Lilies - a framework for building multiple agents for adaptive planning

Patricia Charlton *
CSISE
115 New Cavendish St
London
England
charltp@westminster.ac.uk

Abstract

Lilies (Localisation and InterLeaving stragIES) was developed to deal with a forest fire fighting planning environment. The application required adaptive planning in a reactive and generative environment. To model the application, it was necessary for multiple agents to be developed which added the usual communication [15, 19] issues. The planning environment needed to deal with both reactive and generative problems. To provide a unified and adaptative environment we used reflection [20, 13, 4, 18]. *Lilies* provides a framework which develops and extends the taxonomy of actions [16] and the interleaving [1] for planning environments. To deal with the planning in a reactive and generative environment, and having only partial information required a system which could evaluate, interleave and adapt on both single and multiple agent(s) model. In this paper we explain why and how we developed *Lilies*, a system which provides agents with a means of inspection and introspection [4].

1 Introduction

The motivation behind *Lilies* was to deal with the planning problems of a forest fire fighting environment. Our theory was based on a hypothesis, that strategies as a plan driver should exist at various levels in a planning system, these levels would contain abstraction details from operational facts to meta-level reasoning. This approach was necessary if the planning system was to be flexible enough for adaptation to take place at all levels of plan reasoning. We were considering the problem of agents generating their own plans once they have been allocated a task, and also the execution of these plans. The process of generation and execution also had to deal with the unexpected, this either being a reaction to danger or a failed plan. We assumed that each agent may not have complete information [10] for a goal and so developed a number of methods of dealing with this problem.

Communication between agents operates at a level of either requesting information or giving information. If an agent is unable to complete a plan there are two levels of operation that can be used. The first is reflection and the second is the observer. Each agent contains an observer which can communicate between the physical level and the "thinking" level. The observer is an abstraction of Newell's [17] knowledge level and can act in a *part-of* relationship with the agent which it is attending (the observer in *Lilies* moves between the action of being passive and active, where as in Newell's work the observer was a mirror image of the agent. We use our observer to be both a "watcher" and participator). The internal planning level for the agent is checked by a monitor. The monitor is used in a similar way to IPEM [1] in a plan, execute and mon-

*Research done by the author while she was working at Bath University, Bath, England.

itor cycle. However, the agent has a "thinking" environment, in which the agent can generate a plan, and the monitor checks the plan at the generative stage and execution stage, hence the need for an evaluation mechanism, which is based on McDermott's [16] taxonomy of actions.

In this paper we will first describe the physical agents that are necessary for the application and the type of planning tasks these agents will perform. We will then explain how *Lilies* is used to represent these agents and deal with the planning properties.

2 Physical Agents

Agents are objects which are part of the problem domain. In our application we recognise five: (i) Bulldozers, (ii) Fires, (iii) Environment, (iv) Central Control and (v) Air-reconnaissance. State information and aims are associated with each agent; for example, if there is a fire, a bulldozer will try to prevent it from spreading. The problem domain itself also has aims, which, for this problem, are (i) prevent harm coming to the bulldozers, (ii) prevent forest fires, (iii) minimise destruction of the forest. However, in order to satisfy the third problem domain goal, a bulldozer must take risks, which conflicts with the first domain goal. To mediate between these demands it is clear that control must be both local—in a bulldozer—and global, corresponding to the problem domain. This is similar to the Phoenix system [6], where these two levels are called *cognitive* and *reflex*. Which are known in planning as generative and reactive, respectively.

3 Representing the agents

The agents are built using the blackboard architecture [9, 8]. Each agent can be thought of as a small blackboard system, similar to CASANDRA [7]. However, the agents are able to reflect and adapt to the environment. In the planning context they are physical agents which interact with the environment but are also able to generate plans in their own environment. This representation means the monitoring of plan execution can occur and the problem of adaption can be analysed. To allow for localisation as an agent representation and grouping we have extended the blackboard [4, 3, 5].

Figure 1.1 below demonstrates the levels of operations and relates to OPM [11]. There can be a number of agents which interact in different ways on the environment level. As explain previously an agent can only contribute if it exists in the environment. The levels are updated via the use of knowledge sources. The database only contains entries which are changed through events. The agent knows the knowledge that it has access to. This is explicit on the knowledge level. The monitor is not represented on the agent level as this operation is an action at the generative level. This is a class of reasoning, that is summing up the information and making a decision [5] if the plan is failing. We have not yet explained how reflection can be incorporated in the context of generating a plan. This requires the knowledge sources to take into account of applying this level. The monitor is part of the agent as a physical consideration. This way of thinking about the representation can be thought of as a grain size and we can look behind the representation of the object to see more information about that object.

Part-view of the blackboard Database

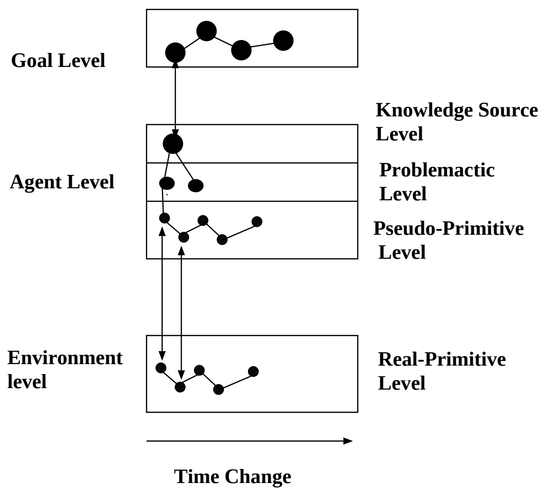

Figure 1.1

The environment holds the active entries of the agents involved with the planning system. These entries change as their location state or action state changes. The agent's generative representation has its own blackboard. If a focus is placed on the environment then the entry's current positions and actions can be identified. To consider the next action of an agent requires looking at the agent's blackboard.

The self-model of the agent can be seen when the agent is generating the solution to a problem. It is when the agent attempts to execute the solution that the task becomes explicitly a worldly operation if the operation contains some physical interaction, for example moving from one position to another. There are problem solving tasks which are purely at a generative level. This means the agent considers a task only to the pseudo-primitive level. This operation can occur for a number of reasons: (i) the problem has changed or (ii) the agent is speculating on the options to a solution. Generating a plan to a pseudo-primitive level does provide a large amount of detail. The philosophy of *Lilies* is to have various levels of planning. This is achieved using the extended taxonomy of actions (see the next section on evaluation) and the blackboard database. In using the blackboard model the agent can contain these different levels of planning and the self-model. Each agent is a blackboard localisation [5] which means that the plan modeling can be explicitly seen and can contain various models of planning. The blackboard localisation is not a typical feature of a blackboard architecture. In using the extended taxonomy of actions we are able to see the agents' self-model of planning and contain an actual history of the execution of the plans. The monitor is part of the agent and is used to test out results, that is, did the result meet the expectation (pseudo-primitive match the real-primitive). In using the blackboard database we have a method of modeling and reviewing, that is, the agent has a cognitive level for generating a plan and a historical level of past events. This links in with research in CBR systems [18, 14].

3.1 Evaluation

To support the theory of requiring a number of levels and evaluation we draw upon McDermott's work. McDermott [16] developed a taxonomy of actions for describing a task. Tasks are broken down into sub-tasks until a set of executable actions can be performed. Actions are either *primitive* — can be executed — or *problematic* — cannot be executed and require further analysis. The notion of a task (in terms of McDermott's work) is any describable action to which the interpreter is committed.

If we consider `TaskA` to be one level which is problematic then the sub-tasks may themselves be either problematic or primitives, although McDermott only considers a primitive to exist if the code is executable. This is not the approach we take when describing the type of strategy we have put forward.

First we will describe a two level strategy. Here we will provide an example of the terms problematic and primitive. A problematic is a task which requires more work. A primitive is a task which is executable. In this abstract problem of defining `TaskA` we class `TaskA` as problematic. `Sub-task1` and `Sub-task2` are primitives but only in respect of `TaskA` and in relation to `GoalA` — which is to achieve `StateB`.

Let us give a more concrete example and use the forest-fire fighting domain.

- `GoalA` – to have no fires
- `TaskA` – to put out the fire
- `Sub-task1` – work out where to move bulldozers
- `Sub-task2` – Can we move the bulldozers?

`TaskA` has two sub-tasks. In McDermott's taxonomy these would need further refinement and the sub-tasks would be classed as problematic. The strategy employed by NASL is not the approach provided by *Lilies*. The fact that a task was classed as problematic would not necessarily require further work. Doing more work on a task would depend on the goal set out by the application. The class of the task is not dependent on just its executable properties but on the task's relation to the rest of the solution. The construction of such a task hinges on the concepts of evaluation and interleaving. Rationalising these concepts into one framework allows our strategy theory to exist. This framework benefits from reflection as the interleaving can exist at levels and provide change.

We will describe this new approach to the strategy representation by building on the starting example outlined above. In terms of the system we are describing here we would express a strategy as — strategy-A requires `TaskA` to achieve `GoalA`. The strategy can provide a solution to `GoalA`, in this case using `TaskA`. The amount of detail represented in the solution depends on: (i) the goal requirements in terms of the problem and (ii) the strategy objectives. The strategy objectives are not purely domain dependent. The main purpose of the strategy is to satisfy the goal but the strategy may be required to perform other sub-goals. These sub-goals may only exist through the main goal, for example (i) main-goal – Prevent forest fires and (ii) sub-goal check all bulldozers are in working order. The main goal could succeed without considering the sub-goal. This classification relates to McDermott's Primary and Secondary characteristics of the actions.

The strategy has the main objective — `GoalA`. The strategy may also have to report information in a certain form at different stages towards reaching the goal. The purpose of this information is to "assess how well the goal is achieving". We could imagine reporting this information to another "system". There may be no other purpose than collecting the information for future analysis, for example as an off-line mechanism. There is another possibility and that is the strategy itself is analysing the stages of the developing solution. At this point the reader may feel that this process is dependent on the success of `GoalA`. In classical planning systems such as ABSTRIPS the goal can be achieved without having to apply some analysis of the strategy applied. The strategy in ABSTRIPS applies a means-end analysis and provides a set of actions as the solution. By allowing re-analysis the system can check at each stage its own performance. If the system is given a performance evaluation mechanism then the system can determine within this evaluation how the system is performing. Using this evaluation to make decisions may mean the system can improve its own performance.

If we consider a two level view then the problem would be expressed as (`Problematic TaskA` (`Primitive Subtask1`, `Primitive Subtask2`)). From an execution point of view (see both McDermott and Ambros-Ingerson) we may not have a primitive, the task requires further work. From our *Strategy* perspective, if the context is correct then we do have a primitive. In this respect we are generating a plan and only evaluating to either the level required or to the level of detail available. Reporting the *strategy* solution would take the form of the set of actions and the type of actions they are classed as. Although the solution contains primitives they have not yet been executed. The type of structure presented is viewed in terms of the problem goals and the information which can be derived from the domain. The (sub)tasks can be looked at in further detail to see their structure and to assess how much more is required to evaluate the task to an executable primitive. This evaluation is only a prediction. Not until the actions are executed and the result returned has the plan been executed (again see McDermott [16] and Ambros-Ingerson [1]). It is at this point that a plan succeeds or fails (although the plan might fail the strategy can succeed). If we add more levels then this execution takes on these levels.

McDermott classifies actions as either *inferential* or *worldly*. They are used to talk about dependence. We do not consider a task to be totally inferential or worldly because they are extremes which describe the boundaries of an action: totally inferential would be a closed system. Completely worldly implies total dependence and contributing nothing. For any relationship to exist means the extremes are not pure. The inferential means as contained as possible. `Subtask1` is inferential (the classification of a task becomes subjective and strongly dependent on the context of how the domain is viewed).

Once the information of the position of the fire is passed to the subtask then the operation can take place. The second task is less inferential or more worldly because the dependency of the task on the information from a *Subtask1* and the world state of the active bulldozers.

`Strategy-A` could be further defined as `Problematic TaskA (Subtask1 Primitive Inferential , Subtask2 Primitive Worldly )`

McDermott classifies defines tasks *primary* and *secondary*. In the case presented here `TaskA` is resolving only one goal. `Subtask1` and `Subtask2` have their own goals (which are subgoals to the overall goal) to achieve `GoalA`. An ordering does exist between the tasks as *Subtask1* must occur before *Subtask2*. The concept of priorities is applied to the goals. However, this is not strictly the concept McDermott is applying to the use of primary and secondary. Below we expand our example to include primary and secondary in the way McDermott intended.

Another goal we can apply to the forest-fire fighting problem is *do the least damage to the forest by the bulldozers*. We will call this goal `GoalB`. We consider `GoalA` to have priority over `GoalB`. Putting out the fire is the priority but we want the strategy to also limit the damage the bulldozers cause. The primary goal is still to put out the fire.

We define Strategy-A further:

```
Strategy-A Motivated by ((GoalA Primary,
GoalB Secondary) Achieved by TaskA
(SubTask1 Primitive Inferential, Subtask2
Primitive Worldly))
```

As seen by the earlier description of `TaskA`, the task is decomposed into further (sub)tasks. To achieve the goal allocated to the task means the task has a local strategy. This process of allocating tasks, local strategies and goals can be applied until there exists a task which is primitive in the executable sense or the goal is satisfied. Perhaps execution is not desirable. There is a disadvantage in reducing the problem to too fine a level of localisation – that is the maintenance of the communication between objects. The distribution of the problem needs to be organised during the development of the planning system. Essentially the execution (primitive) level only needs to be apparent when that operation is to occur.

3.2 Interleaving

A theory of plan interleaving is put forward by Ambros-Ingerson and Steel[1]. They discuss the problem of plan execution and explain when a plan has been executed. The system IPEM (interleaving plans, execution and monitoring) provides an incremental system. A plan is generated and then executed. The result of the execution is monitored. The argument they put forward is important in terms of unexpected change. IPEM generates a plan to a level then executes the plan. Perhaps the world has changed and the plan did not achieve its goal. If this is the case IPEM creates a list of flaws and then fixes are applied. We consider IPEM to be incremental and operating at a low level.

These operations have the initial makings of our *strategy*. They evaluate an execution. However, this theory of IPEM is only applying a method to two levels. The cycle is to generate, execute, monitor and fix. If we apply this theory to our many level structure the evaluations are used to determine if the goal is achieved. If a goal is not being achieved then some “fix” can be applied.

This is not the whole story of applying some “fix” in the context of a strategy environment described here. The flexibility of the strategy means that the strategy needs to determine if a “fix” is necessary. Perhaps the goal has been abandoned or another approach to the problem has been developed. More perspective is gained through a definition of the meta-level. Applying the meta-level develops from McDermott’s idea of a strong incremental heuristic system, that is appreciating and taking advantage of the change. The self-referencing allows us to apply the change to the strategy itself. This move towards a theory of meta-level which the strategy can apply means the system can adapt in context.

In terms of the IPEM system we would want access to the interpreter which runs the interleaving mechanism so that the changes can occur at the strategy level as well as at the "object" level.

4 Reflection and the meta-level

The purpose of the meta-level is to hold theories about the "object" level. In the model used by *Lilies* there are a number of ways a meta-level can be considered, these are: (i) a planning context, (ii) the architecture, (iii) the agent representation and (iv) evaluations. There are very obvious levels, such as, generating a plan and considering re-planning via the meta-theory. The architecture is based on the blackboard model and holds structure which can be used to manipulate a theory through the meta-theory. The agent representation could hold a self-model to provide a meta-theory about it's own objectives or this meta-theory could be external and purely represented as part of the architecture. The evaluations, as a language for representing the level of operation of the strategy, can hold models of the theories of lower level strategy concept. This form of representation is interesting to planning and the structure is expected due to the type of planning frame we are providing. This will give us a consistent model representation to execution of the agent's activities.

Lilies requires the meta-theory to exist at each stage for the type of strategy to be represented, modelled and executed. This method of representing the meta-theory means that the overlap occuring through the modeling becomes part of the modeling. This overlap is part of the strategy theory and is dealt with for implementation purposes by the *observer*.

Our meta-theory needs to have a model of generating a plan, dealing with a reactive situation, have a concept of executing a plan and so deal with re-planning and have evaluations which not only consider the immediate plan and execution but can engage reasoning for future changes before the time is reached, that is have some concept of a reflective reasoning process. Each agent is to contain this theory and so can achieve self-modeling which again incorporates reflection. The strategy is presented using these evaluations.

4.1 The Self-Model - Reflection

Self-referencing is a term used to imply a rule which calls itself [2]. The concept is to use the calling rule when an answer can not be determined. To apply the self-model the concept is the same, only the agent applies a model of itself. This operation is applied in dealing with errors or anticipating the generative plan behaviour.

In analysing the self-model the theory follows in a similar manner to the knowledge-level concept of the observer. For the self-model to exist means the agent has to have an understanding of its own operations and be able to apply this model.

4.2 Meta-level

To have the concept of the self-model requires the existence of the meta-level in a system. This operates the meta-theory [18] as the reflection principle specifies the relationship between a theory and its meta-theory. The upward principles specify the reification process that encodes some aspects of the theory, so at the point of reification the system has constructed a model of the theory.

The meta-theory in *Lilies* is not so extensive as the MILORD-II system. However, the system follows similar principles which operate in the context of agent(s) and planning. The theory is applied to the concept of the agent planning in a generative and reactive environment. The meta-theory gives a framework for the interleaving to operate on a structure containing multiple planning levels as well as a number of agents.

If the meta-level is considered then the purpose is to allow change to the agent's information either in a reactive manner or on a generative level.

4.3 Meta-level for Planning

To explain how the meta-level will operate in the planning and blackboard frame requires an example. We will use the example of the operation to move a bulldozer from one position to a new position.

The first stage in the blackboard model is to create a bulldozer. We will define a knowledge source which creates bulldozers. In creating a bulldozer on the blackboard we have a position and a name.

We have an event identity for creating bulldozers which we can use to group all the bulldozers together. This allows the bulldozers to act as localisations or as a group. This localisation as used by *Lilies* is not a typical feature of the BB1 architecture. In this example we will concentrate on one bulldozer showing how the agent is represented through the use of the blackboard model and we will explain how the model is extended for planning and simulating physical agents.

In running a blackboard task called *create-bulldozer* we will have created a *bulldozer-1* and we could give the bulldozer the position (1,4). There is an environment grid of 100 squares and the environment can be described through the use of a knowledge source. The environment is described as having *difficult-areas*, *impossible-areas* (such as mountains and lakes) and *possible-areas*. These areas of description are used to calculate how long a path will take for the bulldozer to move from one position to another position.

We can now create a task called *move-bulldozer*. We can post this as an event. However, we must state whether the goal is to be generated or executed or an interaction of both. For the purposes of this example we will generate the plan first. The representation of the plan will be a set of events on a blackboard database. In preparing the plan, the bulldozer may know something about the environment, in which case to generate a plan and then execute this plan would seem appropriate. When the bulldozer does not have much information then the level of planning and execution will follow much closer to IPEM.

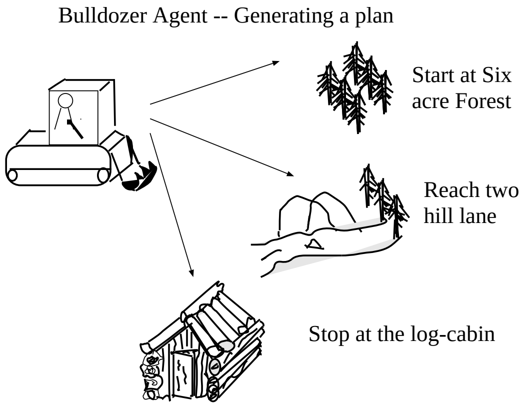

Figure 1.2

Figure 1.2 shows the bulldozer having to plan a route from six acre forest to the log cabin. The route takes the bulldozer through two hill lane. This is a visual representation of how an agent might consider making a plan. A great deal of information needs to be held to make even a simple plan. We will make the assumption that the bulldozer already knows the path to the log cabin.

The goal is classed as problematic and a solution of pseudo-primitives will be generated. The description of the task takes the following form:-

```
(Problem type, task, parameters)
for example
(Problematic move-bulldozer position-1
                          to position-2)
```

The parameters are dependent on the task. There might only be partial information for example we might need a bulldozer at position-2 without knowing the actual positions of the bulldozers in the forest.

In using the blackboard the first stage is to post an event to initiate a problem to solve. In generating a plan the bulldozer determines that to move from six acre forest to the log cabin that it will move through two hill lane. The first stage of generating the plan is to return the following path:-

(pseudo-primitive, move-bulldozer,six acre forest, two hill lane, log cabin)

As the task has been reduced then the type of task changes. This is for two reasons: (i) For monitoring (to know whether it has been executed or not in terms of planning) and (ii)as strategy representation which may only require a pseudo-primitive or select parts of the plan that should be executed, for example, having reached two hill lane the bulldozers task might change and with this its destination.

For our theory of flexibility to hold true means that there are different possible information flows, either from the agents reflective ability or from the changing environment. There are possibilities that the task becomes redundant due to an overall strategy being considered by other agents. So the concept of the strategy not only exists in a many structured form for one agent with tasks being executed to different levels (either generative or reactive) but also on a larger scale of interacting with other agents. The interleaving concept is used at both levels. The strategy either existing as part of one agent or multiple agents contains this adaptive property which can operate on an immediate problem or on generating future solutions.

We can use the blackboard to map the names to value coordinates and execute these values with the operation of executing the task move-bulldozer.

4.4 Representing the generated plan

The BB1 interpreter uses the concept of posting events to the blackboard database to show a developing solution. In generating a plan we need to keep each stage of the activities which make up the plan. When an event is modified, the modification process needs to be recognised. However, on successful execution of each event via a knowledge source the result is the modified event. On the generative view we are considering a cognitive process and the database is a memory which holds the solution. From an execution view the database is a physical view and the state of the "object" can be seen immediately on execution completion.

We can use the blackboard knowledge source structures to our advantage for demonstrating reactive problems. We can simply set up a knowledge source to be triggered at a cycle time which will be executed as part of the environment. This can also demonstrate the concept of reflecting about a current plan. This would only take effect if the agent had access to some new information. The agent may generate a plan to refuel at a location and later may be informed that the location no-longer has any fuel. The plan may only have been generated, so parts of the plan may have been executed but the opportunity is available for the agent to replan before the crucial stage is reached.

We use the notion of problematic and pseudo-primitive to distinguish between a plan which may require further work if physical execution is to take place and a plan which is developed for physical execution. A plan can be made up of various levels of development from problematic to a real-primitive. This means parts of the plan have been executed, other parts require execution and some parts are still being worked on. This is plan representation for the agent, the strategy holds the whole method of planning activity as the process of interleaving.

The basic phases of generating a plan are: (i) set the goal, (ii) post the goal as an event and (iii) generate a plan held as problematic and/or pseudo-primitives. The event is provided with a level of problematic and/or pseudo-primitive. We differ from the classical blackboard in our use of events. This is to give a richer cognitive level and to provide a self-model. In generating a plan we have stages which are implicit operations in the blackboard of triggers and preconditions. However, in our strategic model we require this information to be explicit. This means that the agent drives it's own localised planning environment. Care must be taken that the agent only has access to its world (including interactions with other agents) and does not have "system set up" information, that is special knowledge sources we use to create reactive changes.

The agent contains in its planning capabilities the triggers and preconditions which must be true for a knowledge source to be fired and an event to occur. This complete self-model makes *Lilies* different from other planning environments. This concept of the strategy operating at this system communication level (having given the agent the knowledge of the driving system) means incremental change can be realised and as a consequence the agent is reflecting when self referencing occurs with a realisation of the system that is driving the agent. For this level to operate means the agent has another level of planning which is the knowledge sources that are required to achieve the goal. This provides a cognitive context which is included in *Lilies*

This level of planning is necessary for the completeness of *Lilies* in terms of the theory and implementation. The agent can drive the system once it has determined a solution. This provides interesting levels of planning considered earlier, such as the representation of the agent, the reasoning behind the agent and how the strategy is applied. This provides the generative representation using a blackboard frame. However, in executing the plan the agent updates more than the blackboard for generating a plan, it also needs to update the global blackboard for the physical properties. The operation of the execution of the plan occurs after completion of the generated plan. As explained earlier the agent may develop a partial plan. The information for determine which events are developed to pseudo-primitive level and which are executed are contained in the knowledge source level of the events. To summarise an agent's blackboard where the generative task occurs has three levels: (i) knowledge source, (ii) problematic and (iii) pseudo-primitives. The knowledge source level directly contains the knowledge sources which are required for the plan and the conditions which must occur if the events are to be generated in the form developed at the problematic and pseudo-primitive level. Applying the knowledge source level in a reflective sense requires *lilies* to be in operation as this is not a blackboard feature.

4.5 Executing a plan

Before execution can occur the plan needs to have been generated in some form. Before execution can be achieved the events need to be in a pseudo-primitive status. The agent has three levels in the generative process: (i) knowledge source, (ii) problematic and (iii) pseudo-primitive. The knowledge source level contains the "knowledge sources" required to solve the problem. The problematic contains the goals and tasks which are currently being worked on. The order of events are kept by using the blackboard cycle time. The pseudo-primitive is an event which would normally be posted when running the blackboard. The chain of events are kept to show the process of achieving the result.

Each knowledge source has two trigger types for the level of problematic and execute. When the execute level is triggered the plan is then executed, again as a blackboard task. Each execution makes a call to the monitor which takes the result of the real-primitive and compares the result of the pseudo-primitive.

4.5.1 Generate Plan, Execute and Monitor Cycle

During the set-up stages, the environment database for the physical description is defined. This includes initialising a bulldozer to exist as an event on the environment blackboard along with the description of the environment.

The environment contains the agent as a blackboard event. This is set up for the plan execution process, that is when we want to simulate real-primitives. The agent is classed as a blackboard event as it can make changes to the environment just like a fire is classed as an event. The environment is described and stored on a blackboard database. To make the system more practical for development purposes the set-up uses knowledge sources to create the environment. This means changes can be easily made to the environment for development purposes in the reactive planning stages.

Once the bulldozer is created it is allocated a local area to hold the generating plan process. This process creates the localisation which is necessary if the strategy is to operate through a generative, execution and monitor stage – which is essential if reactive environment changes are to be considered (also this is necessary for considering agent resource clashes).

Accessing the bulldozer event in the environment we use the environment entry identifier *create-bulldozer*. The level of access is *environment* and the attribute is the bulldozers identifier, for example *bulldozer-1*.

The agent-bulldozer is also defined using a knowledge source structure and can only be triggered once the environment event is true. The environment set up just gives a position of the bulldozer.

Once the environment set-up is complete then the goal list is fired. Again this is defined as a knowledge source structure. The agents are triggered if they exist in the environment and they can help achieve a goal. In the simple example we have been using to move the bulldozer would mean an agent would have to exist in the environment and have the knowledge of how to move. If an agent doesn't have the knowledge to complete a task and no other agent is able to, then if the knowledge source exists, the agent can have access to the information through the observer. The knowledge source is then posted on the agents blackboard for future access via an event. It is this type of event which allows communication between the agents and uses the observer. The observer reviews the blackboard database on a global scale looking for the knowledge source required for the task. If the knowledge source exists then the agent which requires this information needs to go through the process of reflection to make the "new knowledge source" part of its environment. However, it is possible that the knowledge source is special to an agent and can only be used by that agent, in this case the part of the task which requires the knowledge source will have to be re-planned. The re-planning will mean that one agent will only be able to complete that part of the task to a problematic level and in providing the solution will describe the task allocation as problematic and worldly naming the agent responsible.

Once an agent has been triggered to complete a task then the knowledge sources which perform the task are applied. Events are posted on a generative level of the blackboard until the task is complete or there are no-more knowledge sources to fire for that agent at the generative level. If there exists pseudo-primitives then these can be executed. The process is to apply the knowledge source at an environment primitive level. This is when the monitor is used to compare the event at the *real primitive* stage back to the *pseudo-primitive*. The monitor is used when the generated plan fails. The monitor is still part of the agent but is used to trigger the re-plan process stating the new goal, which is the new starting position in terms of moving the bulldozer.

5 Overview of Lilies

The previous sections explained the theory behind the type of strategies we consider necessary for a planning system. For *Lilies* to exist required: (i) evaluation, (ii) interleaving, (iii) levels and (iv) meta-level and (v) adaptation. *Lilies* extended the taxonomy of actions and the interleaving theory used in classical planning to build a more flexible system to include adaptation. For a rationalised framework to exist required the use of reflection and self-referencing in an agent's model. Here we will provide an informal notation for providing an overview of *Lilies*. In our planning system, our theory requires two levels, one is the representation the other is the operation of the strategy. The strategy for overall modelling is an environment strategy, that is the real-world simulation and the agent strategy is a contained process. It is due to this operating at two different levels that the theory of *Lilies* becomes rationalised. There are two systems the overall goal and the agents. In terms of multiple-agents there is a need to co-operate [15]. In *Lilies* the agents operate on their localisation and interact with an observer. It is this interaction with the observer that allows the agent and the "system" to co-operate. In terms of modelling a planning system this method of the agent being reflective is different from other multiagent planning systems.

We will give an overview of the types of functions and operations which are required. These functions and operations will be described in more detail in the sections to follow. There are points when the system follows the normal blackboard operation, these operations will not be expanded as they are described in the literature [9]

A task is engaged by posting an entry on to the goal level. This operation is usually used to start the whole planning cycle and is a classical blackboard function of **post-first-entry**.

The operation of allocating a goal/task to an agent is carried out by the function *allocate-goal*. This function is defined using knowledge source format of the blackboard architecture. However, before a goal can be allocated requires agents to be initialised. That is, to exist as an entry in the environment and to have a concept of a *knowledge-level*. The knowledge-level contains the goals the agent can satisfy.

Setup is the first operation once a goal has been posted. There are a list of agents which can either be supplied in the function **post-first-entry** or a default set can be used. The method used for setting up the environment is application specific. In the case of the forest fire fighting problem we have both a physical representation as well as the agents having a self-model and generating plans. There is a function *setup-agents* which will take the list of agents as a triggering condition and create entries on the blackboard.

The first stage of this is to create the entry in the environment and the second stage is to create the agents as an entity in their own right. that is, their own generative space. In the environment domain their are facts which describe the terrain. There are area descriptions which described in more detail in later section.

An agent is triggered if the operations are *setup* or *goal-match*. The setup stage needs to be completed if the allocate-goals is to succeed as the *goal-match* operation can only succeed if the knowledge-level exists.

An agent has three types of processing, that is to, reflect, generate and execute. An agent can be called with *agent(agent method-type)*. An agent may reference its self or another agent when a task is being deligated. The method-type is either *reflect* or *generate* or *execute*. Through the interleaving process the agent can cycle between generate and execute with varying intervals. Reflection deals with an impasse and occurs through the use of three operations: (i) reification, (ii) meta-level and (iii) reflection. The whole process is triggered by the monitor.

If the goal matches the knowledge-level for an agent then the following function is triggered into operation; *generate(goal)*. The context of the generation and execution is described in terms of the value of the variable agent.

We can do the normal blackboard operations of *$entry-level(evaluation-level)*. This will return, in the context of the agent the, following entry information: knowledge-level, problematic-level and pseudo-primitive-level. There is a meta-level which operates the reasoning about the planning methods when problems arise and the entries to this level can be accessed by the normal blackboard operation. However, the meta-level methods are not the usual blackboard operations. With each level there is a method-type to determine the scope of the operation, that is, to execute into the physical environment or into the agent's environment. This combination of method-type and level operates the interleaving process and strategy communication.

We may also have a physical representation of the agent in terms of the environment. We have context of the agent being an object or as a problem solving entity. As an object reactive events can occur causing a change to the agents own knowledge processing. As a problem-solving entity knowledge changes can occur which may or may not affect the physical representation.

We use classical blackboard operations to tell us where the agent is – *($get blackboard-level agent)*. However, we may want the agent to have a concept of its physical representation. The physical representation is required when the problem solving task requires this. *agent(object method-type evaluation-level rule entry)* requires a context when applying to its internal representation and external representation. The context or domain is the object, if the object is the agent then the request is to apply the method-type of execute to the level real-primitive will result in the agent applying a rule (knowledge-source-level) to achieve an entry (goal, sub-goal or entry).

The agent is a localisation and of the type agent, this type is represented as an *id* in the blackboard rule structure. To generate these localisations we have a *create-agent* which creates an entry and associates this entry to an attribute of the type agent. The *create-agent* within a knowledge source requires the use of the functions *$ASSIGN* and *$VALUE*, these are BB1 operations for binding and accessing local variables to a knowledge source. It is not a typical feature of the blackboard to have local environments as a explicit representation. These are however, necessary in our planning system to create interactive environments for plan representation and execution.

The sections to follow summarise the planning model and provide an example of an agent system. The model contains facts, theories and methods which are used to realise the strategy. The planning model contains the agent model which has knowledge-level, facts and meta-level. The facts are related to entries which occur through events. The entries occur through the knowledge-source-level which are functions. The meta-level contains theories about the knowledge-source-level. The facts are entries which are either problematic, primitive, environment or knowledge-source-level.

Operations	Meaning
$e = \{f(e) = v\}$	The entries for an agent
ksl("e")	The meta-functions to create the entries The knowledge source level is also entries in the agent's "thinking world
meta("f(e)")	a set of methods containing the context of the knowledge source level functions and facts (entries).
meta("e")	inference theory to contain a model of the knowledge-level and facts

We will apply NOOS [18] inference level theory to our planning frame. The agent has functions to access the facts. The agents planning model contains a set of facts about the domain – $e = \{f(e) = v\}$. The agent has meta-functions to create the entries which operate at the knowledge-source-level. The set of meta-functions for plan P are described by *ksl("e")*. The reasoning knowledge such as move or dig functions. The meta-function *meta("f(e)")* is the set of methods containing the context of the knowledge-source-level functions and the facts (entries). The inference theory is a set of methods for construction our knowledge-source-level theory, that is, representing the model of the knowledge-level and the facts to construct new facts. The inference theory is represented as *meta("e")*.

The domain method is applied through the blackboard cycle organised by the agenda, that is, a list of KSIs (knowledge source instances) is constructed and fired through the agenda. This create facts in the form of entries. The entries are changed or added through events caused by the executing of the KSI. The domain method is described as a method which will operate at the level of creating an entry on the blackboard for an agent. This is done through using the knowledge-source-level.

The inference methods either construct a model using the context of the knowledge-source-level and the facts or apply the new facts through the knowledge-source-level.

Domain method

The domain method M uses the meta-function *meta("f(e)")*. When we use the function we will return either a new entry, or modify an entry or add to an entry in the database. At this level we are taking the *ksl* (knowledge source level) operations and executing them with respect to the entry (satisfied conditions). The domain method can also return function and entry. The *meta("f(e)")* is a set of structures represented by the meta-agenda, meta-ks and meta-level.

Domain Rule

The operations are either to execute the function on the entry or reify the entry. These are described as either an execute f(e, environment) to return a state change or reify(f(e,environment)) when invoked by the monitor. When we execute an entry in the environment it is a direct planning method using the normal blackboard operations.

Inference Method
The inference method *meta("e")* is taking the knowledge applicable to the entry to operate at the meta-level or is passing back into the environment the operation to be executed. The inference method is either operating on the *ksl("e")* level or passing back the new operation to be executed. This different operation to NOOS, as the function operates with the new entry developed at the meta-level. The new entry has been developed and is again used for matching plan execution. Where as NOOS will pass back a function for the case-model.

Inference Rule
The inference level rule operates on reifying the operation level, so at the inference stage the reify(F(E,environment)) becomes *ksl(E,Environment)*. The **KSL** function can be re-written as *(get-ks knowledge-source-database knowledge-source-level)* which is used at the meta-level. The reflect process passes back an executable KSI to the agenda-list for the agent.

Observer method
The observer functions in a similar manner to the inference method but is a communication structure between the agent and the world. There are set of methods that are used by the agent to communicate to the outside world, these are defined as o(c), where c is defined by *meta("f(e)")* and environment. The function follows through the goal allocation if the operation is possible. However, if the communication fails and there is a partial plan to execute then this will be the status passed back to the meta-level. The environment is defined by the internal knowledge processing and the physical representation.

Observer Rule
The observer rule is defined as *o(meta ("f(e)"), environment)*. The environment is treated in differently as the observer has access to all levels to determine if the problem can be fixed. The order of investigation depends on the priority settings of the rules of the local agenda functions. The restrictions on the observer are dependent on the application and representation required. In the case of the forest fire fighting the agents could communicate via the observer.

When the agent is no longer in generate mode the observer is required. The observer becomes part of the model and its operations, when this occurs the observer is active [4]. The functions of execution, plan change and communication with other agents requires the observer. The execution into the environment requires the observer as the environment is a "type of agent" in the model definition. The action of interacting is different from the generating a plan or resolving a "thinking" problem. In splitting these two operations through the use of the monitor and the observer we not only get an interleaving model which operates on a different scale but a representation which allows both generative and reactive events to occur for an agent simultaneously. This mechanism allows the agent to move between modes of generate, reflect and react. The reflect mode does not require the observer for operation as reflection is contained within the model of the agent. However, reflection will operate the observer if the plan can not be achieved within the means of the agent. We need this operation for communication between agents and to consider partial plan execution. The observer is a level of reflection which can consider further information about the environment. The observer does not reflect in *Lilies* as it is an operation of reflection when part of the agent and is an observer of communication when not in operation, that is, viewing change in terms of other agents requests or reactive events which effect the agent. It is at this level of the agent having the monitor, reification, meta-level, reflection and the observer that the operation of either introspecting or inspecting can be considered. This is not procedural type of reflection but inference level which extends the operation of NOOS to deal with reactive planning and change for a number of agents.

5.1 A planning example

The goal for agent-bulldozer is to get to *Log-cabin*. For the agent to move means the agent has to have the knowledge source *move* as part of its knowledge source level. The goal is allocated through the observer, that is, the observer rule is triggered - *o(meta(task(x),environment))*. The task is a move operation so the rule becomes - *o(meta(move(x),environment))*. Before matching the goal with the knowledge level the observer method is applied, that is reflection is used to create a normalised object. The **KSL** function is operated as part of the matching sequence. If the match succeeds then the task is posted onto the agent's blackboard and this action triggers the planning process. If the pseudo-primitive level is achieved and there are no priority reactions then the plan can be executed. The observer as a communicator has a priority list and is used to interrupt a task if another task/goal takes priority. For example if a bulldozer is suddenly in danger then the digging must stop and move away must be operated. The plan is executed in a physical sense to make changes to the environment which the agent is operating in. This execution occurs through the observer but is checked by the monitor. For this operation of execution to occur requires another level of reflection, that is, code execution. The state changes are checked by the monitor. If a plan fails at a point and the next part no longer makes sense the monitor can cause re-plan action, that is, a reification process into the "thinking" area – the generating plans area of the agent. If more information is required then reification can occur again. The operations move from "e", the entry, to "f(e)", the function to *meta(f(e))*. If this level of processing can not resolve the problem then the observer is engaged and the task is normalised as a goal/task for another agent. For example a bulldozer-agent may be at two-hill lane and can not get to the log-cabin because the road is blocked. Another agent (the controller) is requested for help. The agent is given the task and may either have a solution or attempt to generate a solution. This communication of information is dealt with by the observer of each agent.

The structures and actions are represented as knowledge source structures, such as the reification and reflection rules. Using these structures with the monitor and observer allows the agent to have levels of planning and adapt to situations. Also, the communication means the agents can share knowledge. As knowledge sources are used by the blackboard architecture, we are able to use the blackboard problem solving model for generating plans for each agent. However, the localisation and communication is controlled by *Lilies*.

6 Conclusion

The blackboard system was used as a quick prototyping system and as a control structure. The system had to be extended to allow the type of agent and problem localisation necessary for our planning environment. The extensions were achieved by using theories of evaluations, interleaving and reflection described in this paper.

This approach allows the agents to adapt to situations and gives an approach for the agent to be both doing in a physical sense while having a generative model in operation at the same time. This representation is useful in reactive environments and/or partial information systems.

The agent representation outlined in this paper required reflection and the use of a self-model to provide a heuristic adequate [12] system – a system which could immediately realise change. For the agent to be able to move from various levels of planning and execution required each agent to have some form of monitor. The more optimised the representation and structure means these decisions can be incorporated in the initial design and improve performance as the overhead of communication and reflection can be removed. This does require a good knowledge of the application being developed, which is difficult in dealing with reactive environments. To over optimise (as some optimisation after a time is always possible) has the disadvantage of becoming to controlled and hence less flexible in problems requiring adaption. One solution is to use the reflective model to simulate and optimise, as this type of representation means the agent can look ahead to anticipate problems. For example the control-agent realises that the bulldozer-agent is in difficulty as the bulldozer-agent has not arrived at its destination. The control-agent can generate plans to find another solution to a goal. If the goal becomes critical then a backup plan can be used.

The agents are given certain explicit knowledge during setup which determines the tasks they can do. There is the implicit knowledge, such as moving between evaluations. The representation and structure is motivated by the adaption requirements. The blackboard architecture has also influenced the development. This system gives the agent a default control strategy which is useful in the development stages. Having built the system using the blackboard architecture we can now consider a larger scale simulation and execution by extending the concept of the observer level. A system observer could be used to monitor and collate information about the agent's activities. The process of collecting information is made possible as a blackboard function, as the agent is from a blackboard view just another entry. The advantage of doing this would be to anticipate and/or speculate about problems by simulating worlds. The higher level observer basically mirroring the events (at various levels of processing). If problems are encountered then the active agent can be informed, new measures can be taken such as giving the agent the new "problem solving" knowledge, or a request for further resources which may include a re-plan from the strategic sense to remove less priority goals. This type of simulation of environments means that problems can be anticipated and prevented before they are encounter by the physical agents. The information gained by the simulation can be given to the agents to assist in optimising their activities. *Lilies* does use this concept of simulation but only on a generative and execution level for each agent. The next stage is to experiment with *Lilies* to analyse the behaviour of a large scale simulation of observer agents and physical agents.

References

[1] J. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. *Readings in Planning*, pages 735–740, 1990.

[2] B.G. et al Buchanan. *Rule-Based Expert Systems*. Addison-Wesley, 1984.

[3] P Charlton. Building reflective strategic rules for planning. *12th UK Planning and Scheduling special interest group*, 1993.

[4] P Charlton. A reflective strategic problem solving model. *Springer Verlag - Lecture Notes in Artificial Intelligence*, 1993.

[5] P. Charlton. *Reflective Lilies for Planning*. PhD thesis, Maths, Bath University, Bath, In preparation.

[6] R. Cohen, D. Greenberg, D. Hart, and A. Howe. Trial by fire. *AI Magazine*, 10(3):33–48, fall 1989.

[7] A. D. Craig. *Formal Specification of advanced AI Architectures*. Ellis Horwood Series in Artificial Intelligence, 1991.

[8] I. D. Craig. The blackboard architecture: A definition and its implications. Research report, University of Warwick, 1987.

[9] R. Englemore and T. Morgan. *Blackboard Systems*. Addison and Wesley, 1988.

[10] R.V. Guha and Douglas B. Lenat. Enabling agents to work together. *Communications of the ACM*, 37(7):127–142, July 1994.

[11] F. Hayes-Roth, B. Hayes-Roth. A cognitive model of planning. *Readings in Planning*, pages 245–262, 1990.

[12] Peter Jackson, Han Reichgelt, and Frank Van Harmelen. *Logic-Based Knowledge Representation*. In Login Programming. The MIT Press, Cambridge, Massachusetts, 1989.

[13] S. Jefferson and D. P. Friedman. A simple reflective interpreter. *Reflection and Meta-Level Architecture*, 1992.

[14] B. Lopez and E. Plaza. Case-based Learning of Strategic Knowledge. *European Working Session on Learning:LN in AI*, 1991.

[15] F. Martial, von. *Coordinating Plans of Autonomous Agents*. PhD thesis, University of Massachusetts at Amherst, 1992.

[16] D. McDermott. Planning and acting. *Readings in Planning*, pages 225–244, 1990.

[17] A. Newell. The knowledge level. *Artificial Intelligence*, 1982.

[18] E. Plaza. Inference-level reflection in NOOS. *IMSA International Workshop on Reflection and Meta-level Architectures*, 1992.

[19] Doug Riecken. M: An architecture of integrated agents. *Communications of the ACM*, 37(7):107–116, July 1994.

[20] B. Smith. Reflection and semantics in lisp. *ACM*, pages 23–35, 1983.