PROGRESSIVE PLAN EXECUTION IN A DYNAMIC WORLD

S. Au N.Parameswaran
Department of Information Engineering,
School of Computer Science and Engineering,
The University of New South Wales, P.O.Box 1, Kensington, N.S.W. 2052 Australia
Fax: 61-2-93851814, Phone: 61-2-93853940
{sherlock,paramesh}@cse.unsw.edu.au

Abstract

In complex dynamic worlds, reactive behaviour alone is inadequate for dealing with the diverse and unpredictable situations which may occur. To overcome this inadequacy, an agent in a dynamic world must exhibit goal directed behaviour, which is acquired through the execution of plans. However, completely specified plans are unsuitable in dynamic worlds for several reasons, and in particular the view that plans may be considered as executable programs is not suitable. We argue that an agent working in such situations must explicitly incorporate resource based abstractions in its plan representation. But, while executing such plans, a plan executor must, at any point during execution, be capable of recognising the current state of the world in order to execute an action in the plan. Since this task is difficult while an abstract action is executed, we suggest that plans be made progressively abstract into the future. We demonstrate that when an agent employs such a strategy, it faces the problem of how to generate its own abstractions in order to keep the plan simple and easy to maintain in a dynamic world. We present a methodology as to how an agent can generate simple abstractions in a small fire world, and we evaluate the performance of the agent in its ability to achieve its goals when the world changes unpredictably.

1 Introduction

Agents which exhibit goal oriented behaviour often need plans, particularly in multiagent dynamic worlds. Classically, plans are viewed as a set of actions which are scheduled for execution in a particular order. However in dynamic environments, classical long plans are not always successfully executed due to unpredictable changes in the world. A change in the world can make plans invalid. When plans become invalid, they must be either dropped or modified. (Modification is also called maintenance.) Dropping plans is the least preferred option in a multiagent world, since the plan structure may form the basis for updating the values of other mental attributes of the agent. However, maintaining a completely specified plan is difficult as this plan tends to become invalid as soon as any resource used in the plan changes its state.

In this paper, we propose a plan structure for agents existing in a dynamic world, incorporating abstractions explicitly in the plan structure. We believe plans must not only be abstract but also be complete in the sense that it should specify a complete solution to the problem. In particular, they must be progressively abstract as opposed to being uniformly abstract. As the world changes in a major way, many objects, particularly the abstract ones, may no longer exist, and an agent must be able to invent suitable new abstractions. We present a methodology for modifying a plan structure in response to changes occurring in the world. Minor changes result in the modification of plans at the action level, whereas major changes in the world result in modifications of the world model, using agent invented abstractions. We present a fire world environment where changes are highly dynamic and we evaluate our proposed plan structure in this domain. Finally, we discuss the results from our preliminary implementation.

2 Plan Structure

Planning involves the construction of a detailed sequential list of actions which when executed will take an agent from an initial state to a final goal state. Classical planning bases its foundations on the assumption that the domain is static, the agent has full knowledge of the world, and the actions of the agent have deterministic effects on the state of the world. However, for agents residing in a large and dynamic world where the above assumptions no longer hold true, classical plans of enumerating all basic actions are unacceptable for one significant reason: the plan becomes too long, and long plans are difficult to reason with and maintain. Therefore, it is necessary to explicitly incorporate abstractions in the plan. We discuss two types of abstractions that an agent may use in its plan representation: one is resource based abstraction and the other one is action based abstraction.

2.1 Proposed Plan Structure

Our proposed plan structure consists of both primitive actions and subgoals that are yet to be planned for. (We refer to achieving subgoals as abstract actions henceforth.) The actions (both primitive and abstract) in the plans, are arranged such that the plan becomes progressively abstract from the current time point to the future. Figure 1 is an example of a progressively abstract plan structure.

In figure 1, the agent is given a goal, G. Solid lines denote planned actions while dotted lines indicate plan segments that are yet to be derived. The plan contains primitive actions that can be scheduled and executed by

the agent, but as we move forward into its future it contains less and less refined subgoals (that is, more and more abstract actions). The agent divides G into G_1, G_2, etc., where G_1 is to be executed before G_2. G_1 is further divided into several levels of subgoals. The leftmost subgoal will have the immediate part more refined while the future segment remains abstract. In this example, a_1, a_2 and a_3 are primitive actions. While the agent is executing these three actions, it will also be deriving the actions (a_4 a_5 a_6 a_7) for subgoal G_{112}. Similarly, while the agent is executing actions (a_4 a_5 a_6 a_7), it will at the same time be deriving actions for subgoal G_{12}. This process continues until the given goal G is accomplished.

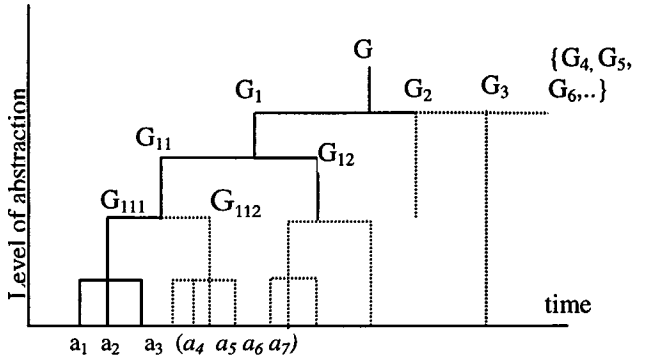

Figure 1. A simple progressively abstract plan.

One merit of the plan structure proposed above is that it subdivides a long plan into numerous short subplans as short subplans are easier to maintain. In addition, the progressively abstract nature of the plan ensures that the agent focuses more on the immediate problems and yet at the same time paying partial attention to the future subgoals (that is, future commitments are attended to continuously). A plan structure such as this is more tolerant to changes in the world than a completely refined plan.

It should be noted that the plan structure discussed is a tentative, but nevertheless, a complete solution for the entire goal. It is complete in the sense that it contains a solution (albeit abstract) for the problem from the start to the end. A plan must be monitored not only at the action level, but also at the subgoals level. In a dynamic environment, changes occurring in the world often create situations in which a planned action could still be executed without failure but leading to the wrong subgoal. This is because the agent would not know that the plan has became a wrong plan because of the changes in the world until the agent has fully executed all the actions. By incorporating abstract objects in the plan, not only can the plan tolerate more changes in the world, but can also help agent to evaluate the progress and provide confirmation about the subgoals during plan execution.

Also note that the progressive abstract plan structure is only a general strategy for plan representation and there can be exceptions to this strategy. For example, sometimes committing to a primitive action in the distant future will simplify the reasoning processes of the agent while selecting a current option.

3 Plan Execution

As the world changes, plans are affected and consequently must be repaired. Agents have to carry out different strategies to maintain the plan for different kinds of changes occurring in the world. For incremental changes which result in action failure, agents may simply have to plan and re-execute the action, but for major changes where abstractions have been destroyed (for example, a wall has been burnt down in a fire world and therefore a room has disappeared), agents will have to invent new abstractions.

3.1 Action Failure

In our plan execution strategy, instead of taking an action and carefully verifying it before execution, we would take the action from the plan and execute it immediately with the assumption that all of its conditions are already satisfied. Because actions are always generated a short while before they are executed, we believe that the availability of the resources in appropriate states is reasonably updated and it is therefore not necessary to re-examine them. The advantage of this approach is that plan execution can be sped up such that the plan is executed before the environment changes again.

Although actions are always executed shortly after they are planned, there is always a possibility that changes affecting the validity of the action can still occur within the period of its planning and execution. In this case, an action will fail. An action could also fail during execution as changes in the world directly affect the resources that the agent is using for the action. In situations such as these, the agent will rescue the plan by generating new subgoals to repair the conditions of the action, achieve these subgoals and then re-execute the previously failed action.

3.2 Subgoal Failure

A change in the world may also affect a plan at the higher (abstract) levels. This can happen when the abstractions employed in the world representation are affected. For example, in a fire world, where a fire burns down a wall existing between two rooms, the two rooms are effectively reduced to one single "large room". As a result, the two rooms disappear from the world, and the "large room" is a

new abstract object introduced into the world and the agent may not be able to reason with this new object. In order to accommodate for this change in its plan, the agent generates a flat plan for the affected subgoal, invents new abstractions over this flat plan, and uses the new abstractions to simplify the plan representation (so that the plan satisfies the "progressive abstraction" requirement). The agent then uses this abstraction to repair the world model.

Consider that, in figure 1, an unexpected change in the world has made the decomposition of G_{11} into G_{111} and G_{112} invalid. This will make the already planned actions for G_{11} meaningless because these actions are related to the old abstractions. (see figure 2).

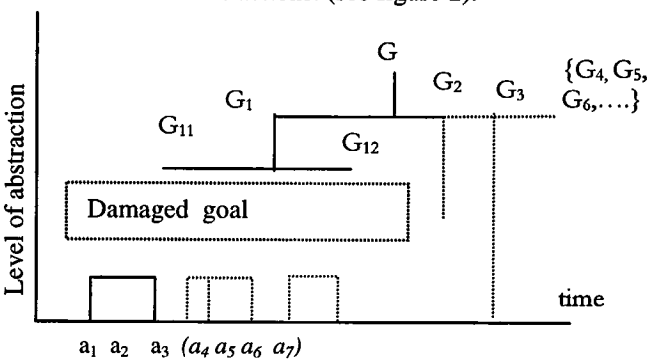

Figure 2. A damaged abstract plan.

Since the abstractions in the world are destroyed, the agent has to invent new abstractions to repair the plan. For example, consider an abstract plan: [start, walk-to(tree), turn-left, walk-to(building), stop]; suppose that the tree is burnt down; then, walk-to(tree) becomes non-executable and this plan cannot be executed directly. New actions have to be derived to replace "walk-to(tree)" with the same action effect and obviously the new actions cannot contain the abstract object "tree". For example, the new plan may look like [start, *move_one_step, move_one_step, move_one_step, move_one_step, move_one_step, move_one_step*, turn-left, walk-to(building), stop]. The agent may simplify this plan by introducing an abstract action called *walk-some-steps*, and the plan can now be represented as [start, *walk-some-steps*, turn-left, walk-to(building), stop]. This plan is short and easy to reason with; however, it is specific to this agent since the definition of *walk-some-steps* may not be known to other agents.

Let us put this strategy into a more formal structure. Consider the example in figure 1 again. The agent first generates a completely refined subplan called P_{11} for G_{11} where $P_{11}=[b_1,b_2,...,b_n]$, where b_i is a primitive action for each i. If this plan is short enough for the agent to maintain, the plan structure for G has been repaired. (Whether a plan is short enough for an particular agent depends on the planning capability of each individual agent.) If it is not (which will typically be the case), then the agent performs the following steps;

1. Divide the plan P_{11} into several segments. For example, $p1=[b_1,b_2]$, $p_2=[b_3,b_4,b_5]$,etc. Note that this divides the plan P_{11} into several subplans. For example, the new representation for P_{11} is $P_{11}=[p_1,p_2,..]$
2. Define a decomposition operator :

   (operator: name *new-decomp-operator1*
   goal G_{11}
   subgoal-list $[p_1,p_2,..,p_k]$ /* *each* p_i *represents a subplan* */)
3. If the newly derived P_{11} is not short enough, repeat steps 1 and 2 above on this version of P_{11}. This may result in the definition of additional new decomposition operators.
4. Now, the original sequence of actions for G_{11} (that is the original version of P_{11}) is given a simpler representation using the subplans defined in steps 1,2, and 3.

When this plan is finally executed, the newly generated decomposition operators will be used by the agent to decompose a goal into subgoals and then refine them to actions. Continuing with the example in figure 1, G_{11} is refined first by a plan P_{11} of nine actions: $P_{11} = [a_1,...,a_9]$. Since we consider this to be too long, we subdivide this plan into three subplans $p_1 = [a_1,a_2,a_3]$, $p_2 = [a_4,a_5,a_6]$, and $p_3 = [a_7,a_8,a_9]$, where for example p_1 achieves a new (hypothetical) subgoal G_{111}. This allows us to define a new decomposition operator:

(operator: name *op1*
goal G_{11}
subgoal-list $[G_1,G_2,G_3]$ /* each Gi represents a self generated subgoal, and Gi is achieved by the execution of plan pi */)

Note that now the plan P_{11} can be rewritten as $P_{11}=[p_1,p_2,p_3]$. If this plan is short enough we stop here; otherwise, we simplify P_{11} further by defining more abstract decomposition operators and subplan segments, until we arrive at a representation for P_{11} which is short enough for the agent to reason with. We then use the resources in the plan segment to update the world model (as discussed in the following section).

4 Fire World

The fire world that we have considered in this work consists of a large number of objects (in the order of hundreds) and a large number of agents (in the order of tens). Objects in the fire world include walls, buildings,

furniture, open areas and LPG gas tanks. An example of such a fire world is shown (partially) in figure 3. In a world such as this, no agent can have full knowledge of the whole world. This world contains all the significant features of a dynamic environment and thus serves as a suitable domain for our plan execution agent.

4.1 A Simple Fire World

In the fire world, humans and animals are modelled as heterogeneous agents. While animals run away from fire instinctively, fire fighters can tackle and extinguish fire. An agent responds to fire at different levels. At a lower level (the physical level), the agent burns like any object, such as a chair. At a higher level, the agent reacts to fire by quickly performing actions, generating goals and achieving goals through planning and plan execution.

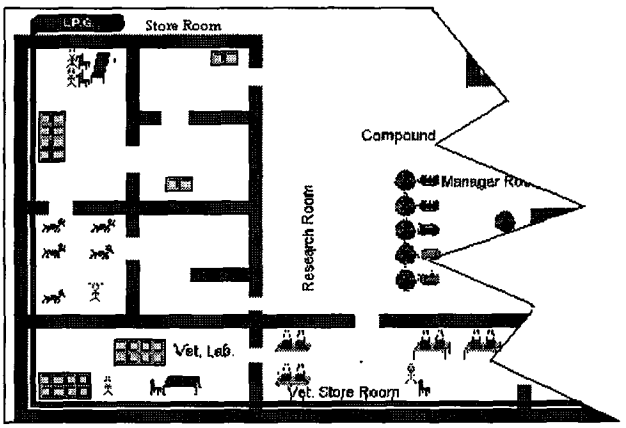

Figure 3. A fire world.

Our world is different from others (like the air combat world of Tambe [Tambe 97] and Robocup [ICJAI 95]) in that problems posed to the agents and the changes in the environment are not only caused by the actions of other agents but also by the changes the objects themselves undergo in the world, caused by the fire. The main aim in this domain is for the agents to co-operate in order to achieve some goals (both common and individual).

4.2 Plan Execution

Consider a subworld (figure 4), where the agent is required to perform a subactivity of reaching the exit starting from the entrance (subgoal G_1). To make the problem interesting, we have placed several inflammable objects (shelves Sh) in the world.

This subworld has been hierarchically divided into several regions and subregions. There are two rooms, the store room and the research room and they are subdivided into eight smaller rooms (with entrances and exits) marked r1,.., r8. These rooms are further subdivided into tiles marked 1,...,16. The sections are grouped to form four regions R1,...,R4. Figure 5 shows a hierarchical

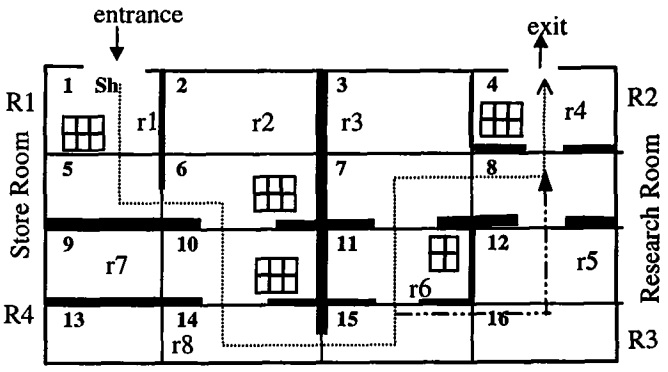

Figure 4. Map of store room and research room.

representation of the store room where each node refers to an object (also referred to as resource).

In order to solve the problem, the agent generates a (progressively abstract) plan P_0=[1,5,r2,R4,RR] and starts executing it. The plan P_0 is to be read as: plan P_0=[move-to(1), move-to(5), move-to(r2), move-to(R4), move-to(RR)]. Plan execution involves attending to action failures and repairing plans at the higher levels of abstraction whenever unpredictable changes take place in the world (as discussed in Section 3). In the following, we only consider subgoal failures.

Subgoal Failure Consider a fire that has burnt down the walls between tiles 6 and 10, and tiles 10 and 14. This change affects the plan at subgoal level as the fire has affected two major abstractions R1 and R4. This has produced several primitive object (tiles) which the agent does not yet know how to group. The subworld at this point looks like the one shown in figure 6.

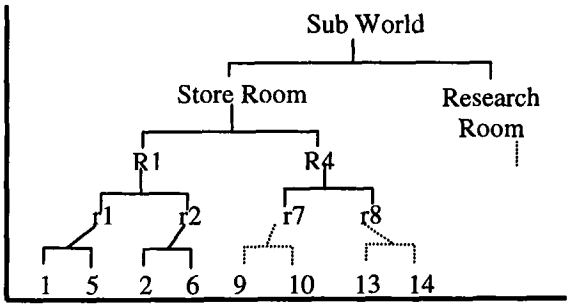

Figure 5. Representation store room and research room.

In order to repair this world, the agent begins with a plan. Since the abstraction over the tiles 6,10, and 14 are lost, the agent derives a completely refined plan [6,10,14] for this region and inserts it in the plan P_0 to produce a new version P_1 where P_1=[1,5,6,10,14,RR]. However, let us suppose this plan looks too long for the agent, and the agent would like to simplify it. As a result, P_1 is simplified into P_2. P_2= [1,5,6,rp',RR]. This plan now is both short and progressively abstract and allows us to define two new (abstract) resources rp' and rp based on the

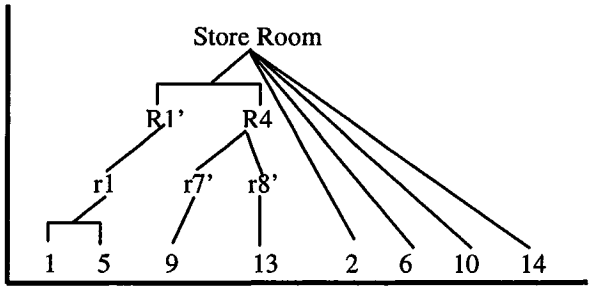

Figure 6. World after fire has burnt part of the walls.

newly introduced subplan segments (See figure 7). These new abstractions are incorporated into the world model as shown in figure 8. Notice that the original abstractions are *spatial abstractions* while the newly created ones are agent *action based.* There are several interesting differences between *action based* abstractions and the *spatial abstractions* originally supplied by the designer. For example, *action based* abstractions are more convenient for plan generation by a planning agent.

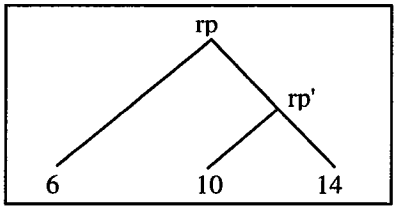

Figure 7. New resources depend on rp and rp'.

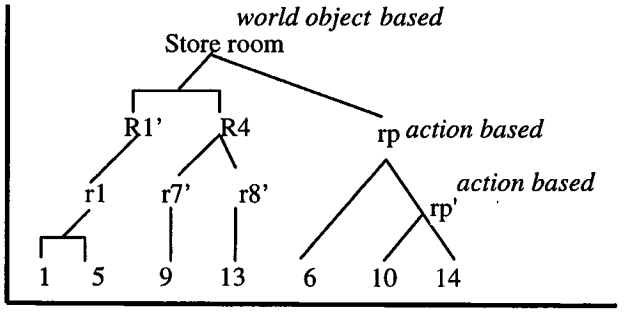

Figure 8. Incorporating new resources in the world.

4.3 Implementation

We implemented the agent and the fire world, using our multiagent production system language called MPS (Multiagent Production System) which we developed for multiagent applications [Au 97]. An MPS program is similar to an OPS5 program and has the capability of communicating with other MPS programs and sharing resources. Often a single agent is implemented by several MPS programs working cooperatively and concurrently together. The agent primarily consists of two concurrent modules (units) called the deliberative reactive unit (DRU), and the physical reactive unit (PRU). We use MPS in our implementation because MPS provides a natural programming paradigm for building communicating reactive modules.

PRU will carry out the actions assigned by the plan and at the same time react to any local situations which are harmful to the agent, such as hot tiles.

DRU defines the mental behaviour and mostly carries out the planning, plan execution, and plan maintenance activities. It is the core component of the agent and the generator of the progressively abstract plans. It is also reactive in the sense that it generates goals and plans in response to changes in world situation. When the agent notices that there is a fire in the campus, it generates a progressively abstract plan and executes it. In the case where the plan fails, the agent creates new abstractions based on its actions and saves the plan by modifying any necessary section of the plan without regenerating the whole plan from the start again.

The world model simulator unit (WSU) is another MPS program that maintains the world by propagating fire in the world, and simulating the actions sent by the DRU and PRU in the world. During each simulation cycle, temperature of each primitive object (for example, tiles 1,5,2,6...in Region 1) is computed, and then propagated upwards towards the root in the hierarchical model world using heuristics, and the state of each object (node in figure 5) is then set as normal, hot, burning or ash. Thus, it is often the case that while lower level nodes are hot or burning, higher level nodes may still be normal. (This is the reason why our progressively abstract plan structure appears more tolerant than a flat plan.)

4.4 Evaluation

In order to evaluate the effectiveness of the plan structure and the plan maintenance scheme, we carried out several experiments under different fire situations. The goal of the agent in our experiment is to get to the exit whenever there is a fire. Given the goal the agent generates a plan similar to the one shown in figure 4 and executes it to reach the exit. Notice that the agent can take one of the two paths to get through region R3.

Along the exit path of the agent there are shelves which can catch fire. The agent tries to get to the exit without being burnt. Effort expended by the agent is measured by means of the number of rules fired and the number of working memory elements (WME) made in the DRU during execution. The PRU module implements the physical reactive behaviour of the agent, and as such does not contribute directly towards deriving any efficient plan by the DRU; rather, the behaviour of PRU had made planning more difficult for the DRU.

Fires of varying intensities were created in the world. A small fire refers to a single occurrence of fire in

the world. A medium fire refers to two to three occurrences of fire in the world. A large fire refers to five or more simultaneous occurrences of fire in the world . Furthermore, we only consider those occurrences of fire which directly affect the plan of the agent. We ran our experiments using three different strategies: progressively abstract plan, the flat plan and the uniformly abstract plan.

When the agent employs a uniformly abstract plan, we found that whenever it encounters fire, it keeps bouncing from tile to tile but doesn't actually go anywhere. This is because the agent takes a long time to transform an abstract action in the plan to a list of executable actions. By the time the list of actions are derived, the fire has already spread to where the agent is standing and forces the agent to jump. Therefore, the agent has to plan again and so on. This loop keeps repeating until the agent is finally burnt to death. This shows that an agent cannot afford to have totally abstract plans.

Long flat plans and progressively abstract plans perform better than uniformly abstract plans. Figure 9 shows the performance of the agent when it employed the progressively abstract plan and the flat plan structure respectively. The world simulator unit (WSU) consists of 58 MPS rules, while the agent contained 80 rules of which the DRU had 42 rules and the PRU had 38 rules. In each module, the number of rules fired and the number of WME's made give an indication of the total effort put in by that module towards solving the given problem. We ran the experiment over several sets of data and in a variety situations. The results of a typical run are summarised in figure 9.

We note that when the fire is small, both the structures of the plan seem to do equally well. In both the cases, the agent manages to start from the entrance and arrive at the exit safely. However, the agent, when employing the flat plan structure seems to spend more effort in its attempt to reach the exit, than when employing the progressively abstract plan structure.

For medium fires, the flat plan structure demand

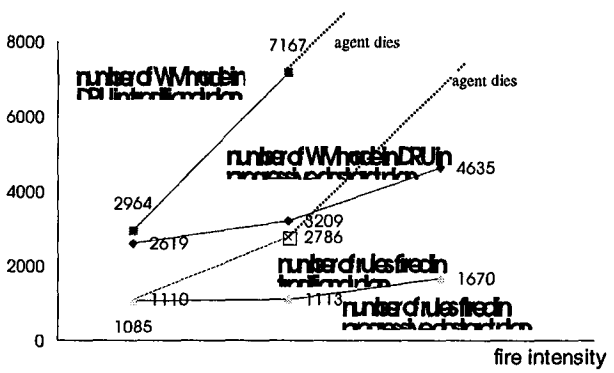

Figure 9. How progressive abstract plan structures compare with the traditional long plan methodology.

about twice as much processing effort than what the progressively abstract structure seems to demand. In the flat plan case, whenever the agent had to modify its plan, the plan always turned out to be much longer. Since the plan specified all the participating lower level resources explicitly, the plan had to be modified every time a resource was affected by the fire. Thus, more work had to be done and it took more time to get to the exit. Meanwhile the fire continued to spread making the plan invalid again. As a result, the number of times the flat plan failed is much higher than the progressively abstract plan.

The increase in the resource usage for the traditional planning agent compared to the progressively abstract planning agent in the DRU is close to 250%. In situations where large areas of the world is on fire, the agent can only save itself by using the progressively abstract plan structure. The flat plan structure took up too much time for planning/plan modification that the fire spread everywhere making it impossible for the agent to execute any plan: the DRU module and the PRU module were excessively busy responding to world changes.

Commenting on the behaviour of the agent, when the agent adopted the flat plan structure, it appeared to be over cautious with regard to its future. The progressively abstract plan structure produces an agent behaviour that appeared "less fuzzy" about future, unless it turned out to be a "major" one. In our implementation, after experimenting with several sets of data, we found that the progressively abstract planning methodology makes use of the alternative path (via tile 16 in figure 4) four times more often than the flat plan approach. Agents executing long flat plans do not consider the alternative path unless tile 11 itself has a large fire thus putting themselves in greater risk. Thus, progressively abstract structure seems to produce a behaviour in which the agent appears more systematic and acreful about its future.

Progressively Abstract Plans as a Complex Mental Attribute Apart from guiding the agent to achieving the goal, the abstractions in a plan can also be used as a basis to judge the severity of the fire in the world. Whether a fire is major or minor fire is estimated relative to the changes it causes to the plan structure. When the fire is large enough to affect the abstraction used by the agent, it may be regarded as major. A fire may be considered minor as long as it only affects the local actions, no matter how large the fire otherwise is.

5 Related Work and Conclusion

Most of the research done on plan execution and monitoring relates to viewing planning and plan execution as functions of stand alone software systems as opposed to

the activities of an agent intending to achieve a goal in a dynamic world. According to Pollack [Pollack 90] plans can be viewed either as an abstract structure which can be synthesised and executed by any computational system, or as a mental attribute of an intelligent agent existing in a complex dynamic world. Some researchers treat plans as if they are executable programs [Musliner 96]. This view is not wholly suitable for our dynamic world. In our progressively abstract model, plans are viewed as mental attributes at the higher levels; and at the same time lower subplans are viewed (partially) as executable programs.

Plans in [Wilkins 94] are represented as abstract structures. Wilkins integrates sophisticated planning techniques (SIPE-2) with a reactive execution system (PRS). When plans fail, error recovery procedures are used to re-plan, ensuring that the future consequences of the new plan do not interfere with the still-active execution threads of the original plan. The plan executors need to encode procedures that will allow recovery from failure states. Our system extends the techniques used by Wilkins in that potential plan failure can be detected in advance. As a result, re-planning for the failed future plan segment can be carried out in parallel with plan execution so that the current set of activities may have no need to be aborted.

Tambe [Tambe 97] discusses executing (team) plans in a dynamic environment (a battlefield simulator) where team plans are *explicitly* provided to the agents. However, the representation of these plans appears to be explicit only to the agent designer, and not necessarily to the agents themselves. Such plans are hard to maintain during execution when the environment changes.

In [Haddawy 96], abstract plans are derived while searching for completely refined optimal plans. The DRIPS planner permits interleaving execution and planning. This could result in a plan in which the initial segment is completely refined but the future segments still remain abstract (though not progressively abstract). However, this abstract structure appears more incidental than intentional; further, the effect of world changes on the plan and its abstractions have not been considered.

Knoblock [Knoblock 94] presents automated techniques for generating abstractions for planning. His algorithm generates abstraction hierarchies by dropping lower levels of abstractions from the original problem definition. In our approach, the agent generates new abstractions based only on its primitive actions and they are used to reconstruct the world model that is more suitable for a planning agent.

Steel [Steel 88] implemented the IPEM structure which enables interleaving planning and execution. Control of the system is done by IF-THEN rules production system architecture. Both our plan structure and the IPEM offer replan capability either after execution failure or after the occurrence of unexpected effects, and both systems are based on rule based architecture. In constract, we provide concurrent planning and execution, and the capability to handle agent generated abstract resources.

Firby [Firby 87] proposed a reactive action package (RAP) that pursues a planning goal until that goal has been achieved. Failures caused by unexpected world situations are relegated to the underlying hardware interface in an attempt to remodel the world.

Chapman and Agre [Chapman 86] do reactive planning organised around situation-action like rules. However, they do not seem to use abstractions for handling action failures, since they do not employ explicit representation for plans.

In our approach agents often modify their plans at the lower levels in their plan structure, leaving the over all plan appearance unaffected as far as possible. This is important because agents often use each others plans as a basis for working out co-operation strategies. Dynamic environments also require that the agents be capable of building abstractions on their own. While traditionally long flat plans are too long and therefore difficult to maintain, uniformly abstract plans are not responsive enough for changes that happens in the immediate future. Abstraction in plans seem to generate an agent behaviour wherein the agent appears to care for its future more systematically. As a result we believe a progressively abstract plan is more suitable for agents residing in a dynamic environment. In this paper, the abstractions generated are specific to the agent that generated it, and are not suitable for communication purposes. Our future work will involve generating abstract operators automatically and abstractions that are compatible to all agents in a society.

References

[Tambe 97] Milind Tambe. Implementing Agent Teams in Dynamic Multi-agent Environments. Applied AI, 1997.

[Au 97] Sherlock Au, MPS: A Multiagent Production System Language, Master Thesis, UNSW 1997.

[Haddawy 96] J. Helwig and P. Haddawy. An Abstraction-Based Approach to Interleaving Planning and Execution in Partially-Obserable Domains. Working Notes of the AAAI Fall Symposium on Plan Execution: Problems and Issues. Cambridge, MA, November 1996.

[Musliner 96] David Musliner. Plan execution in Mission-Critical Domain. Working Notes of the AAAI Fall Symposium on Plan Execution: Problems and Issues. Cambridge, MA, November 1996.

[ICJAI 95] H.Kitano, M.Asada, Y.Kuniyoshi, I.Noda, E.Osawa. Roboup: The robot world cup initiative. In Proceedings of ICJAI-95 Workshop on Entertainment and AI/Alife, 1995.

[Knoblock 94] Craig Knoblock. Automatically generating abstraction for planning. AI journal Vol 68. No. 2 1994.

[Wilkins 94] David E. Wilkins and Karen L. Myers, A Common Knowledge Representation for Plan Generation and Reactive Execution, Journal of Logic and Computation, 1995.

[Pollack 90] M. Pollack. Intentions in Communication. Edited by P. Cohen, J. Morgan and M. Pollack. Massachusetts Institute of Technology. 1990.

[Steel 88] Jose Ambros-Ingerson and S Steel, Integrating Planning, Execution and Monitoring. AAAI, 1998.

[Firby 87] R James Firby, An Investigation into Reactive Planning in Complex Domains, AAAI, 1987.

[Chapman 86] David Chapman and Philip E. Agre. Abstract reasoning as emergent from concrete activity. In Workshop on Planning and Reasoning about Action, Portland, Oregon, June 1986.