
8. If the time following execution, *posttime*, is later than the estimated execution finishing time, *latestft*.
   (a) Estimate the deadlines associated with each of the actions belonging to the newly updated partial plan.
   (b) If there is insufficient time available to achieve each of the goals in the newly updated partial plan.
      i. Edit the partial plan.
      ii. Continue with stage 8. (a) above (i.e. estimate the deadlines associated with each of the actions belonging to the edited partial plan).
   (c) Else, return the modified partial plan.
9. Else, return the newly updated partial plan.

execution (and as a consequence of the activities of other agents and physical processes) will directly affect the agent's motivations which in turn might lead to the generation of new goals or cause existing goals to be updated. Again, because a detailed implementa-

tion of the components "Update motivations" and "Generate/update goals" are beyond the scope of this thesis, a user is responsible for both updating the motivations and generating new or updating existing goals. In the following sections we describe each of these components in more detail. The algorithm used in the implementation of the "Execute action" and "Recover" components is presented in table 7.1.

7.2 Executing an action

The task of the "Execute action" component of figure 7.1 is to update the partial plan to reflect the state of the environment following execution and to update the search space of partial plans. This consists of the procedures described in the following sections.

7.2.1 Updating the agent's model of the environment

The agent's model of the environment must be updated to reflect the changes brought about as a consequence of execution as well as due to the activities of other agents and physical processes. Once an action has been executed, the environment will have changed - the agent perceives these changes and updates its world model to reflect those changes. In the current implementation, a user supplies a set of predicates which represent the state of the environment following execution (see table 7.1, stage 2.). This means that unforeseen changes to the environment may be represented - for example, execution may not result in the expected outcome, or other agents or physical processes may have made changes to the world. The user-supplied set of predicates (which represent the state of the environment following execution) are used in order to update the "initial state" action belonging within the partial plan as this contains the agent's model of the current state of the environment (see section 7.2.3 below).

In practice, changes to the environment may occur while the action is being executed. Actions take time to execute, so, although the preconditions of an action must remain true at least until execution commences, an action's effects may become true at any time during execution. We assume however, that an action's effects are true once execution is complete (i.e. we do not model the exact point during execution at which an action's effects become true).

7.2.2 Updating the time

The agent's model of the environment must be updated to reflect the actual time following execution. In the current implementation, this time is supplied by a user (see table

7.1, stage 3.). The actual time may differ from the predicted time as execution may take longer or shorter than anticipated.

7.2.3 Updating the set of actions

The set of actions belonging to the current partial plan must be updated to reflect the outcome of execution. Firstly, the action *act* which has been executed must be removed from this set of actions. Secondly, the "initial state" action (which is the agent's model of the current state of the environment) must be updated, using the user-supplied set of predicates (see figure 7.1 and table 7.1, stage 2.), to reflect the changes that have occurred within the environment following execution.

7.2.4 Updating binding, temporal and persistence constraints

Binding, temporal and persistence constraints (these represent goals or subgoals which have been achieved by the "Plan to achieve goal" component) must be updated (see table 7.1, stage 7. (d), 7. (c) and 7. (b) respectively). All binding constraints which contain variables associated with the action *act* (which has been executed) must be modified so that those variables are removed (this may mean that some binding constraints become redundant and must also be removed).

Certain persistence and temporal constraints also become redundant following execution and must be removed from the partial plan. In particular, persistence constraints which maintain the truth of the preconditions associated with the action *act* are redundant once *act* has been executed. Once established, the preconditions associated with *act* must only remain true until the point at which execution commences. This means that once execution is complete, the persistence constraints that are responsible for maintaining the truth of those preconditions are no longer required. Likewise, persistence constraints which maintain the truth of any goals that have been achieved following execution (see table 7.1, stage 6.), are now redundant and must be removed from the partial plan. In addition, *prior* to executing *act*, the partial plan will contain a temporal constraint which specifies that the "initial state" action (this action represents the state of the world *prior* to executing *act*) must occur prior to *act*. Once *act* has been executed, this constraint must be removed from the partial plan. Likewise, if any goals have been achieved following execution, all temporal constraints specifying the ordering of such goals must be removed from the partial plan.

Finally, some of the remaining temporal and persistence constraints must be modi-

fied to reflect the fact that the action *act* has been removed from the partial plan following execution. For example, the action *act* may have been chosen previously by the "Plan to achieve goal" component in order to achieve the subgoal *g*. Once *act* is executed, *g* becomes true and *act* is removed from the partial plan. The persistence constraint representing the fact that the subgoal *g* is established by *act*, must be modified to reflect the fact that now *g* is true, it is the newly updated "initial state" action (which has been newly updated - see section 7.2.3 - to reflect the changes made to the environment) which now establishes *g*. In addition, all temporal constraints that specified (prior to execution) that the action *act* is ordered to occur prior to some other action, must be modified to specify that, following execution, it is now the newly updated "initial state" action that occurs prior to the other action. (A consequence of executing *act* is that the environment results in a state that is captured within the newly updated "initial state" action.)

7.2.5 Removing goals which have been achieved

All goals that have been achieved as a consequence of execution must be removed from the partial plan - see table 7.1, stage 7. (f).

7.2.6 Updating the search space of partial plans

Following execution of some action *act*, all previously generated partial plans must be removed from the search space. When the "Plan to achieve goal" component plans to achieve a goal or subgoal, a search space of partial plans is generated in which each partial plan contains the same "initial state" action (i.e. an action which represents the current state of the world) and in which each partial plan achieves the goal or subgoal (using different actions). The "Select best partial plan" component then selects the "best" partial plan for subsequent refinement and, if the "Select goal or action" component decides to achieve a goal or subgoal belonging to the "best" partial plan, a new set of partial plans which achieve that goal or subgoal is generated by the "Plan to achieve goal" component. These partial plans all contain the same "initial state" action and are added to the search space of partial plans. The "Select best partial plan" component chooses the best partial plan from this search space of partial plans.

However, when an action is executed, the world changes, which means the "initial state" action belonging within all previously generated partial plans in the search space no longer corresponds to the actual state of the world. The "Select best partial plan" component cannot choose to refine a previously generated partial plan whose "initial

state" no longer corresponds to the actual state of the world, as this partial plan is no longer valid. This means that once an action has been executed, all previously generated

Table 7.2 Plan 4 following execution of *drive-truck(truck city4 city2)*

actions:	id:	name:			
	9 *11* *4* *8* *2*	*load-truck(parcel truck city2)* *drive-truck(truck city2 city4)* *drive-truck(truck city4 city5)* *unload-truck(parcel truck city5)* *unload-truck(package truck city5)*			
links	establisher:	condition:	consumer:		
	11 *0* *9* *8* *2* *4* *0* *4* *0* *0* *0* *0* *0* *0*	*at(truck city4)* *at(truck city2)* *in(parcel truck)* *at(parcel city5)* *at(package city5)* *at(truck city5)* *at(parcel city2)* *at(truck city5)* *connects(city2 city4)* *has-fuel(truck)* *at(truck city2)* *in(package truck)* *connects(city4 city5)* *has-fuel(truck)*	*4* *9* *8* *7* *1* *2* *9* *8* *11* *11* *11* *2* *4* *4*		
open:	*nil*				
unsafe:	*nil*				
ordering:	*load-truck(parcel truck city2) -> drive-truck(truck city2 city4) ->* *drive-truck(truck city4 city5) ->* *unload-truck(parcel truck city5),* *unload-truck(package truck city5)* *(9 -> 11 -> 4 -> 8 -> 7) & (4 -> 2 -> 1)*				
bindings:	*<varset {(?from11 city2 ?loc9)} = city2 not(?to11 ?truck11 ?truck9 ?ob9)>* *<varset {(?truck4 ?truck2 truck ?truck9 ?truck8 ?truck11)} = truck not(?from4 ?to4 ?loc2 ?ob2 ?ob9 loc9 ?loc8 ?ob8 ?to11 ?from11)>* *<varset {(?from4 city4 ?to11)} = city4 not(?to4 ?truck4 ?truck11 ?from11)>* *<varset {(?ob8 parcel ?ob9)} = parcel not(?truck8 ?loc8 ?loc9 ?truck9)>* *<varset {(?to4 city5 ?loc2 ?loc8)} = city5 not(?from4 ?truck4 ?truck2 ?ob2 ?truck8 ?ob8)>* *<varset {(?ob2 package)} = package not(?truck2 ?loc2)>*				
goals:	id:	conditions:	deadline:	effort:	importance:
	1 *7*	*at(package city5)* *at(parcel city5)*	*20* *16*	*21* *14*	*6* *20*

partial plans in the search space must be deleted. To continue the planning/execution process, a new search space is created which consists solely of the partial plan (which has been newly updated by the "Execute action" component to reflect the outcome of executing the action *act*).

7.2.7 A truck world domain example

Table 7.2 shows the partial plan that arises as a consequence of executing the action *drive-truck(truck city4 city2)* (see chapter 5, section 5.4.2, table 5.13) which belongs to the partial plan shown in chapter 5, section 5.4.2, table 5.16. The outcome of execution is as predicted which means that the current time (following execution) is 8 and the state following execution, which is supplied by a user, is that shown in table 7.3. The "Select

Table 7.3 Initial action for Plan 4, time = 8

id:	name:	add:
0	*initial*	*connects(city1 city2) & connects(city2 city1) &* *connects(city1 city3) & connects(city3 city1) &* *connects(city2 city4) & connects(city4 city2) &* *connects(city2 city5) & connects(city5 city2) &* *connects(city3 city4) & connects(city4 city3) &* *connects(city4 city5) & connects(city5 city4) &* *has-fuel(truck) &* *at(truck city2) &* *in(package truck) &* *at(parcel city2)*

goal or action" component takes the partial plan shown in table 7.2 and chooses to execute the action *load-truck(parcel truck city2)*. We first demonstrate how a partial plan is updated when execution of the action *load-truck(parcel truck city2)* results in the intended outcome. In sections 7.3.1 and 7.3.2 below we demonstrate how a partial plan is

Table 7.4 Initial action for Plan 5, time = 9

id:	name:	add:
0	*initial*	*connects(city1 city2) & connects(city2 city1) &* *connects(city1 city3) & connects(city3 city1) &* *connects(city2 city4) & connects(city4 city2) &* *connects(city2 city5) & connects(city5 city2) &* *connects(city3 city4) & connects(city4 city3) &* *connects(city4 city5) & connects(city5 city4) &* *has-fuel(truck) &* *at(truck city2) &* *in(package truck) &* *in(parcel truck)*

updated when execution goes awry.

Once the action *load-truck(parcel truck city2)* has been executed successfully, the

Table 7.5 Plan 5 - *load-truck(parcel truck city2)* goes as expected, time=9

actions:	id:	name:			
	11 *4* *8* *2*	*drive-truck(truck city2 city4)* *drive-truck(truck city4 city5)* *unload-truck(parcel truck city5)* *unload-truck(package truck city5)*			
links	establisher:	condition:	consumer:		
	11 *0* *8* *2* *4* *4* *0* *0* *0* *0* *0* *0*	*at(truck city4)* *in(parcel truck)* *at(parcel city5)* *at(package city5)* *at(truck city5)* *at(truck city5)* *connects(city2 city4)* *has-fuel(truck)* *at(truck city2)* *in(package truck)* *connects(city4 city5)* *has-fuel(truck)*	*4* *8* *7* *1* *2* *8* *11* *11* *11* *2* *4* *4*		
open:	*nil*				
unsafe:	*nil*				
ordering:	*drive-truck(truck city2 city4) -> drive-truck(truck city4 city5) ->* *unload-truck(parcel truck city5),* *unload-truck(package truck city5)* *(11 -> 4 -> 8 -> 7) & (4 -> 2 -> 1)*				
bindings:	*<varset {(?from11 city2)} = city2 not(?to11 ?truck11)>* *<varset {(?truck4 ?truck2 truck ?truck8 ?truck11)} = truck not(?from4 ?to4 ?loc2 ?ob2 ?loc8 ?ob8 ?to11 ?from11)>* *<varset {(?from4 city4 ?to11)} = city4 not(?to4 ?truck4 ?truck11 ?from11)>* *<varset {(?ob8 parcel)} = parcel not(?truck8 ?loc8 ?loc9)>* *<varset {(?to4 city5 ?loc2 ?loc8)} = city5 not(?from4 ?truck4 ?truck2 ?ob2 ?truck8 ?ob8)>* *<varset {(?ob2 package)} = package not(?truck2 ?loc2)>*				
goals:	id:	conditions:	deadline:	effort:	importance:
	1 *7*	*at(package city5)* *at(parcel city5)*	*20* *16*	*21* *14*	*6* *20*

current time following execution is 9 (the action only takes 1 unit of time to execute) and the state of the world following execution is shown in table 7.4. Both the time and state of the world following execution are supplied by a user.

Table 7.5 illustrates the partial plan that results following the successful execution of

the action *load-truck(parcel truck city2)*.It can be seen that the action *load-truck(parcel truck city2)* together with its associated links, ordering constraints and bindings have been removed from the partial plan shown in table 7.2.

7.3 The Recovery Component

Once an action is executed, the outcome may not be as expected. The task of the "Recover" component (see figure 7.1) is to determine whether the outcome of execution is as expected and, if not, to repair the partial plan. When an action is executed, the outcome may not be as expected for two reasons.

1. Executing the action could result in an unexpected outcome and/or the world could change in unforeseen ways due to the activities of other agents or physical processes.
2. Execution could take longer than expected.

Figure 7.1 illustrates what happens when the outcome of execution is not as expected. In the following paragraphs we describe in detail how the "Recover" component deals with both scenarios.

The actual state of the environment following execution may differ from the predicted state either because executing the action could go wrong, or because other agents or physical processes may change the environment in unforeseen ways. In particular, one or more of the action's effects may not become true following execution (for example, an action which involves picking up a block could fail so that the block has not been picked up following execution), or facts which were previously made true as a consequence of executing previous actions may be accidentally undermined or undone following execution (for example, a block which was previously picked up may be knocked out of a robot's grippers by another agent). This means that plan links or persistence constraints (these represent goals or subgoals which have previously been achieved by the "Plan to achieve goal" component) within the partial plan may be violated or clobbered, in which case subgoals (i.e. the preconditions of subsequent actions) or goals within the partial plan which should be true following execution, may not actually be true. One of the tasks of the "Recover" component is to determine whether any of the persistence constraints belonging to the partial plan have been clobbered/violated following execution (see table 7.1, stage 5.) and, if so, to repair the partial plan. The repair process involves converting

the clobbered persistence constraints into a set of goals or subgoals (to be reachieved) which are added to the set of open conditions belonging to the partial plan (see table 7.1, stage 7. (e)), and removing the clobbered persistence constraints from the set of persistence constraints belonging within the partial plan (see table 7.1, stage 7. (b)).

If execution takes longer than expected, the agent could miss the "latest" execution deadline associated with subsequent actions which in turn means the agent may not be able to achieve various goals by their deadlines. Another task of the "Recover" component is to determine whether, as a consequence of execution taking longer than anticipated, it is still possible to achieve all goals by their associated deadlines. The "Recover" component compares the actual time following execution with the predicted time (the predicted time is determined by adding the estimated duration of the action which has been executed to the estimated execution deadline associated with that action - see table 7.1, stage 8.) - if the actual time is later than the predicted time, the "Estimate deadlines" component (described in chapter 6, section 6.2) reestimates the deadlines associated with each action in order to determine whether there is sufficient time available to achieve each goal (see table 7.1, stage 8. (a)). If there is insufficient time (i.e. the "Estimate deadlines" procedure returns the value :fail) available to achieve each goal by its associated deadline, the "Edit the partial plan" component (described in chapter 6, section 6.3) edits the partial plan to remove one goal (the goal which caused the deadline estimation routine to fail) and its associated actions and constraints from the partial plan - see table 7.1, stage 8. (b). Once the plan has been edited, the "Estimate deadlines" component reestimates the deadlines associated with the remaining actions in the plan in order to determine whether there is sufficient time available to achieve each remaining goal. This process is repeated (see figure 7.1) until the "Estimate deadlines" component is able to successfully assign deadlines to each of the actions within the partial plan.

7.3.1 An example of execution failure - the parcel fails to load

In this section we show how the partial plan of table 7.2, i.e. the partial plan prior to executing the action *load-truck(parcel city2)*, is updated when the *parcel* fails to be loaded into the *truck* at *city2*. This could be either due to the truck-driver failing to successfully execute the action, or because some other malicious agent unloaded the *parcel* once it was loaded into the *truck*. The current time following execution is 9 while the current state is shown in table 7.6 which indicates that the *parcel* is still at *city2*, *at(parcel city2)*,

as opposed to being in the *truck.*

Table 7.6 Initial action for Plan 6, time = 9 (fail to load parcel)

id:	name:	add:
0	*initial*	*connects(city1 city2) & connects(city2 city1) & connects(city1 city3) & connects(city3 city1) & connects(city2 city4) & connects(city4 city2) & connects(city2 city5) & connects(city5 city2) & connects(city3 city4) & connects(city4 city3) & connects(city4 city5) & connects(city5 city4) & has-fuel(truck) & at(truck city2) & in(package truck) & at(parcel city2)*

Table 7.8 shows the partial plan which results following execution. One of the consequences of execution not resulting in the expected outcome, is that the plan link *(9 in(parcel truck) 8)* belonging to the partial plan prior to execution (which is shown in table 7.2) is undone or clobbered - the link means that the action *load-truck(parcel truck city2)* was supposed to establish the precondition *in(parcel truck)* belonging to the action *unload-truck(parcel truck city5).* The clobbered link is converted into a subgoal or open condition to be reachieved.

The "Select goal or action" component chooses to achieve this open condition *in(parcel truck)* which means that a new action *load-truck(parcel truck city2)* with the unique identifier 12 is created (see table 7.7) and added to the partial plan of table 7.2 - see the newly updated partial plan in table 7.8. The resulting partial plan is moreorless

Table 7.7 New action for Plan 6

id:	12
name:	*load-truck(parcel truck city2)*
precondition:	*at(truck city2) & at(parcel city2)*
delete:	*at(parcel city2)*
add:	*in(parcel truck)*
goals:	*(7)*
pros:	*nil*
cons:	*nil*
duration:	*1*
importance:	*20*
effort:	*17*

identical to the partial plan created prior to executing *load-truck(parcel truck city2)* (see table 7.2 except that a new version of *load-truck(parcel truck city2)* has been created and that different deadlines have been assigned to each action. Table 7.9 below shows the

Table 7.8 Plan 6 - *load-truck(parcel truck city2)* fails

actions:	id:	name:			
	12 *11* *4* *8* *2*	*load-truck(parcel truck city2)* *drive-truck(truck city2 city4)* *drive-truck(truck city4 city5)* *unload-truck(parcel truck city5)* *unload-truck(package truck city5)*			
links	establisher:	condition:	consumer:		
	11 *0* *12* *8* *2* *4* *0* *4* *0* *0* *0* *0* *0* *0*	*at(truck city4)* *at(truck city2)* *in(parcel truck)* *at(parcel city5)* *at(package city5)* *at(truck city5)* *at(parcel city2)* *at(truck city5)* *connects(city2 city4)* *has-fuel(truck)* *at(truck city2)* *in(package truck)* *connects(city4 city5)* *has-fuel(truck)*	*4* *12* *8* *7* *1* *2* *12* *8* *11* *11* *11* *2* *4* *4*		
open:	*nil*				
unsafe:	*nil*				
ordering:	*load-truck(parcel truck city2) -> drive-truck(truck city2 city4) ->* *drive-truck(truck city4 city5) ->* *unload-truck(parcel truck city5),* *unload-truck(package truck city5)* *(12 -> 11 -> 4 -> 8 -> 1) & (4-> 2 ->1)*				
bindings:	*<varset {(?loc12 ?from11 city2)} = city2 not(?ob12 ?truck12 ?to11 ?truck11)>* *<varset {(?truck12 ?truck4 ?truck2 truck ?truck8 ?truck11)} = truck not(?ob12 ?loc12 ?from4 ?to4 ?loc2 ?ob2 ?loc8 ?ob8 ?to11 ?from11)>* *<varset {(?from4 city4 ?to11)} = city4 not(?to4 ?truck4 ?truck11 ?from11)>* *<varset {(?ob8 parcel ?ob12)} = parcel not(?truck8 ?loc8 ?loc12 ?truck12)>* *<varset {(?to4 city5 ?loc2 ?loc8)} = city5 not(?from4 ?truck4 ?truck2 ?ob2 ?truck8 ?ob8)>* *<varset {(?ob2 package)} = package not(?truck2 ?loc2)>*				
goals:	id:	conditions:	deadline:	effort:	importance:
	1 *7*	*at(package city5)* *at(parcel city5)*	*20* *16*	*21* *17*	*6* *20*

deadlines that have been assigned to each action belonging to the partial plan in table 7.8. The deadlines that were previously assigned to each action can be seen in chapter 6, section 6.2.6, table 6.9.

Table 7.9 Deadlines are assigned to actions

id:	type:	name:	duration:	deadline:
12	*:action*	*load-truck(parcel truck city2)*	*1*	*9*
11	*:action*	*drive-truck(truck city2 city4)*	*2*	*10*
4	*:action*	*drive-truck(truck city4 city5)*	*3*	*12*
8	*:action*	*unload-truck(parcel truck city5)*	*1*	*15*
2	*:action*	*unload-truck(package truck city5)*	*1*	*15*
1	*:goal*	*at(package city5)*	*0*	*20*
7	*:goal*	*at(parcel city5)*	*0*	*16*

7.3.2 Another example - execution takes longer than expected

In this section we describe what happens when execution takes longer than anticipated. Taking the partial plan generated in table 7.8 in the previous section, we attempt to execute the newly created action *load(parcel truck city2)*, with the unique identifier 12, which was created to reachieve the clobbered plan link *(9 in(parcel truck) 8)* of table 7.2. This time, the truck-driver succeeds in loading the *parcel* into the *truck* at *city2*, but takes longer than anticipated to do so. The current time following execution is 11 (this is supplied by a user), when it was anticipated to be 10. Table 7.10 illustrates the current state of the environment following execution.

Table 7.10 Initial action for Plan 7, time = 11

id:	name:	add:
0	*initial*	*connects(city1 city2) & connects(city2 city1) &* *connects(city1 city3) & connects(city3 city1) &* *connects(city2 city4) & connects(city4 city2) &* *connects(city2 city5) & connects(city5 city2) &* *connects(city3 city4) & connects(city4 city3) &* *connects(city4 city5) & connects(city5 city4) &* *has-fuel(truck) &* *at(truck city2) &* *in(package truck) &* *in(parcel truck)*

Table 7.11 illustrates how the partial plan of table 7.8 has been initially updated to reflect the outcome of execution. It can be seen in this table that the action *load-truck(parcel truck city2)* together with its associated constraints have been removed from

the partial plan shown in table 7.8.

Table 7.11 Plan 7 - *load-truck(parcel truck city2)* has been executed

actions:	id:	name:			
	11 *4* *8* *2*	*drive-truck(truck city2 city4)* *drive-truck(truck city4 city5)* *unload-truck(parcel truck city5)* *unload-truck(package truck city5)*			
links	establisher:	condition:	consumer:		
	11 *0* *8* *2* *4* *4* *0* *0* *0* *0* *0* *0*	*at(truck city4)* *in(parcel truck)* *at(parcel city5)* *at(package city5)* *at(truck city5)* *at(truck city5)* *connects(city2 city4)* *has-fuel(truck)* *at(truck city2)* *in(package truck)* *connects(city4 city5)* *has-fuel(truck)*	*4* *8* *7* *1* *2* *8* *11* *11* *11* *2* *4* *4*		
open:	*nil*				
unsafe:	*nil*				
ordering:	*drive-truck(truck city2 city4) -> drive-truck(truck city4 city5) ->* *unload-truck(parcel truck city5),* *unload-truck(package truck city5)* *(11 -> 4 -> 8 -> 7) & (4 -> 2 -> 1)*				
bindings:	*<varset {(?from11 city2)} = city2 not(?to11 ?truck11)>* *<varset {(?truck4 ?truck2 truck ?truck8 ?truck11)} = truck not(?from4 ?to4 ?loc2 ?ob2 ?loc8 ?ob8 ?to11 ?from11)>* *<varset {(?from4 city4 ?to11)} = city4 not(?to4 ?truck4 ?truck11 ?from11)>* *<varset {(?ob8 parcel)} = parcel not(?truck8 ?loc8 ?loc9)>* *<varset {(?to4 city5 ?loc2 ?loc8)} = city5 not(?from4 ?truck4 ?truck2 ?ob2 ?truck8 ?ob8)>* *<varset {(?ob2 package)} = package not(?truck2 ?loc2)>*				
goals:	id:	conditions:	deadline:	effort:	importance:
	1 *7*	*at(package city5)* *at(parcel city5)*	*20* *16*	*21* *17*	*6* *20*

Once the partial plan of table 7.8 has been updated to reflect the outcome of execu-

Table 7.12 Plan 8 - *load-truck(parcel truck city2)* takes longer than expected

actions:	id:	name:			
	11 *4* *2*	*drive-truck(truck city2 city4)* *drive-truck(truck city4 city5)* *unload-truck(package truck city5)*			
links	establisher:	condition:	consumer:		
	11 *2* *4* *0* *0* *0* *0* *0* *0*	*at(truck city4)* *at(package city5)* *at(truck city5)* *connects(city2 city4)* *has-fuel(truck)* *at(truck city2)* *in(package truck)* *connects(city4 city5)* *has-fuel(truck)*	*4* *1* *2* *11* *11* *11* *2* *4* *4*		
open:	*nil*				
unsafe:	*nil*				
ordering:	*drive-truck(truck city4 city2) -> drive-truck(truck city4 city5) -> unload-truck(package truck city5)* *(12 -> 11 -> 4 -> 2)*				
bindings:	*<varset {(?from11 city2)} = city2 not(?to11 ?truck11)>* *<varset {(?truck4 ?truck2 truck ?truck11)} = truck not(?from4 ?to4 ?loc2 ?ob2 ?to11 ?from11)>* *<varset {(?from4 city4 ?to11)} = city4 not(?to4 ?truck4 ?truck11 ?from11)>* *<varset {(?loc2 ?to4 city5)} = city5 not(?from4 ?truck4 ?truck2 ?ob2)>* *<varset {(?ob2 package)} = package not(?truck2 ?loc2)>*				
goals:	id:	conditions:	deadline:	effort:	importance:
	1	*at(package city5)*	*20*	*21*	*6*

tion (see table 7.11), the "Estimate deadline" procedure reestimates the deadlines associated with each action because the actual time following execution, 11, is later than the predicted time, 10. In this example, the "Estimate deadline" procedure fails to assign deadlines to each of the actions belonging to the partial plan because there is insufficient time available to achieve the goal *at(parcel city5)* by its deadline 16. The "Edit the partial plan" component therefore edits the updated partial plan to remove the goal *at(parcel city5)* together with its associated actions and constraints. In this example, the "Edit the partial plan" process removes the action *unload(parcel truck city5)* with the unique identifier 8 together with its associated constraints from the updated partial plan as it is the

only action that contributes solely towards the goal *at(parcel city5)*. Once the partial plan has been edited, the "Estimate deadline" process reassigns deadlines to the remaining actions within the plan as shown in table 7.13. Note that in this example, the *parcel* is in the *truck* even though there is insufficient time available to deliver the parcel to *city5* by the required deadline.

Table 7.13 Deadlines are assigned to actions

id:	type:	name:	duration:	deadline:
11	*:action*	*drive-truck(truck city2 city4)*	*2*	*14*
4	*:action*	*drive-truck(truck city4 city5)*	*3*	*16*
2	*:action*	*unload-truck(package truck city5)*	*1*	*19*
1	*:goal*	*at(package city5)*	*0*	*20*

7.3.3 Discussion

The techniques used to enable the planning/execution architecture to recover, should execution not result in the intended outcome, are similar to those used by PLANEX [Fikes et al 72]. The differences in approach are due to the different planning algorithms - PLANEX repaired linear plans generated by STRIPS which do not record persistence constraints. PLANEX did this with the aid of triangle tables which contained records of the dependencies between actions and preconditions.

When one or more persistence constraints or plan links have been undone/clobbered as a consequence of execution, the current implementation of the "Recover" component repairs the partial plan by converting the clobbered persistence constraints into a set of goals/subgoals which are to be reachieved at a later stage by the "Plan to achieve goal" component. This is not necessarily the most efficient way of repairing the partial plan as it involves making local repairs by reachieving the preconditions or subgoals associated with actions currently within the partial plan. This method effectively patches the existing plan, making it longer by incorporating new actions which are added to reachieve the previously violated preconditions or subgoals. Instead of reachieving violated preconditions or subgoals, it may be more efficient instead to determine which goals each violated precondition or subgoal contributes towards, to edit the partial plan to remove these goals together with their associated actions and constraints, and then to plan to reachieve these goals. Instead of patching the partial plan (which results in a longer plan) this approach plans to reachieve goals from scratch which may result in a shorter plan.

In addition, the actual outcome following execution can also present the following opportunities: propositions that were expected to have been denied (or made untrue) by executing an action, may not have been denied (i.e. they may still be true); effects may be established (or made true) unexpectedly which could lead to goals or subgoals being achieved prematurely. These opportunities mean there could be a number of redundant achievers within the partial plan. (By "achiever" we mean an action that achieves some goal or subgoal. Normally, the "Plan to achieve goal" component selects an action to achieve a goal or subgoal. However, if execution goes awry, the newly updated "initial state" action may establish or achieve some goal or subgoal that was to be achieved at a later stage by executing some other action. This means that the action that was previously selected to achieve that goal or subgoal may now be redundant.) The current implementation does not address the issue of there being redundant achievers within the partial plan and so does not take advantage of opportunities.

7.4 Updating the Motivations

When an action is executed, the world changes, both as a consequence of execution and as a consequence of other agents or physical processes acting within the world. The change in the world/environment directly affects the strength associated with the agent's motivations (see chapter 2, section 2.2.3). The "Update motivations" component is responsible for updating the strength associated with each motivation to reflect the changes within the agent's environment as well as changes made to the agent's partial plan. This component has not been implemented and is beyond the scope of this work. However, by examining the sets *pros* and *cons* associated with the action which has been executed, it is possible to calculate a new value indicating the strength associated with each motivation. *pros* and *cons* both contain a set of tuples where each tuple consists of two fields: *name,* which indicates a motivation which the action, when executed, supports (if the tuple belongs to the set *pros*) or undermines (if the tuple belongs to the set *cons*); *strength*, a value indicating the degree to which the action, when executed, supports or undermines the motivation identified by *name* (see chapter 4, section 4.2.3). The algorithm shown in table 7.14 enables the strength associated with each motivation to be calculated using the fields *pros* and *cons* associated with the action which has been exe-

cuted.

Table 7.14 Updating the motivations

1. Let *exact* be the action which has been executed.
2. For each motivation *motivation* belonging to the agent's set of *motivations*.
   (a) If *exact* supports the motivation *motivation* (the action *exact* supports the motivation *motivation* if there is a tuple *promot* belonging to the set *pros* associated with *exact* whose value *name* is the equal to the value *name* associated with the motivation *motivation*), subtract the value indicating the degree to which *exact* supports the motivation *motivation* (i.e. subtract the value *strength* associated with the tuple *promot*) from the value *strength* associated with the motivation *motivation*. The result is the new value indicating the *strength* associated with the motivation *motivation*.
   (b) Else, if *exact* undermines the motivation *motivation* (the action *exact* undermines the motivation *motivation* if there is a tuple *conmot* belonging to the set *cons* associated with *exact* whose value *name* is the equal to the value *name* associated with the motivation *motivation*), add the value indicating the degree to which *exact* undermines the motivation *motivation* (i.e. add the value *strength* associated with the tuple *conmot*) from the value *strength* associated with the motivation *motivation*. The result is the new value indicating the *strength* associated with the motivation *motivation*.
   (c) Else, do nothing (the value indicating the *strength* associated with the motivation *motivation* remains the same as the action *exact* neither supports nor undermines the motivation *motivation*).

This algorithm however, only takes into account the effect that executing the action has on the motivations, it does not take into account how unforeseen changes brought about by other agents or physical processes, or how changes made to the agent's partial plan might change the strength associated with each motivation. In addition, the tuples assigned to the fields *pros* and *cons* are an estimate of the degree to which executing the action supports/undermines the agent's motivations and their value assumes that the outcome of execution is as expected. Should execution go awry, the agent's motivations may be supported/undermined to a different degree. The "Update motivations" component has not been implemented - currently, once an action has been executed, a user keys in the change in strength associated with each motivation.

7.5 Generate/Update Goals

In the previous section we saw that changes to the environment (which occur both as a consequence of the agent executing an action and as a consequence of other agents or physical processes acting in that environment) and changes to the agent's partial plan cause a change in the strength associated with the agent's motivations. The "Update motivations" component is responsible for determining the new values indicating the

strength associated with each motivation to reflect the changes that have occurred both to the environment and to the agent's plan. Such changes in the strength associated with each motivation, together with the newly updated partial plan (updated by the "Execute action" and "Recover" components to reflect the changes that have occurred within the environment) may lead to the generation of new goals. The "Generate/Update Goals" component is responsible for generating new goals in response to the newly updated set of motivations and to information which can be inferred from the newly updated partial plan. It is assumed that goals are generated in response to the following information (see chapter 2, section 2.2.4).

1. The newly updated set of motivations.
2. The current state of the environment (this is represented within the partial plan).
3. Future states of the environment (these may be predicted using information contained within the partial plan - i.e. we can predict that certain actions will be executed and that certain goals will be achieved).
4. The future strength of the agent's motivations (this may be estimated using the values *pros* and *cons* associated with each action belonging within the partial plan).

This component has not been implemented - currently, a user is able to key in new goals which are added to the set of open conditions belonging within the partial plan.

In addition to generating new goals, the "Generate/Update goals" component is responsible for updating existing goals (these are goals previously generated by the "Generate/Update goals" component). Changes in the agent's motivations and environment may mean that the value indicating the *importance* and/or the *deadline* associated with a previously generated goal may change or that a previously generated goal should be deleted as it is no longer necessary (for example, if a meeting is cancelled, the goal to attend that meeting should be removed from the partial plan). Such changes to goals that have been previously generated have repercussions within the partial plan.

1. If the value indicating the *importance* associated with a previously generated goal is changed, the importance associated with each action and subgoal/precondition that contributes towards that goal changes and must therefore be recalculated.
2. If the deadline associated with a previously generated goal is changed, the deadline associated with each action and subgoal/precondition that contributes towards that goal changes (the deadlines associated with other actions may also be affected

depending upon the ordering constraints within the partial plan) and must therefore be recalculated (the "Estimate deadlines" procedure must be called and, if this returns the value :fail, the "Edit the partial plan" procedure must also be called).

3. If a previously generated goal is deleted/removed, all actions/constraints that contribute towards that goal must be removed (the partial plan editing routine must be called).

A facility which enables a user to update the importance/deadline associated with previously generated goals or to delete previously generated goals has not been implemented although implementation is trivial as most of the routines necessary to support this requirement have already been implemented.

7.6 Summary

In this chapter we described in detail each of the components that are responsible for updating the partial plan and the agent's motivations to reflect the outcome of execution. In particular we described how a description of the time and state of the environment following execution is supplied by a user - this enables us to model changes made to the environment as a consequence not only of the planning/execution agent executing an action, but also as a consequence of the activites of other agents/physical processes (these are unforeseen changes). This facility is powerful as it provides a user with the opportunity to supply new information to the planning/execution architecture once execution has taken place. For example, in the truck world domain, in chapter 5, section 5.4.2 we showed how new information concerning the location of the *parcel* was supplied by a user following execution of the action *drive-truck(truck city3 city4)* (see table 5.11). This information was supplied because the truck-driver was requested to achieve the new goal *at(parcel city5)* shown in table 5.12.

Using the truck world domain, we then demonstrated how a partial plan is updated to reflect the outcome of execution ("Execute action"), and how the "Recover" component repairs a partial plan if the world changes in such away as to adversely affect that partial plan. Finally we discussed how the components "Update motivations" and "Generate/ update goals", which are beyond the scope of this thesis, might be implemented.

Chapter 8

Conclusions

8.1 Introduction

In this thesis we described a planning/execution system to be used onboard an agent to enable it to plan and act in order to satisfy its aims or goals. The system was designed to address four requirements: to model the context of the agent; to plan and act in real or simulated time; to interleave planning and execution; to cope with unpredicted changes made to the environment. In this chapter we first discuss how the system may be evaluated as well as its limitations. We then discuss the contributions of the work to the area of AI planning, and finally describe possibilities for future research.

8.2 Evaluation

8.2.1 Introduction

One of the main problems with the planning/execution architecture described in this thesis is the question of how to evaluate its performance. Planning and execution are interleaved, and the world may change in unexpected ways due to execution failure as well as the activities of other agents or physical processes. In such a scenario, how is it possible to determine whether the agent is acting efficiently? What is an optimal plan? When faced with an unpredictable environment, new goals and time constraints, an agent can only aim to act in the best way it can (i.e. satisficing) - whatever this may mean. Context may play a role in determining performance. For example, a good plan for a lazy agent may be one that enables the agent to do as little as possible, whereas a good plan for an agent working with resource constraints may be one that maximises (or conserves) resources. One method of evaluating the performance of the overall system is to monitor how many of the goals generated by the "Generate/update goals" component are

achieved by their deadlines (this involves maintaining a list of all goals that have been achieved) over a certain period of time. Achieving a large number of important goals is good, a small number of less important goals is bad. Another criteria for evaluating performance is to examine how busy the agent is over a period of time - i.e. by determining how many actions the agent performs as well as how much time is spent acting. As we mentioned above, context plays a role - the agent might like being busy, in which case, spending as much time acting as possible is good and not acting is bad.

Because the two components "Update motivations" and "Generate/update goals" components have not been implemented however, it is not possible to accurately evaluate the performance of the entire system. Instead, we focus upon the various ways in which the components of the planning/execution architecture which have been implemented (these are the components with solid rectangular boxes in chapter 2, section 2.2.1, figure 2.1) may be evaluated.

8.2.2 Efficiency

The plan generation process described in this thesis is based on SNLP, a partial order, causal link planner which uses STRIPS representations. SNLP has been proved to generate plans that are both sound, complete and systematic. In this thesis we described how the SNLP algorithm and the STRIPS representation have been extended to reason with goals that have deadlines and actions with duration. In addition, the problem solving context has been represented by modelling the motivations of the agent. Such extensions, (i.e. estimating the deadlines of actions, editing and evaluating partial plans) increase the computational load. Unlike SNLP and many other planners (e.g. Graphplan) which, when presented with a goal, create a complete plan to achieve that goal, the planning/ execution framework described in this thesis is continually presented with goals to achieve and must therefore interleave planning and execution, which means that planning is an ongoing activity and is never complete. The framework plans to achieve goals until it finds an action which can be executed, whereupon it executes that action, and continues planning with the updated world model. The amount of planning that takes place prior to execution may therefore be less than the amount of planning required to generate a complete plan. However, the plan generation process "Plan to achieve goal" will not be fast as it incorporates extended versions of algorithms used by SNLP.

8.2.3 Rationality

In this section we discuss the rationality of the planning/execution system. One of the aims of the work presented in this thesis was to design a planning/execution system to enable an agent to satisfy some of the requirements described in chapter 1, section 1.5 - in particular we designed the system to take into account the context of the agent by modelling the motivations of that agent. In chapter 2, section 2.2.3 and chapter 3, section 3.2.1, we discussed how motivations change in strength in response to changes in the environment as well as to changes in the agent's plan. Changes to the motivations and to the plan are internal to the agent, which means that the agent's internal state changes while it is both planning and acting. Changes in the agent's internal state affect the choices the agent makes - for example, whether to plan to achieve a goal or whether to execute an action; which goal to achieve; which action to execute; which partial plan to choose for subsequent refinement. When making such choices, the agent is able to reflectively evaluate each choice in order to decide upon its subsequent behaviour. We believe that the system's ability to reflectively evaluate each choice is one of the main contributions of this thesis.

However, it is important that the system behaves rationally when deciding upon its behaviour. We define a rational agent as being an agent that is capable of goal-directed decision making and whose behaviour is robust with respect to changes within its internal state. This means that the agent should be able to make sensible decisions about achieving its goals regardless of what changes occur to its internal state. The behaviour of the planning/execution system can therefore be said to be rational if it is robust with respect to changes in the agent's motivations and plans. In the following paragraphs we demonstrate how the planning/execution system behaves with regard to differences in the motivations and plans.

In chapter 4, section 4.6.1 we demonstrated how the "Select goal or action" component decided whether to achieve the goal *at(package city5)* (see chapter 4, section 4.4.2, table 4.3) or whether to execute the action *drive-truck(truck city4 city5)*. Chapter 4, section 4.6.1, table 4.9 shows the values assigned to the fields *importance*, *effort* and *deadline* associated with the goal *at(parcel city5)* and the action *drive-truck(truck city4 city5)*. If, at this point, the agent's motivations had been different, the value indicating the importance associated with the goal *at(package city5)* might have been lower (in chapter 2, section 2.2.4 we described how the agent's motivations directly determine the impor-

tance associated with goals). In this case, the "Select goal or action" component would have chosen to execute the action *drive-truck(truck city4 city5)* instead of achieving the goal *at(parcel city5)*.

In chapter 6, section 6.4.5 we demonstrated how the motivations of the truck-driver (shown in chapter 4, section 4.4.4, table 4.5) influenced the truck-driver to select the plan illustrated in chapter 5, section 5.4.1, table 5.9, to achieve the goal *at(package city5)* which involved the truck-driver driving from *city1* to *city5*, via *city3* and *city4*. If the agent's plan had been different, for example if the deadline associated with the goal *at(package city5)* had been 9 instead of 20, the "Plan to achieve goal" component would have generated a plan in which the truck-driver drove from *city1* to *city5*, via *city2*. This is because there would have been insufficient time available for the truck-driver to achieve the goal by using either of the other two routes (via *city2* and *city4*, or via *city3* and *city4*).

These examples indicate that the planning/execution system is capable of behaving rationally as it makes sensible choices in response to differences in its internal state. We believe that further experimentation will corroborate this view.

In conclusion, we discuss why the development of a rational agent might be of interest to the community in terms of what kind of tasks may require such an agent. The planning/execution architecture described in this thesis generates goals, and aims to achieve as many of those goals as possible within given time constraints taking into account unpredictable changes occurring within its environment. The system cannot guarantee to achieve all of its goals however, making it not particularly suitable for performing safety-critical tasks. In addition, the system cannot guarantee to achieve goals in an optimal manner - at most, it tries to achieve its goals in the best way possible, depending upon its motivations. It is envisaged that this system would be suitable for performing tasks that are tedious for human operators but which require non-trivial problem-solving and goal-directed decision making, for example, tasks in the warehouse domain described in chapter 1, or tasks involving logistics such as an extended version of the truck world domain described in chapter 4.

8.2.4 Generality

The planning/execution architecture described in this thesis is domain-independent and can therefore be applied to solving problems in various domains. In addition, the architecture is agent-dependent as long as the agent satisfies the requirements described in

chapter 1, section 1.5. In chapter 4, section 4.4, we introduced the truck world domain in which a truck-driver must collect packages and parcels from one city and deliver them to another city. The implemented components of the planning/execution architecture have also been tested using an extended version of the blocks world domain. Although the system has only been tested in two domains, it is our belief that it will be effective in many other domains.

8.3 Limitations

8.3.1 A situated agent

In chapter 1, section 1.5 we described the properties required of an agent for which the planning/execution architecture presented in this thesis was designed. The most controversial of these requirements is the assumption that the agent has perfect sensors (i.e. they deliver data which is accurate) which enable the agent to construct a sufficient, accurate, symbolic representation of the environment (by sufficient we mean that the sensors must be able to detect all aspects of the environment that are relevant to the agent's choice of action). [Brooks 86], [Brooks 91] has argued persuasively that it is difficult, if not impossible to construct an accurate model of the environment using sensor data and that to construct and continually update such a model requires intensive computation. Work by [Aylett et al 97] demonstrated that it is possible to combine symbolic reasoning techniques such as planning, with a behavour-based or reactive execution control system, where the resulting agent requires only an abstract symbolic representation of the environment (consisting only of those features that are either static, such as walls or doors, or that are moved as a consequence of the planner deciding that they should be moved). The execution agent, using a behavioural control architecture, is able to deal with changing features - i.e. it is able to avoid obstacles without requiring a symbolic representation of the location of those obstacles. We argue that such an approach could be used to implement an agent with the properties stated in chapter 1 which means that only an abstract model of the environment is required which could be user-supplied as part of the domain description process.

8.3.2 Why use SNLP?

Using SNLP as a basis for the "Plan to achieve goal" algorithm is a severe limitation as SNLP is unable to deal with complex planning problems. Nonlinear causal link planners

such as SNLP and UCPOP are outperformed by more recently developed planners such as Graphplan planners and SAT-based planning systems. In this section we discuss why we chose to extend SNLP when implementing the "Plan to achieve goal" component.

SNLP is well-known and understood by the planning community and has been proved to be sound, complete and systematic (systematicity is the property that the same plan, or partial plan, is never examined more than once). The SNLP algorithm has formed the basis of UCPOP ([Penberthy & Weld 92], [Weld 94]) and many other planners (such as Cassandra [Pryor & Collins 96], [Collins & Pryor 92], Buridan [Kushmerick et al 95], CNLP [Peot & Smith 92], C-Buridan [Draper et al 94], etc.). The code (SNLP is implemented in Allegro Common Lisp) was freely available at the time implementation of the work presented in this thesis commenced.

One of the main advantages of nonlinear causal link planners is their insensitivity to irrelevant information in the initial state (due to their backward-chaining regression search) [Weld 99] which makes them well-suited to domains which change unpredictably. In addition, they maintain a record of which actions achieve each goal or subgoal (persistence constraints) making it relatively easy to determine whether such goals or subgoals are undone or clobbered if execution goes awry. Although planning technology has matured with the development of Graphplan planners and SAT-based planners, such technology is not easy to adapt to cope with interleaving planning and execution in unpredictable environments, especially when goals have deadlines. Although such planners are able to generate plans quickly, once execution commences, if it does not result in the intended outcome, planning would have to recommence from scratch. Graphplan planners would have to build a new planning graph from the new initial state that arises following execution. SAT-based planners would also have to start planning from scratch. The advantage of POCL (partial order causal link) planners is that if execution does not result in the expected outcome, the partial plan is relatively easy to update and repair - it does not necessarily need to be thrown away - which means planning does not have to restart from scratch.

It is also easier to extend POCL planners to enable to reason about time. Unlike the STRIPS representation (which uses a discrete model of time in which all actions are assumed to be instantaneous), in the POCL approach to planning, actions can be of arbitrary duration as long as the conditions under which actions interfere are well-defined [Smith et al 2000]. DEVISER [Vere 83] was the first planner to exploit this by allowing

actions of arbitrary duration and goals and actions to be restricted to user-specified time windows. Many other systems (TRAINS-95 [Ferguson et al 96], Zeno [Penberthy & Weld 94], IxTeT [Ghallab & Laruelle 94], HSTS [Muscettola 94]) have combined ideas from POCL planning with an interval representation for actions and propositions (first introduced by [Allen 84]) using constraint-satisfaction techniques to manage the relationship between intervals. [Smith et al 00] describe this approach as the Constraint-Based Interval (CBI) approach.

In contrast, it is much harder to adapt Graphplan planners to reason with continuous time [Smith et al 00] (see TGP [Smith & Weld 99]) while SAT-based planners can only cope with discrete time with difficulty.

8.4 Contributions

8.4.1 A prototype rational system

A prototype domain-independent and agent-independent planning/execution architecture (with the exception of the "Update motivations" and "Generate/update goals" components) has been implemented using Allegro Common Lisp which extends the partial order causal link planning algorithm SNLP to meet the four objectives described in chapter 1, section 1.6. The truck world domain examples shown in chapters 4, 5, 6 and 7 show the partial plans output by the components of this architecture at various stages of planning and execution. These examples demonstrate the performance of each of the implemented components of the planning/execution architecture and, as discussed in section 8.2.3 indicate that the behaviour of the planning/execution architecture is rational. In the following sections we demonstrate how the system has been designed to meet the four objectives described in chapter 1, section 1.6.

Context

The context of the agent (captured partly by modelling the motivations of that agent) plays an important role in enabling the agent to generate goals (context constrains which goals are generated), to prioritise amongst actions and goals as well as to select the best partial plan in which such goals are achieved. The planning/execution architecture presented in this thesis takes the agent's context into account in the following ways.

Firstly, context is partially captured by modelling the agent's motivations (which represent the agent's desires or preferences). These play an important part in the archi-

tecture as they capture part of the dynamic which exists between an agent and its environment (see chapter 1, section 1.5.2, Embodiment) - motivations directly influence an agent to act in order to satisfy its aims and, whilst acting to satisfy its aims the agent changes its environment which (together with other changes brought about by other agents or physical processes) directly affects the agent's motivations.

Motivations directly influence an agent to act as they affect which goals are generated, enable newly generated goals (and therefore actions) to be assigned a value indicating their priority (i.e. their importance) and enable partial plans to be evaluated/ranked. In order to support these requirements, the planning/execution architecture presented in this thesis contains a component responsible for both creating new and updating existing goals ("Generate/update goals") which takes as input the agent's motivations, its model of the current state of the environment and its current partial plan (which contains information that enables the agent to predict future states of the environment). This component is an important part of the architecture but is not yet implemented. Goals created by this component have an associated value indicating their importance or priority as well as a deadline by which they must be achieved. The "Plan to achieve goal" component (this is described in chapters 5 and 6) is fully implemented and supports the requirement of capturing the agent's context in two ways. The "Achieve goal" component (see chapter 5, sections 5.2.5, 5.2.7 and 5.2.8) assigns and propagates values indicating the importance associated with actions and their preconditions (the step addition and simple establishment procedures estimate the value indicating the importance of actions and their preconditions). This component also estimates the degree to which actions support or undermine the agent's motivations. Finally, the "Evaluate partial plans" process (see chapter 6, section 6.4) evaluates and ranks each partial plan by calculating the degree to which a plan supports or undermines the agent's motivations.

When acting to satisfy its aims or goals, the agent makes changes to its environment (unpredicted changes are also brought about by other agents and physical processes). Such changes to the environment directly affect the agent's motivations. The planning/execution architecture contains a component responsible for updating the agent's motivations once that agent has executed some action. This component, "Update motivations" is described in chapter 7, section 7.4 and is not currently implemented.

To summarise, the planning/execution architecture described in this thesis satisfies the objective of modelling the context of the agent.

Planning and acting in real or simulated time

Time passes while the agent executes actions. Goals that are created by the "Generate/ update goals" component have an associated deadline by which they must be achieved, and duration indicating how long they must remain true. It is assumed that deadlines associated with goals are hard (i.e. if the goal cannot be achieved by its deadline, the agent has effectively failed to achieve that goal). Actions also have duration. It is important that the agent ensures that it can achieve goals by their deadline, if possible. If there is insufficient time available to achieve all of the goals in the partial plan by their deadlines, it is necessary to create extra time by abandoning the achievement of one or more goals (i.e. by editing the partial plan to remove such goals).

In this section we describe the ways in which this requirement is supported by the planning/execution architecture. Firstly the "Achieve goal" (described in chapter 5, section 5.2) process both estimates how long it takes to execute each action (i.e. it assigns a value indicating the duration of each action) and keeps track of which goals each action contributes towards (to facilitate plan editing). Secondly, the "Estimate deadlines" process (described in chapter 6, section 6.2) estimates the deadlines associated with actions and subgoals belonging within each partial plan. If the "Estimate deadlines" process returns the value :fail, it cannot assign deadlines to each action because there is insufficient time available to achieve all goals. In this case, the "Edit the partial plan" process (described in chapter 6, section 6.3) edits the partial plan by removing one or more goals and their associated actions and constraints - a consequence of plan editing is to create more time for the agent to achieve its remaining goal. The planning/execution architecture described in this thesis therefore satisfies the objective of enabling the agent to plan and act in real or simulated time.

Interleaving planning and execution

Planning and execution are ongoing activities - because the agent is capable of continually generating new goals, planning is never complete which means that planning and execution must be interleaved. To support this requirement, the "Select goal or action" component of chapter 2, section 2.2.1, figure 2.1, decides, once each cycle, whether to plan to achieve a goal/subgoal or whether to execute an action. In order to do this goals/ subgoals and actions are evaluated on the basis of values indicating their importance, deadlines and effort (for goals, the value *effort* represents the amount of effort previously

expended by the planner in achieving those goals, while for actions and their associated preconditions, the value *effort* represents the amount of effort previously expended by the "Achieve goal" process in achieving the set of goals to which each action and its preconditions contributes). The "Plan to achieve goal" component is responsible for assigning values indicating the importance, effort and deadline to each action and its preconditions - the "Achieve goal" process assigns values indicating the importance, effort and duration associated with each action and its preconditions, while the "Estimate deadlines" process assigns deadlines to each action and its preconditions (using the values indicating the duration of each action). The planning/execution architecture described in this thesis therefore satisfies the objective of enabling the agent to interleave planning and execution.

Planning and acting in an unpredictable environment

In chapter 1, section 1.6, we described how one of the required properties of the agent is that it should be able to cope with an unpredictable environment. The agent is unable to accurately predict the environment for the following reasons.

1. When the agent executes an action, the actual outcome may differ from the predicted outcome (for example, the warehouse agent may drop a commodity whilst picking it up off the shelf, or the time taken to pick up the commodity may take longer or shorter than anticipated).
2. Other agents act in the environment - because the agent in question has only access to the external state of other agents acting within the environment, the agent is unable to predict the consequences of the activities of other agents. For example, there may be several other agents of differing capabilities acting within the environment, but because the agent does not have knowledge of the capabilities of these agents, it will be unable to predict how they might act and change the environment. In the warehouse domain, another warehouse agent might be occupying the battery recharge point, thereby causing it to be unavailable.
3. Physical processes occurring within the environment may cause changes to that environment (for example, an ice cube will melt if it is exposed to a temperature greater than 0° Celcius) - the agent can only predict the consequences of physical processes if it has knowledge of such processes.

When unexpected changes such as those described above, occur to the agent's envi-

ronment, the agent must be able to adapt. The planning/execution architecture described in this thesis supports this requirement by enabling the agent to first monitor the actual changes that have occurred within the environment, and then to recover, if necessary, by reachieving goals or subgoals that have been accidentally undermined. The "Execute action" component described in chapter 7, section 7.2 is responsible for updating the partial plan to reflect the changes which have occurred within the environment following execution. The "Recover" component is responsible for repairing the partial plan in two ways: if previously achieved goals or subgoals have been undermined following execution, such goals/subgoals are placed in the set of outstanding goals/subgoals so that they can be reachieved; if execution takes longer than predicted, the deadlines of actions/preconditions are reestimated to ensure that there is still sufficient time available to achieve all goals by their deadlines - if there is insufficient time, the partial plan will be edited. The planning/execution architecture described in this thesis therefore satisfies the objective of enabling the agent to cope with an unpredictable environment.

8.4.2 Reflective evaluation

Finally, the main contribution of this thesis is the design and implementation of a planning/execution architecture that enables an agent to deliberate about its choices of behaviour. The architecture allows the agent to deliberate when making the following choices.

1. Deciding whether to plan to achieve a goal or whether to execute an action. The "Select goal or action" component described in chapter 4, section 4.5 enables the agent to choose whether to plan or whether to execute by reasoning about the importance, effort and deadlines associated with actions/goals/subgoals.
2. Deciding which is the best goal to achieve or which is the best action to execute. Again, this facility is provided by the "Select goal or action" component.
3. Deciding which is the best partial plan. The "Evaluate partial plans" component presented in chapter 6, section 6.4 enables the agent to evaluate each partial plan by assessing the degree to which that partial plan supports or undermines the agent's motivations. By modelling the agent's motivations as well as the degree to which each action supports or undermines the motivations, we partially capture the context of that agent.

The agent is therefore provided with the ability to perform reflective evaluation when deciding upon its behaviour.

8.5 Future Work

The research presented in this thesis has presented many areas that merit further investigation. Firstly, in order to effectively evaluate the performance of the planning/execution architecture described in chapter 2, section 2.2.1, figure 2.1 we require an implementation of the "Update motivations" and "Generate/update goals" components. (Currently, a user both updates the strength associated with each motivation and generates/updates goals.) We need to investigate how changes to the agent's environment as well as changes to the agent's plan affect the agent's motivations - in chapter 2, section 2.2.3, chapter 3, section 3.2.1 and chapter 7, section 7.4 we presented an overview of the ways in which such changes might affect the agent's motivations but a deeper analysis is required in order to implement the "Update motivations" component. In addition, we need to further investigate how changes to the agent's motivations, plan and environment might cause new goals to be generated and existing goals to be updated. Again, in chapter 2, section 2.2.4, chapter 3, section 3.2.2 and chapter 7, section 7.5, we presented some ideas as to why or when new goals are generated and existing goals updated. [Norman 97] describes in detail a mechanism for generating goals which could be extended to suit the planning/execution architecture presented in this thesis.

Another area for further research is to implement execution. The work presented in this thesis simulates execution - once an action has been executed, a user supplies a set of predicates which represent the state of the environment following execution. One approach is to interface the planning/execution architecture described in this thesis with a behaviour-based execution control architecture (see [Aylett et al 97], [Barnes et al 97]). This requires an agent to be implemented with sensors and effectors. A simulated environment could then be implemented to show how effective an implemented agent is in achieving its goals and aims within that environment.

8.6 Conclusions

The most significant contribution to the field of AI planning made by the planning/execution architecture described in this thesis is the facility it provides to enable an agent using such a system to perform reflective evaluation when choosing its course of behaviour. This is achieved primarily by using motivations to model the context of an agent capable of planning and execution and by enabling such an agent to reason about time. In addi-

tion, one of the strengths of the planning/execution system is that it is both domain and agent independent. The implementation of the system demonstrates that the ideas and mechanisms proposed are computationally possible and effective, making it a prototype as opposed to a fully functional system capable of planning and acting within the real world.

Bibliography

[Allen 84] Allen, J., "Towards a general theory of action and time", *Artificial Intelligence*, vol. 23, no. 2, pp. 123-154, 1984.

[Ambite et al 97] Ambite, J., Arens, Y., Ashish, N., Knoblock, C. A., Minton, S., Modi, J., Muslea, M., Philpot, A., Shen, W., Tejada, S. & Zhang, W., "The SIMS Manual: Version 2.0", Information Sciences Institute and Department of Computer Science, University of Southern California, 4676 Admiralty Way, Marina del Rey, CA 90292, USA, 1997.

[Ambros-Ingerson 87] Ambros-Ingerson, J., "IPEM: Integrated Planning, Execution, and Monitoring", *Ph.D. Thesis*, University of Essex, 1987.

[Aylett et al 97] Aylett, R. S., Coddington, A. M., Barnes, D. P. & Ghanea-Hercock, R. A., "What does a planner need to know about execution", In *Recent Advances in AI Planning, Proceedings of the Fourth European Conference on Planning (ECP-97)*, eds. Steel, S. & Alami, R., pp. 26-38, Springer, 1997, ISBN 3540639128.

[Bacchus & Kabanza 98] Bacchus, F. & Kabanza, F., "Using Temporal Logics to Express Search Control Knowledge for Planning", submitted to *Artificial Intelligence*, 1998.

[Barnes et al 97] Barnes, D. P., Ghanea-Hercock, R. A., Aylett, R. S. & Coddington, A. M., "Many hands make light work? An investigation into behaviourally controlled co-operant autonomous mobile robots", *Proceedings of the First International Conference on Autonomous Agents*, pp. 413-420, 1997.

[Bates et al 92] Bates, J. B., Loyall, A. B. & Scott Reilly, W., "An Architecture for Action, Emotion, and Social Behaviour", Technical Report CMU-CS-92-144, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, May 1992. Also appearing in *Artificial Social Systems: Fourth European Workshop on Modeling Autonomous Agents in a Multi-Agent World,*

Springer-Verlag, Berlin, 1994.

[Blum & Furst 97] Blum, A. L. & Furst, M. L., "Fast Planning through Planning Graph Analysis", *Artificial Intelligence*, vol. 90, pp. 281-300, 1997.

[Blythe 99] Blythe, J., "Decision-theoretic Planning", *AI Magazine*, vol. 20, no. 2, pp.37-54, 1999.

[Bonet & Geffner 99] Bonet, B. & Geffner, H., "Planning as heuristic search: New results", In *Recent Advances in AI Planning, Proceedings of the Fifth European Conference on Planning (ECP'99)*, eds. Biundo, S. & Fox, M., pp. 360-372, Springer 1999, ISBN 3540678662.

[Boutilier et al 99] Boutilier, C., Dean, T. & Hanks, S., "Decision-Theoretic Planning: Structural Assumptions and Computational Leverage", *Journal of Artificial Intelligence Research*, vol. 11, pp. 1-94, 1999.

[Brooks 86] Brooks, R., "A Robust Layered Control System for a Mobile Robot", *IEEE Journal of Robotics and Automation,* RA-2(1) pp. 14-23, 1986.

[Brooks 91] Brooks, R., "Intelligence without Representation", *Artificial Intelligence*, vol. 47, pp. 139-159, 1991.

[Chapman 87] Chapman, D., "Planning for Conjunctive Goals", *Artificial Intelligence*, vol. 32, pp. 333-337, 1987.

[Collins & Pryor 92] Collins, G. & Pryor, L., "Achieving the functionality of filter conditions in a partial order planner", In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, August, 1992.

[Dean et al 87] Dean, T., Firby, J. & Miller, D., "Hierarchical planning involving deadlines, travel time, and resources", *Computational Intelligence,* vol. 4, no. 4, pp. 381-398, 1987.

[Doherty & Kvarnström 99] Doherty, P. & Kvarnström, J., "TALplanner: An Empirical Investigation of a Temporal Logic-based Forward Chaining Planner", *Proceedings of the Sixth International Workshop in Temporal Representation and Reasoning (TIME'99)*, 1999.

[Drabble & Tate 94], Drabble, B. & Tate, A., "The Use of Optimistic and Pessimistic Resource Profiles to Inform Search in an Activity Based Planner", In *Proceedings of the Second International Conference on AI Planning Systems*,

Chicago, AAAI Press, June 1994.

[Draper et al 94] Draper, D., Hanks, S. & Weld, D., "Probabilistic planning with information gathering and contingent execution", In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, ed. Hammond, K., pp. 31-37, Chicago, AAAI Press, 1994.

[El-Kholy & Richards 96] El-Kholy, A. & Richards, B., "Temporal and Resource Reasoning in Planning: the *parc*PLAN approach", In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, pp. 614-618, 1996.

[Erol et al 94] Erol, K., Nau, D. & Hendler, J., "UMCP: A Sound and Complete Planning Procedure for Hierarchical Task-Network Planning", In *Proceedings of the Second International Conference on AI Planning Systems* (AIPS-94), Chicago, June, 1994.

[Etzioni et al 92] Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N. & Williamson, M., "An approach to planning with incomplete information", In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pp. 115-125, Cambridge, MA, 1992.

[Ferguson et al 96] Ferguson, G., Allen, J. & Miller, B., "TRAINS-95: towards a mixed-initiative planning assistant", In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems (AIPS-96)*, pp. 70-77, 1996.

[Fikes & Nilsson 71] Fikes, R. E. & Nilsson, N., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *Artificial Intelligence*, vol. 5, no. 2, 1971.

[Fikes et al 72] Fikes, R. E., Hart, P. E. & Nilsson, N. J., "Learning and Executing Generalized Robot Plans", *Artificial Intelligence*, vol. 3, no. 4, pp. 251-288, 1972.

[Gat 96] Gat, E., "ESL: A language for supporting robust plan execution in embedded autonomous agents", In *Plan Execution: Problems & Issues*, Papers from the 1996 AAAI Fall Symposium, Technical Report FS-96-01, November 9-11, 1996, Cambridge, Massachusetts, AAAI Press, 1996, ISBN

1577350154.

[Ghallab & Laruelle 94] Ghallab, M. & Laruelle, H., "Representation and control in IxTeT, a temporal planner", In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pp. 61-67, 1994.

[Ghosh et al 92] Ghosh, S., Hendler, J., Kambhampati, S. & Kettler, B., "UM Nonlin Version 1.2.2 User Manual", Parallel Understanding Systems Group, Department of Computer Science, University of Maryland, College Park, MD 20742, 1992.

[Hoffman & Nebel 00] Hoffmann, J. & Nebel, B., "The FF Planning System: Fast Plan Generation through Heuristic Search", submitted to the *Journal of Artificial Intelligence Research*, 2000.

[Hoffmann 00] Hoffmann, J., "A Heuristic for Domain Independent Planning and its use in an enforced Hill-climbing Algorithm", *Technical Report*, Institut für Informatik, 2000.

[Kautz & Selman 99] Kautz, H. & Selman, B., "Blackbox: A new approach to the application of theorem proving to problem solving", In *AIPS98 Workshop on Planning as Combinatorial Search*, pp. 58-60, 1998.

[Knoblock 92] Knoblock, C. A., "An Analysis of ABSTRIPS", In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, pp. 126-135, Morgan Kaufmann, 1992.

[Knoblock 95] Knoblock, Craig A., "Planning, Executing, Sensing and Replanning for Information Gathering", In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

[Knoblock 96] Knoblock, Craig A., "Why Plan Generation and Plan Execution are Inseparable", In *Plan Execution: Problems & Issues*, Papers from the 1996 AAAI Fall Symposium, Technical Report FS-96-01, November 9-11, 1996, Cambridge, Massachusetts, AAAI Press, 1996, ISBN 1577350154.

[Koehler et al 97] Koehler, J., Nebel, B., Hoffmann, J. & Dimopoulos, Y., "Extending Planning Graphs to an ADL Subset", In *Recent Advances in AI Planning, Proceedings of the Fourth European Conference on Planning*, eds. Steel,

S. & Alami, R., pp. 273-285, Springer-Verlag, 1997, ISBN 3540639128.

[Koehler 98] Koehler, J.,"Planning under Resource Constraints", In *Proceedings of the Thirteenth European Conference on Artificial Intelligence*, Brighton, U.K., pp. 489-493, August 23-28, 1998.

[Kunda 90] Kunda, Z., "The Case for Motivated Reasoning", *Psychological Bulletin*, vol. 108, no. 3, pp. 480-498, 1990.

[Kushmerick et al 95] Kushmerick, N., Hanks, S. & Weld, D., "An algorithm for probabilistic planning", *Artificial Intelligence*, vol. 76, pp. 239-286, 1995.

[Laborie & Ghallab 95] Laborie, P. & Ghallab, M., "Planning with Sharable Resource Constraints", In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, vol. 2, pp. 1643-1649, Montreal, Canada, 1995.

[Lever & Richards 94] Lever, J. & Richards, B., "*parc*PLAN: A Planning Architecture with Parallel Actions, Resources and Constraints", In *Proceedings of the 9th International Symposium on Methodologies for Intelligent Systems*, pp. 213-222, 1994.

[Long & Fox 99] Long, D. & Fox, M., "Efficient Implementation of the Plan Graph in STAN", *Journal of Artificial Intelligence Research*, vol. 10, pp. 87-115, 1999.

[Luck 93] Luck, M. M., "Motivated Inductive Discovery", *Ph.D. Thesis*, University of London, 1993.

[McAllester & Rosenblitt 91] McAllester, D. & Rosenblitt, R., "Systematic Nonlinear Planning", In *Proceedings of the Ninth National Conference on Artificial Intelligence, AAAI-91,* vol. 2, pp. 634-639, Anaheim, California, USA, AAAI Press/MIT Press, July 1991.

[Muscettola 94] Muscettola, N., "HSTS: Integrating planning and scheduling", In *Intelligent Scheduling*, eds. Fox, M. & Zweben, M., Morgan Kaufmann, 1994.

[Muscettola et al 98] Muscettola, N., Pandurang Nayak, P., Pell, B. & Williams, B. C., "Remote Agent: To Boldly Go Where No AI System Has Gone Before", *Artifical Intelligence*: 100th volume on the Best of IJCAI 1997, 1998.

[Norman 97] Norman, T. J. F., "Motivation-based direction of planning attention in

agents with goal autonomy", *Ph.D. Thesis*, University of London, 1997.

[Pednault 89] Pednault, E., "ADL: Exploring the middle ground between STRIPS and the situation calculus", In *Proceedings Knowledge Representation Conference*, 1989.

[Pell et al 96a] Pell, Bernard, Chien, Gat, Muscettola, N., Pandurang Nayak, P., Wager, M. D. & Williams, B. C., "An implemented architecture integrating onboard planning, scheduling, execution, diagnosis, monitoring and control for autonomous spacecraft", In *Proceedings of the SPIE conference on Optical Science, Engineering and Instrumentation*, 1996.

[Pell et al 96b] Pell, B., Gat, E., Keesing, R., Muscettola, N. & Smith, B., "Plan Execution for Autonomous Spacecraft", In *Plan Execution: Problems & Issues*, Papers from the 1996 AAAI Fall Symposium, Technical Report FS-96-01, AAAI Press, pp. 109-116, 1996, ISBN 1577350154.

[Penberthy & Weld 92] Penberthy, J. S. & Weld, D., "UCPOP: A sound, complete, partial-order planner for ADL", *Proceedings, 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, pp. 103-14, October 1992.

[Penberthy & Weld 94] Penberthy, J. S. & Weld, D., "Temporal Planning with Continuous Change", *Proceedings of the 12th National Conference on Artificial Intelligence*, vol. 2, pp. 1010-1015, 1994.

[Peot & Smith 92] Peot, M. A. & Smith, D. E., "Conditional nonlinear planning", In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, ed. Hendler, J., pp. 189-197, Morgan Kaufmann, 1992.

[Picard 97] Picard, R., "Affective Computing", MIT Press, 1997, ISBN 0262161702.

[Pryor & Collins 96] Pryor, L. & Collins, G., "Planning for contingencies: a decision-based approach", *Journal of Artificial Intelligence Research*, vol. 4, pp. 287-339, 1996.

[Russell & Norvig 95] Russell, S. & Norvig, P., "Artificial Intelligence: A modern approach", Prentice-Hall, 1995, ISBN 0131038052.

[Sacerdoti 74] Sacerdoti, E. D., "Planning in a hierarchy of abstraction spaces", *Artificial Intelligence*, vol. 5, no. 2, pp. 115-135, 1974.

[Sacerdoti 75] Sacerdoti, E. D., "The Nonlinear Nature of Plans", *Proceedings of the In-*

ternational Joint Conference on Artificial Intelligence, pp. 206-214, 1975.

[Smith et al 00] Smith, D. E., Frank, J. & Jonsson, A. K., "Bridging the Gap between Planning and Scheduling", To appear in *Knowledge Engineering Review*, vol. 15, no. 1, 2000.

[Smith & Weld 99] Smith, D. & Weld, D., "Temporal planning with mutual exclusion reasoning", In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, pp. 326-333, 1999.

[Tate 77] Tate, A., "Generating Project Networks", *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 888-893, 1977.

[Vere 83] Vere, S. A., "Planning in Time: Windows and Durations for Activities and Goals", *Pattern Analysis and Machine Intelligence,* vol. 5, pp.246-267, IEEE 1983.

[Weld 94] Weld, D. S., "An Introduction to Least Commitment Planning", *AI Magazine*, vol. 15, no. 4, pp. 27-61, 1994.

[Weld et al 98] Weld, D. S., Anderson, C. & Smith, D. "Extending graphplan to handle uncertainty and sensing actions", In *Proceedings of the Sixteenth National Conference on AI*, 1998.

[Weld 99] Weld, D. S., "Recent Advances in AI Planning", Technical Report UW-CSE-98-10-01, Department of Computer Science & Engineering, University of Washington, Seattle, WA 98195-2350, USA, To appear in: *AI Magazine*, 1999.

[Wilkins 88] Wilkins, D. E., "Practical Planning: Extending the Classical AI Planning Paradigm", Morgan Kaufmann, ISBN 093461394X, 1988.

[Wooldridge & Jennings 95] Wooldridge, M. & Jennings, N., "Intelligent Agents: Theory and Practice", *Knowledge Engineering Review*, vol. 10, No. 2, pp. 115-152, 1995.