Declarative Goals in Motivated Agent Architectures

A report submitted for transfer from MPhil to PhD

Felipe Meneguzzi
Supervisors: Professor Michael Luck
Professor Andrew Jones
Examiner: Dr. Tomasz Radzik

11th April 2007

1 Introduction

As computer systems become more complex, abstraction mechanisms have become more and more important. One abstraction mechanism that is increasingly becoming accepted is the notion of autonomous agents [Jennings, 2000], which embody a component independent from direct external control, and which are expected to operate unsupervised for an undetermined amount of time. Research into agent systems has yielded an extensive body of work in terms of both theoretical results and practical models and systems, and is still ongoing. In the particular area of *autonomous* agents, the theoretical work has not always been matched by the creation of practical models. Despite this difficulty, there have been many applications requiring autonomy; for example, control systems for space exploration vehicles, and for operation in hazardous environments. These applications are often specifically designed for their application domain and require regular human intervention.

For agent designers to expect an agent to act autonomously, they must be able to specify *what* the agent must accomplish and allow for the agent's reasoning process to select the best way of accomplishing it. In contrast, current practical agent models require the designer to provide detailed specifications of *how* an agent should achieve its goals (*e.g.* create a plan library), as well as precise descriptions of the conditions under which an agent should pursue their achievement. In this setting, the success or failure of a goal is implied by the successful execution of these plans, so that when an agent selects one of the available plans, it has no way of determining how appropriate that selection is until the plan has either succeeded or failed. Given the association of plan execution with goal achievement, the way in which an agent's plan library is defined might interfere with the agent's perception of which goals are possible or not. Recent work on declarative agent languages has partially addressed the problem by dissociating plan execution from goal achievement, and allowing the agent to try other plans to accomplish the same goal in case the currently chosen plan fails. However, agents still rely on a plan library and lack any kind of knowledge about the plan's suitability for a given situation until that plan has been achieved, so that an agent has no way of evaluating a course of action before trying to follow it. This lack of knowledge regarding the courses of action to follow applies also in the context of multiagent systems. That is, an agent never evaluates whether interacting with other agents is actually required, relying on a set of hard-coded rules specifying when it should go into *social mode*.

Normal living beings constantly evaluate their surroundings and their past experiences to decide their courses of action, and changes to ongoing courses of action are rarely sudden and arbitrary, reflecting a mechanism that is much more elaborate than simply obeying a set of rules or maximising rewards. Therefore, we believe that for autonomous behaviour to be possible, an agent has to be able to evaluate its available courses of action before investing any resources in pursuing them. For an agent to perform this kind of evaluation, it must be supplied not only with a set of design objectives, but also with a mechanism that allows it to evaluate how the importance of accomplishing individual objectives changes in response to events in the world.

1.1 Research problem

Considering the gap in real autonomy of current agent architectures discussed previously, the research problem lies in the integration of two components that we consider key for flexible behaviour and hence, autonomy. First, for agents to be able to truly operate without intervention in a dynamic environment, they need to be able to create plans to address new problems by composing their basic capabilities, instead of relying on specific plans created at design time. Second, by allowing the search for new solutions to occur at runtime instead of simply reacting with pre-defined ones, an agent needs to manage its own reasoning process in what is known as *meta-level* control, to avoid losing its capability to react timely in critical situations. Moreover, when operating in a social environment, it is necessary to consider the behaviour of others in the generation of new solutions, since third parties might collaborate or oppose the achievement of one's goals. Therefore, we intend to investigate the integration of planning and meta-level control into a traditional agent architecture for a single-agent setting in the first part of our research, and later expand it to consider multi-agent interactions.

1.2 Expected Results

Since the research focuses on the construction of a new architecture integrating planning and meta-level control, the main output of this research is the architecture itself, as well as an associated agent language to allow the development of generic agents using this architecture.

Developing agents able to use a planning component together with a declarative view of goals have a number of advantages over traditional approaches:

- by allowing the creation of new plans at runtime, we expect to augment an agent's ability to deal with unforeseen circumstances;
- goals specified declaratively as desired world states are much less prone to omission errors by the designer; and
- a declarative specification of goals underpinned by a planner allows an agent description to be more concise, since the planner can be relied on to quickly generate trivial combinations of basic capabilities instead of having the designer to this manually.

The addition of a model of meta-level control allows agents operating in most real-world situations to assess the consequences of following one of many possible courses of action, as this assessment is not always realisable in the form of quick and hard rules. To this end, the usage of the motivational states as an abstraction for meta-level control provides at least two advantages:

- describing meta-reasoning rules in terms of motivations is more intuitive than a possible abstract representation; and
- there is a rich body of work on motivations from diverse areas of study, such as psychology, philosophy and ethology, which can be leveraged in our research;

It is clear that such an architecture will not be created from the ground up, but rather be a specialisation of an established architecture with an associated language, and therefore we must focus on comparing them to assess the benefits of the new components on a practical level, which are plan to be done throughout this research. Therefore, once a prototypical architecture is designed, we expect to refine it using the feedback obtained from our initial experiments so that our final architecture and language should have clear advantages in terms of efficiency and expressivity.

1.3 Overview

This report is organised as follows: Section 2 briefly summarises existing efforts related to our research objectives; Section 3 describes extended AgentSpeak interpreter capable of planning developed during the first year of research; Section 4 describes another extension of an AgentSpeak interpreter, which uses a model of motivations to perform meta-level control; finally, Section 5 contains our plans for future work towards the PhD thesis.

2 Related Work

2.1 BDI Agents

One commonly used way of informally describing autonomous behaviour is using the notions of beliefs, desires and intentions, in which beliefs describe one's knowledge about the world, while desires are states of affairs one seeks to achieve and intentions are one's commitment to achieving a particular subset of desires. This model was proposed for human practical reasoning by philosopher Michael Bratman [Braubach et al., 2004], to account for the way in which humans select a series of actions directed at the achievement of a larger goal while avoiding spending time considering less important ones. The BDI model serves as an architecture for intelligent agents [Bratman, 1987], using the same mental abstractions of beliefs desires and intentions to describe the operation of software programs. This BDI architecture has become one of most widely known and studied models of deliberative agents, and evolved from Bratman's seminal work [Bratman, 1987] into formalisations [Cohen and Levesque, 1990] and subsequently a more complete computational theory [Rao and Georgeff, 1995b; Wooldridge, 2000].

The components that characterise the BDI model can be briefly described as follows [Müller, 1996].

- **Beliefs** are the agent's model of the current world, as perceived by its sensors, including the knowledge an agent has about how to modify the world.
- **Desires** are a (possibly inconsistent) set of preferences regarding world states.
- **Intentions** represent the agent's choice regarding alternative courses of action, constraining the consideration of new objectives to allow it to fulfil one subset of its desires at a time.

In essence, BDI agents operate as follows.

1. The agent's beliefs are constantly updated by its sensors.
2. The agent chooses an internally and externally consistent subset of its desires. Internally consistent desires are those that do not conflict with each other, like being in two places at the same time. Externally consistent desires are those that are not impossible given the agent's beliefs and are not already satisfied.
3. Considering the chosen desires, the agent commits to following a course of action (or intention) to fulfil them.

2.2 Declarative Agent Architectures

The BDI model has been the focus of agents research for a significant time, and is still ongoing. Examples of recent research include improving the model through the construction of new theories to underpin it as a unified system [van Riemsdijk et al., 2005], and extending pre-existing BDI theories to allow for more flexible BDI agents [Meneguzzi et al., 2004]. Among these efforts, many seek to address the fact that BDI architectures and models tended to avoid including many of the declarative aspects of desires/goals in support of practicality. More specifically, the first instances of complete BDI logics [Rao and Georgeff, 1995b] assumed an agent able to foresee all of the future ramifications of its actions as part of the process of deciding which courses of action to take. This assumption was clearly too strong if computationally bounded BDI architectures were to be constructed. Therefore, when designing practical architectures based on specific BDI logics, modifications were necessary to avoid unbounded computations. Since the agent cannot look directly into future world states and then select the sequence of actions that leads to the desired future (as this would imply omniscience), the inverse approach was taken; that is, an agent would select from a set of known courses of action, the one that would lead to the desired future. In practice, this means that the agent no longer selects directly what he wanted to achieve, but rather what he wants to perform under the assumption that his actions would ultimately bring about the desired state of affairs. This way of selecting agent goals was later dubbed goals *to do* [Winikoff et al., 2002].

Concurrently with goals *to do* are what have been termed goals *to be* [Winikoff et al., 2002]; the difference being that here, an agent selects the desired state of affairs directly. Consequently, the actions required by the agent to reach such a state of affairs are decoupled from the ultimate goal. The most widely known BDI agent implementations bypass this problem through the use of plan libraries where the courses of action for every possible objective an agent might have are stored [d'Inverno and Luck, 2004] (which we have seen are associated with *to do* agents). The near absence of pragmatic architectures that implement the notion of *to be* goals represents a gap that current research is trying to address.

2.3 AI Planning

Generic planning systems operate on problem descriptions that contain three main elements, a domain specification containing the operators available to the agent, a description of an initial state of the world and a description of a goal state of the world that the agent wishes to attain. A planning algorithm solves a problem by finding a sequence of instantiated operators that transform the world from the initial state to the goal state. This specification is used to generate the search-space over which the planning system searches for a solution. The search-space consists of all possible instantiations of the set of operators using the *Herbrand universe*[1] derived from the symbols contained in the initial and goal state specifications.

Initial approaches to general purpose planning include the *Stanford Research Institute Planning System* (STRIPS) [Fikes and Nilsson, 1971], whose notational concepts are still used as the basis for the specification of planning problems, as well as multiple approaches to *Partial Order Planning* (POP) [Ambros-Ingerson and Steel, 1988]. These approaches to generic planning were very limited in the size and type of problems that could be handled in reasonable time, due to their method of navigating the search-space. After a lull in new approaches to planning, several new algorithms were developed, such as Graphplan, SATPlan and HTN, representing a significant leap of efficiency allowing more complex planning problems to be computed in reasonable time.

A generic planner takes three inputs: a description of the initial state of the world; a description of the goal state that should be true after a plan is executed; and a description of the available operators, in some formal language (such as the language of STRIPS). The planner then tries to generate a sequence of actions that when applied to the initial state modifies the world so that the goal state becomes valid.

2.4 Meta-level Control

The widespread use of simple fixed rules for agent control is mainly concerned with maintaining agent reactivity in dynamic scenarios. This approach is usually associated with avoiding computationally expensive operations instead of allowing the risk of losing responsiveness. However, for an agent to operate in complex environments, it must have the means to decide on the tradeoff of computational cost and the worth of goals. This might involve the possibility of sacrificing smaller short-term goals to allow the achievement of larger, longer-term goals. It is clear that in this situation, the cost of executing some internal process can outweigh the cost of executing external actions, justifying a more complex mechanism for deciding how computing power must be spent, in a process which is known as *meta-level control* [Raja and Lesser, 2004]. In most agent architectures, it is often the case that behaviour selection occurs as a result of a set of fixed reactive rules, an approach that also applies to an agent's decision to adopt social behaviour to solve a problem. As a result, this type of agent is merely an abstraction for traditional software development, endowing the agent with no true autonomy. True autonomy involves decisions on the *meta-level*, that is, decisions regarding the reasoning process itself, which may lead the agent to spend resources *thinking* on a problem rather than acting immediately. We believe that meta-level control is a key component any such autonomy component, and that a suitable abstraction for a meta-level component is needed. In our architecture, the motivational model provide a valuation of the relative importance of certain goals, allowing the agent to decide which goals warrant a greater investment of processing power more effectively than fixed logic-based rules would allow.

[1] Any formal language with symbols for constants and functions has a Herbrand universe, which describes all of the terms that can be created by the application of all combinations of constant symbols as parameters to all functional symbols.

3 AgentSpeak-PL

Typically, agent interpreters select plans using more or less elaborate algorithms, but these seldom have any knowledge of the contents of the plans, so that plan selection is ultimately achieved using fixed rules, with an agent adopting *black box* plans based solely on the contextual information that accompanies them. Alternatively, some agent interpreters allow for plan modification rules to allow plans to be modified to suit the current situation [van Riemsdijk et al., 2003], but this approach still relies on a designer establishing a set of rules that considers all potentially necessary modifications for the agent to achieve its goals. The problem here is that for some domains, an agent description must either be extremely extensive (requiring a designer to foresee every possible situation the agent might find itself in), or will leave the agent unable to respond under certain conditions.

This *procedural* response to goal achievement has been favoured to enable the construction of practical systems that are usable in real-world applications. However, it also causes difficulties in cases of failure. When a procedural agent selects a plan to achieve a given goal the selected plan may fail, in which case the agent typically concludes that the goal has also failed, regardless of whether other plans to achieve the same goal might have been successful. By neglecting the *declarative* aspect of goals in not considering the construction of plans on-the-fly, agents lose the ability to reason about alternative means of achieving a goal, making it possible for poor plan selection to lead to an otherwise avoidable failure.

In this section, we describe how a procedural agent model can be modified to allow an agent to build new plans at runtime by chaining existing fine-grained plans from a plan library into high-level plans. We demonstrate the applicability of this approach through a modification to the AgentSpeak architecture, allowing for a combination of declarative and procedural aspects. This modification requires no change to the plan language, allowing designers to specify predefined procedures for known tasks under ideal circumstances, but also allowing the agent to form new plans when unforeseen situations arise. Though we demonstrate this technique for AgentSpeak, it can be easily applied to other agent architectures with an underlying procedural approach to reasoning, such as JADEX or the basic 3APL [Bordini et al., 2005a]. The key contribution is a method to augment an agent's runtime flexibility, allowing it to add to its plan library to respond to new situations without the need for the designer to specify all possible combinations of low-level operators in advance.

3.1 AgentSpeak

AgentSpeak [Rao, 1996] is an agent language that allows a designer to specify a set of procedural plans which are then selected by an interpreter to achieve the agent's design goals. It evolved from a series of procedural agent languages originally developed by Rao and Georgeff [Rao and Georgeff, 1995a]. In AgentSpeak an agent is defined by a set of beliefs and a set of plans, with each plan encoding a procedure that is assumed to bring about a desired state of affairs, as well as the context in which a plan is relevant. Goals in AgentSpeak are implicit, and plans intended to fulfil them are invoked whenever some triggering condition is met in a certain context, presumably the moment at which this implicit goal becomes relevant.

The control cycle of an AgentSpeak interpreter is driven by events on data structures, including the addition or deletion of goals and beliefs. These events are used as triggering conditions for the adoption of plans, so that adding an achievement goal means that an agent desires to fulfil that goal, and plans whose triggering condition includes that goal (*i.e.* are *relevant* to the goal) should lead to that goal being achieved. Moreover, a plan includes a logical condition that specifies when the plan is *applicable* in any given situation. Whenever a goal addition event is generated (as a result of the currently selected plan having subgoals), the interpreter searches the set of relevant plans for applicable plans; if one (or more) such plan is found, it is pushed onto an intention structure for execution. Elements in the intention structure are popped and handled by the interpreter. If the element is an action, this action is executed, while if the element is a goal, a new plan is added into the intention structure and processed. During this process, failures may take place either in the execution of actions, or during the processing of subplans. When such a failure takes place, the plan that is currently being processed also fails. Thus, if a plan selected for the achievement of a given goal fails, the default behaviour of an AgentSpeak agent is to conclude that the goal that caused the plan to be adopted is not achievable. This control cycle is illustrated in the diagram of

Figure 1,[2] and strongly couples plan execution to goal achievement.

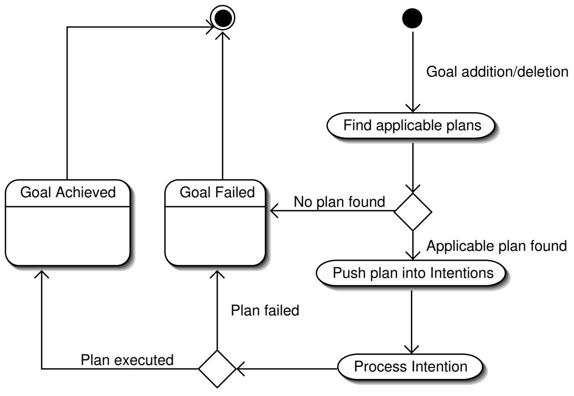

Figure 1: AgentSpeak control cycle.

The control cycle of Figure 1 allows for situations in which the poor selection of a plan leads to the failure of a goal that would otherwise be achievable through a different plan in the plan library. While such limitations can be mitigated through meta-level [Georgeff and Ingrand, 1989] constructs that allow goal addition events to cause the execution of applicable plans in sequence, and the goal to fail only when *all* plans fail, AgentSpeak still regards goal achievement as an implicit side-effect of a plan being executed successfully.

3.2 Planning in an AgentSpeak interpreter

In response to these limitations, we have created an extension of AgentSpeak that allows an agent to explicitly specify the world-state that should be achieved by the agent. In order to transform the world to meet the desired state, the agent uses a propositional planner to form high-level plans through the composition of plans already present in its plan library. This propositional planner is invoked by the agent through a regular AgentSpeak action, and therefore requires no change in the language definition. The only assumption we make is the existence of plans that abide by certain restrictions in order to be able to compose higher-level plans taking advantage of planning capabilities introduced in the interpreter. Whenever an agent needs to achieve a goal that involves planning, it uses a special planning action that converts the low-level procedural plans of AgentSpeak into STRIPS operators and invokes the planning module. If the planner succeeds in finding a plan, it is converted back into a high-level AgentSpeak plan and added to the intention structure for execution. Here, we liken the low-level procedural plans of AgentSpeak to STRIPS operators, connecting the agent interpreter to the planner by converting one formalism into the other and *vice versa*. We have chosen to use STRIPS as the planning language in this paper for simplicity reasons, and this approach would not lose applicability if one was to use PDDL [Fox and Long, 2003] (or another language) as the planning language.

3.2.1 The planning action

In order to describe the connection of the planning component with AgentSpeak, we need to review the main constructs of this agent language. As we have seen, an AgentSpeak interpreter is driven by events on the agent's data structures that may trigger the adoption of plans. Additions and deletions of goals and beliefs are represented by the plus $(+)$ and minus $(-)$ sign respectively. Goals are distinguished into *test goals* and *achievement goals*, denoted by a preceding question mark (?), or an exclamation mark (!), respectively. For example, the addition of a goal to achieve g would be represented by $+!g$. Belief additions and deletions arise as the agent perceives the environment, and are therefore outside its control, while goal additions and deletions only arise as part of the execution of an agent's plans.

[2]For a full description of AgentSpeak, refer to d'Inverno *et al.* [d'Inverno and Luck, 1998]

$$+goal\_conj(Goals) : true \leftarrow plan(Goals).$$

Table 1: Planner invocation plan.

In our approach, in addition to the traditional way of encoding goals for an AgentSpeak agent implicitly as triggering events consisting of achievement goals ($!goal$), we allow desires including multiple beliefs ($b_1, \ldots, b_n$) describing a desired world-state in the form $goal\_conj([b_1, \ldots, b_n])$. An agent desire description consists of a conjunction of beliefs the agent wishes to be true simultaneously at a given point in time. The execution of the planner component is triggered by an event $+goal\_conj([b_1, \ldots, b_n])$ as shown in Table 1.

Now, the key to our approach to planning in AgentSpeak is the introduction of a special *planning action*, denoted $plan(G)$, where G is a conjunction of desired goals. This action is bound to an implementation of a planning component, and allows all of the process regarding the conversion between formalisms to be encapsulated in the action implementation, making it completely transparent to the remainder of the interpreter.

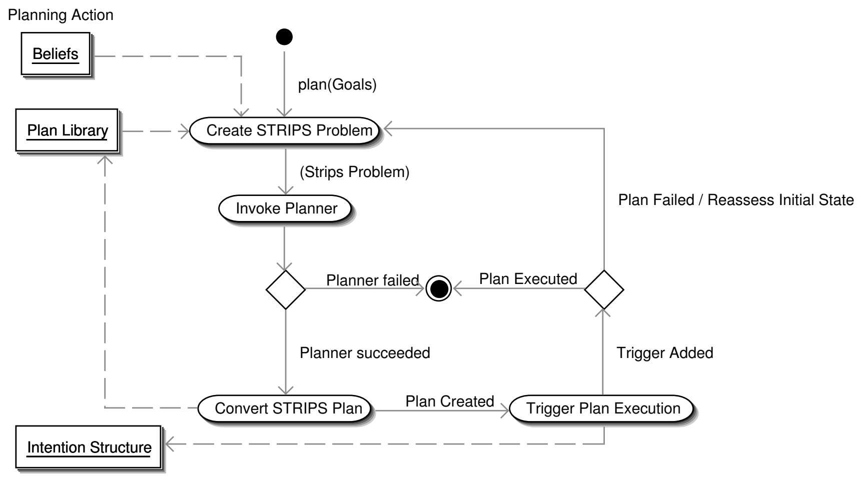

Figure 2: Operation of the planning action.

As illustrated in Figure 2, the internal action to plan takes as an argument the desired world-state, and uses this, along with the current belief database and the plan library, to generate a STRIPS [Fikes and Nilsson, 1971] planning problem. This action then invokes a planning algorithm; if a plan is found, the planning action succeeds, otherwise the planning action fails. If the action successfully yields a plan, it converts the resulting STRIPS plan into a new AgentSpeak plan to be added to the plan library, and immediately triggers the adoption of the new plan. If the newly created plan fails, the planner may then be invoked again to try and find another plan to achieve the desired state of affairs, taking into consideration any changes in the agent beliefs.

3.2.2 Chaining plans into higher-level plans

The design of a traditional AgentSpeak plan library follows a similar approach to programming in procedural languages, where a designer typically defines fine-grained actions to be the building blocks of more complex operations. These building blocks are then assembled into higher-level procedures to accomplish the main goals of a system. Analogously, an AgentSpeak designer traditionally creates fine-grained *plans* to be the building blocks of more complex operations, typically defining more than one plan to satisfy the same goal (*i.e.* sharing the same trigger condition), while specifying the situations in which it is applicable through the context part of each plan. Here, we are likening STRIPS actions to low-level AgentSpeak

$$\begin{aligned} +!move\_to(A,B) : {} & available(car) \\ & \leftarrow get(car); \\ & drive(A,B). \end{aligned}$$

$$\begin{aligned} +!move\_to(A,B) : {} & available(car) \\ & \leftarrow walk(A,B). \end{aligned}$$

Table 2: Movement plans.

plans, since the effects of primitive AgentSpeak actions are not explicitly defined in an agent description. For example, an agent that has to move around in a city could know many ways of going from one place to another depending on which vehicle is available to it, such as by walking or driving a car, as shown in Table 2.

Modelling STRIPS operators to be supplied to a planning algorithm is similar to the definition of these building-block procedures. In both cases, it is important that operators to be used sequentially *fit*. That is, the results from applying one operator should be compatible with the application of the possible subsequent operators, matching the effects of one operator to the preconditions of the next operator.

Once the building-block procedures are defined, higher-level operations must be defined to fulfil the broader goals of a system by combining these building blocks. In a traditional AgentSpeak plan library, higher-level plans to achieve broader goals contain a series of goals to be achieved by the lower-level operations. This construction of higher-level plans that make use of lower-level ones is analogous to the planning performed by a propositional planning system. By doing the *planning themselves*, *designers* must cope with every foreseeable situation the agent might find itself in, and generate higher-level plans combining lower-level tasks accordingly. Moreover, the designer must make sure that the subplans being used do not lead to conflicting situations. This is precisely the responsibility we intend to delegate to a STRIPS planner.

Plans resulting from propositional planning can then be converted into sequences of AgentSpeak achievement goals to comprise the body of new plans available within an agent's plan library. In this approach, an agent can still have high-level plans pre-defined by the designer, so that routine tasks can be handled exactly as intended. At the same time, if an unforeseen situation arises, the agent is flexible enough to find novel ways to solve problems, while augmenting the agent's plan library in the process.

Clearly, lower-level plans defined by the designer can (and often will) include the invocation of *atomic actions* intended to generate some effect on the environment. Since the effects of these actions are not usually explicitly specified in AgentSpeak (another example of reasoning delegated to the designer), an agent cannot reason about the consequences of these actions. When designing agents using our model, we expect designers to explicitly define the consequences of executing a given AgentSpeak plan in terms of belief additions and deletions in the plan body as well as atomic action invocations. The conversion process can then ignore atomic action invocations when generating a STRIPS specification.

3.2.3 Translating AgentSpeak into STRIPS

Once the need for planning is detected, the plan in Table 1 is invoked so that the agent can tap into a planner component. The process of linking an agent to a propositional planning algorithm includes converting an AgentSpeak plan library into propositional planning operators, declarative goals into goal-state specifications, and the agent beliefs into the initial-state specification for a planning problem. After the planner yields a solution, the ensuing STRIPS plan is translated into an AgentSpeak plan in which the operators resulting from the planning become subgoals. That is, the execution of each operator listed in the STRIPS plan is analogous to the insertion of the AgentSpeak plan that corresponded to that operator when the STRIPS problem was created.

Plans in AgentSpeak are represented by a header comprising a triggering condition and a context, as well as a body describing the steps the agent takes when a plan is selected for execution. If e is a triggering event, $b_1, \ldots, b_m$ are belief literals, and $h_1, \ldots, h_n$ are goals or actions, then $e : b_1 \& \ldots \& b_m \leftarrow$

$h_1; \ldots ; h_n$. is a plan. As an example, let us consider a triggering plan for accomplishing `!move(A,B)` corresponding to a movement from A to B, where:

- e is `!move(A,B);`
- `at(A) & not at(B)` are belief literals; and
- `-at(A); +at(B).` is the plan body, containing information about belief additions and deletions.

The plan is then as follows:

```
+!move(A,B) : at(A) & not at(B)
    <- -at(A);
       +at(B).
```

When this plan is executed, it results in the agent believing it is no longer in position A, and then believing it is in position B. For an agent to rationally want to move from A to B, it must believe it is at position A and not already at position B.

In the classical STRIPS notation, operators have four components: an identifier, a set of preconditions, a set of predicates to be added (add), and a set of predicates to be deleted (del). For example, the same `move` operator can be represented in STRIPS following the correspondence illustrated in Figure 3, in which we convert the AgentSpeak invocation condition into a STRIPS operator header, a context condition into an operator precondition, and the plan body is used to derive add and delete lists.

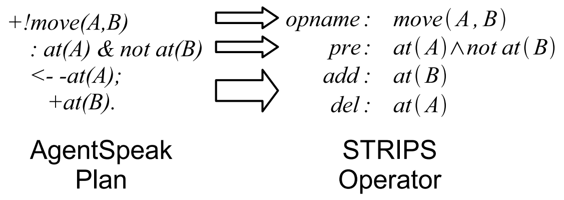

Figure 3: Correspondence between an AgentSpeak plan and a STRIPS operator.

A relationship between these two definitions is not hard to establish, and we define the following algorithm for converting AgentSpeak plans into STRIPS operators. Let e be a triggering event, $b_1\& \ldots \& b_m$ a conjunction of belief literals representing a plan's context, and $a_1, \ldots , a_n$ be belief addition actions and $d_1, \ldots , d_o$ be belief deletion actions within a plan's body. All of these elements can be represented in a single AgentSpeak plan. Moreover let $opname$ be the operator name and parameters, pre be the preconditions of the operator, add the predicate addition list and del the predicate deletion list. Mapping an AgentSpeak plan into STRIPS operators is accomplished as follows:

1. $opname = e$
2. $pre = b_1\& \ldots \& b_m$
3. $add = a_1, \ldots , a_n$
4. $del = d_1, \ldots , d_o$

In Section 3.2.1, we introduced the representation of a conjunction of desired goals as the predicate $goal\_conj([b_1, \ldots , b_n])$. The list $[b_1, \ldots , b_n]$ of desires is directly translated into the goal state of a STRIPS problem. Moreover, the initial state specification for a STRIPS problem is generated directly from the agent's belief database. Regarding the generation of an initial state-specification from the entire set of agent's beliefs, one may ponder about the frame problem [McCarthy and Hayes, 1969] and how it affects the performance of both the agent and the planning system. Here, an agent might be overwhelmed by the number of irrelevant beliefs that may arise from a highly complex environment, while most planning algorithms suffer severe performance loss as a result of a larger search space. In our architecture, the

$$+goal\_conj(Goals) : true \\ \leftarrow !op_1; \ldots ; !op_n.$$

Table 3: AgentSpeak plan generated from a STRIPS plan.

impact of the frame problem can be, in practice, mitigated. At the agent level, many implementations of AgentSpeak provide a filter over what perceptions are actually added to the belief base, and therefore, we can assume that the set of beliefs being submitted to the planning system is optimal. From the planning system perspective, many modern planning systems perform a pre-processing step in which irrelevant propositions are removed through the analysis of the operators, hence any beliefs that were not filtered out by the agent will be removed by this pre-processing step.

3.2.4 Executing generated plans

The STRIPS problem generated from the set of operators, initial state and goal state is then processed by a propositional planner. If the planner fails to generate a propositional plan for that conjunction of literals, the plan in Table 1 fails immediately and this goal is deemed unachievable, otherwise the resulting propositional plan is converted into an AgentSpeak plan and added to the intention structure.

A propositional plan from a STRIPS planner is in the form of a sequence $op_1, \ldots, op_n$ of operator names and instantiated parameters. We define a new AgentSpeak plan in Table 3, where $goal\_conj(Goals)$ is the event that initially caused the planner to be invoked.

Immediately after adding the new plan to the plan library, the event $goal\_conj(Goals)$ is reposted to the agent's intention structure, causing the generated plan to be executed. Plans generated in this fashion are admittedly simple, since the development of a complete process of plan generalisation is not a trivial matter since, for instance, it involves solving the issue of deriving the context condition adequately. An extremely simple solution for this problem uses the entire belief base of the agent as context for that plan, but this solution includes a great number of beliefs that are probably irrelevant to the goal at hand, severely limiting this plan's future applicability. Another solution involves replicating the preconditions of the first operator for the new plan, but this could also lead the agent to fail to execute the plan later on. We have developed an algorithm to derive a minimal set of preconditions, which we omit here due to space constraints, showing instead the simple solution of using a constantly true context. Another possible refinement to the conversion of a STRIPS plan into an AgentSpeak plan is to allow the same generated plan to be reused to handle side-effects of the set of goals that led to its generation.

3.3 Discussion

In this section, we have demonstrated how the addition of a planning component can augment the capabilities of a plan library-based agent. In order to exploit the planning capability, the agent uses a special planning action to create high-level plans by composing specially designed plans within an agent's plan library. This assumes no modification in the AgentSpeak language, and allows an agent to be defined so that *built-in* plans can still be defined for common tasks, while allowing for a degree of flexibility for the agent to act in unforseen situations. Our system can also be viewed as a way to extend the declarative goal semantics proposed by Hübner *et al.* [Hübner et al., 2006], in that it allows an agent designer to specify only desired world-states and basic capabilities, relying on the planning component to form plans at runtime. Even though the idea of translating BDI states into STRIPS problems is not new [Meneguzzi et al., 2004], our idea of an encapsulated planning action allows the usage of any other planning formalism sufficiently compatible with the BDI model.

The prototype implemented for the evaluation of the extensions described in this section has been empirically tested for a number of small problems, but, further testing and refinement of this prototype is still required, for instance, to evaluate how interactions between the addition of new plans will affect the existing plan library. The system can also be improved in a number of ways in order to better exploit the underlying planner component. For example, the effort spent on planning can be moderated by a quantitative model of control, so that an agent can decide to spend a set amount of computational effort into the planning process before it concludes the goal is not worth pursuing. This could be implemented by changing the definition of $goal\_conj(Goals)$ to include a representation of motivational model $goal\_conj(Goals, Motivation)$, which can be used to tune the planner and set hard limits to the amount of planning effort devoted to achieving that specific desire.

4 AgentSpeak-MPL

Considering the gap between the expected level of autonomy and what is currently possible in existing agent architectures, we believe that meta-level control is a key element in improving agent autonomy, and that a suitable abstraction for a meta-level component is needed. Inspired by the research on areas such as psychology and ethology, several researchers [Cañamero, 1997; Grand and Cliff, 1998; Luck et al., 2003; Norman et al., 2004] have proposed to address this gap by emulating motivated behaviour in animals. In this section, we review previous research on motivation aiming at using it as an abstraction for meta-level control, and describe a BDI architecture augmented with a motivated control module.

4.1 Definitions of motivation

Understood as the root cause of future-directed behaviour, motivation has been studied by researchers in a variety of areas, such as psychology [Morignot and Hayes-Roth, 1996], ethology [Balkenius, 1993] and philosophy [Mele, 2003]. A psychology-inspired definition of motivation considers it as representation of an individual's orientation towards particular classes of goals [Morignot and Hayes-Roth, 1996], while a philosophical definition from Mele [Mele, 2003] encompasses the concept of varying motivational strength linked to an agent's urgency in relation to adopting its associated goals. Mele [Mele, 2003] posits based on multiple sources that motivation is a trait present in animals that are capable of representing goals and means to goals. Both goals and means to goals may be influenced by motivation; that is, a motivation may influence both the adoption of a certain goal and the choice of the means to accomplish certain goals. Motivations vary in strength, and this variation dictates the agent's choice of behaviours to those associated with the strongest motivation, so that whenever an agent acts intentionally, its actions stem from underlying motivations.

From an ethological point of view, motivation is commonly associated with *drives* and *incentives* [Balkenius, 1993; Munroe et al., 2003]. In a simplified explanation, drives are internally generated states resulting from the violation of an animal's homeostasis, such as the deprivation of food or the excess of a given hormone. Incentives, on the other hand, are externally generated stimuli that increase certain motivations within the animal, such as in the presence of abundant food causing an animal to feed [Balkenius, 1993]. Motivations have also been described as giving rise to a continuum of appetitive behaviours (*i.e.* those that cause an agent to need something) leading to consummatory ones (*i.e.* those that satisfy this need). This means that some behaviours result in the build up of strength of certain motivations related to appetitive behaviour, and when a motivation has reached a high enough level, consummatory behaviours for the mitigation of this motivation are triggered.

The analysis of the motivational rewards of certain actions can also provide a mechanism to prevent certain undesirable behaviours from occurring simultaneously (also referred to as *lateral inhibition*) [Grand and Cliff, 1998], as in trying to look at a watch while holding a mug with the same hand. More generally, if one assumes that the consequences of any action can be measured as affecting motivations either positively or negatively, then this values can be used to determine which plans an agent can execute simultaneously without incurring detrimental interference among these plans.

The aspect of motivation most commonly sought to be captured by computational architectures is the continuous representation of priorities as a means to determine the *focus of attention* at any given time [Cañamero, 1997; Griffiths and Luck, 2003; Norman and Long, 1995]. This is important as it allows an agent with limited resources to concentrate its efforts on achieving goals that are relevant to it at specific moments, and to adapt such a concentration of effort to the current reality. Contrasting with the traditional process of goal selection based solely on environmental state, real biological systems often generate different plans of action under the same environment. Here, motivations can be modelled as a mechanism associated with internal *cues* that trigger goal generation in parallel with external factors [Munroe et al., 2003]. An internal cue can be seen as a trigger condition that, when activated, causes an agent to consider the adoption of a set of associated goals. It differs from the simple logical preconditions traditionally used in agent languages in that internal cues are a result of the dynamics of motivation strength, rather than a simple binary condition over the current state of the world.

4.2 A Motivated AgentSpeak Interpreter

Autonomous agents are expected to generate goals pro-actively instead of simply reacting to discrete events in the environment [Duff et al., 2006]. Generating goals pro-actively entails that an agent has a way of assessing its current situation and predicting how the environment (or other agents in the environment) will behave, in order to provide a rational justification for the adoption of a goal. Since motivations can be used to associate a measure of importance to goals, it is possible to use motivational intensity to guide an agent's choice of action when faced with multiple conflicting courses of action.

As our previous survey suggests [Meneguzzi, 2006],[3] models of motivation typically provide some kind of function that associates a motivational value representing intensity to world states and actions, which tell how important a certain state is and how to react to it. Agents can use expected motivational rewards to predict how much certain courses of action will affect its motivational state, allowing the agent to select plans more effectively. At a more concrete level, motivation intensity information can be used to refine several parts of the reasoning cycle in AgentSpeak, such as: goal selection, plan adoption and intention selection. We have, therefore, defined a series of motivation-based extensions for the various functions performed by an agent in its reasoning process.

4.2.1 Goals and Requirements

The addition of a model of motivations to underpin the generation of goals in autonomous agents provides a rational basis not only for the goals thus generated, but also for the subsequent selection of plans to fulfil these goals, and the actions carried out in the execution of the selected plans. This information-rich connection between key parts of the agent reasoning process can be exploited in the refinement of these processes to improve an agent's ability to interact with the environment as well as other agents. For example, explicit knowledge of what caused the adoption of a certain goal allows an agent to decide the best course of action to achieve it. When using motivational intensity thresholds as triggers for goal adoption, one can also use this quantitative information to compare concurrent goals and prioritise them in case of conflicts. Furthermore, taking into consideration the shortcomings of existing agent architectures regarding meta-level control, we have identified three requirements that our motivated architecture should fulfil, namely:

- there must be a language for abstracting meta-level control based on motivation;
- agents must be able to start behaviours even in the absence of a steady stream of events; and
- there must be a quantitative association between world-states and actions or plans, allowing the evaluation of alternative courses of action.

Abstract Motivation Functions .

From our previous survey [Meneguzzi, 2006] we have seen that several models of motivated control mechanisms have been developed, each of which focuses on improving a particular application of an agent system. These models share many commonalities, in particular regarding the *flow* of motivation dynamics, which generally consists of: *updating motivational levels* based on the current agent state and perceptions; *generating goals* as a result of this update; and *mitigating motivations* as a result of goal achievement. However, they differ on the specific strategies for motivation update, goal selection and motivation mitigation. With this in mind, it is necessary to design a generic motivation framework for this agent architecture,

[3]This survey is not included in this document for brevity, but it is available at `www.dcs.kcl.ac.uk/pg/meneguzzi`.

providing abstract functions for the three common elements we identified while allowing specific strategies to be used within these functions.

Flexible Integration with Agent Control .

Besides the particularities of the motivational models themselves, these efforts also explore different ways in which motivated control can be used to improve an agent. These improvements are targeted at specific parts of the agent reasoning process, such as the update of an agent's internal state based on perceptions, goal selection, or the prioritisation of adopted plans. Conveniently, the architecture of an idealised AgentSpeak interpreter [d'Inverno and Luck, 1998] is built around a set of abstract functions for many of these processes that can be refined with information from the motivational module. Thus, the design of our motivated architecture focuses on these abstract functions as the *points of contact* between motivational information and agent control, allowing the composition of control strategies based on the application of this information to specific parts of the reasoning process.

In turn, this arrangement of abstract functions enables us to evaluate the impact of using motivation information indifferent parts of the agent, so that the limitations of one refinement do not negate the advantages of another. For instance, it is possible to refine the plan selection function with an assessment of the motivational value of adopting a certain plan to achieve a goal, and it is also possible to use a similar assessment in the intention selection function to prioritise plans executing concurrently. Using our architecture, the outcome of using these refinements can be evaluated individually or in conjunction, providing a clearer picture of the applications of motivated control.

A basic model of motivations .

In order to create the motivation component for our experiments, we take the model of Griffiths *et al.* [Griffiths and Luck, 2003] as a base, since it fulfils the requirements set forth in Section 4.2.1. This model represents a motivation as a tuple $< m, i, t, f_i, f_g, f_m >$, where m is the name of the motivation, i is its current intensity, t is a threshold, f_i is an intensity update function, f_g is a goal generation function, and f_m is a mitigation function.

The model underpins the mBDI architecture [Griffiths and Luck, 2003], which in turn is based on the PRS/AgentSpeak architecture plus motivations. The reasoning cycle for an mBDI agent is illustrated in Algorithm 1.

Algorithm 1 mBDI control cycle.

```
1: loop
2:     perceive the environment and update the beliefs;
3:     for all motivation $m$ do
4:         apply $f_i$ to $m$ to update its intensity;
5:     end for
6:     for all motivation $m$ do
7:         apply $f_g$ to $m$ to generate new goals;
8:     end for
9:     select a plan for the most motivated of these new goals and adopt it as an intention;
10:    select the most motivationally valuable intention and perform the next step in its plan;
11:    on completion of an intention apply $f_m$ to each motivation to reduce its intensity;
12: end loop
```

The model of motivations used in the mBDI architecture has been created for a procedural agent architecture, as is apparent from Steps 9 and 11 of the control cycle in Algorithm 1, which describe an intention as a plan to be executed, and mitigation of a motivation is equated to the completion of that plan. Such an approach is not well suited to a declarative agent, in which goals are described as world-states to be achieved, since a plan may execute successfully and still fail to bring about the desired world-state.

```
Motivation processBay {
   Threshold = 10;

   IntensityUpdate MyIntensityUpdateFunction { ... }
   GoalGeneration  MyGoalGenerationFunction { ... }
   Mitigation      MyMitigationFunction { ... }
}
```

Table 4: High-level description of a motivation.

Besides this limitation, the mBDI architecture does not specify the behaviour of the goal generation function between threshold activation and goal mitigation. The effects of this ambiguity become apparent when there is a significant delay between goal adoption and goal achievement. For example, consider a nourishment motivation that generates a goal to feed whenever its threshold is reached, and for which all available plans take 3 units of time to be executed before the goal is achieved. Moreover, suppose that this agent performs one reasoning cycle per unit of time so; assuming the plan is successful, the agent will perform 3 reasoning cycles before mitigating this motivation. In the meantime, it is not clear whether or not the agent should generate the same goal 3 times until the motivation is mitigated or generate goals only once between a motivation's threshold being reached and its subsequent mitigation.

A Language of Motivation .

So far, we have described the abstract machinery that drives motivated control, following some of the requirements of Section 4.2.1. Therefore, it is necessary to associate these abstractions to concrete motivations. However, as we have seen in the literature reviewed in Section 4.1, different individuals can have particular sets of motivations, and consequently, be affected by their motivations in varying ways. We must assume then, that every agent can be driven by a unique set of motivations, each of which having a particular dynamics to allow the agent to evaluate its current situations and achieve its goals according to its own priorities.

In order to allow a designer to describe the motivational aspects individually for each agent, we must have a language that supports the description of unique sets of motivations based on the abstract functions and data structures of the mBDI model. Therefore, we have designed a language centred on the three abstract functions described in Section 4.2.1: intensity update; goal generation; and mitigation. Concrete versions of these functions are essentially mappings between beliefs and an intensity value in the case of intensity update and mitigation, or new goals for the goal generation function. These functions are specified for each individual motivation, of which the agent can have several.

At a high level, each motivation is composed of an identifier, a threshold, and the name of a concrete function to be used for each of the required abstract functions of our motivation model. These basic elements of a single motivation are shown in the excerpt of Table 4. Whenever the intensity of a motivation reaches the declared *threshold* as a result of the intensity update function, this motivation is said to be *activated* (following the terminology of *Alarms* [Norman and Long, 1995]), and the goal generation function is invoked, after which the mitigation function is invoked to verify if the condition for the motivation to be mitigated is reached. Within the declaration of each concrete function, details of the mapping process are described, so if we are dealing with an intensity update function, the mapping consists of belief-value correspondences, while if we are dealing with a goal generation function, the mapping is a series of belief-goal associations.

We consider each of these in detail below. The whole BNF of the language described in this Section is shown in Table 1, in which the logical framework has been derived from the BNF of the Jason parser [Bordini et al., 2005b]. We review the relevant constructs in more detail as we describe specific parts of the language.

Intensity Update and Mitigation Functions .

$$
\begin{aligned}
parse &::= (motivation)+\\
motivation &::=< MOTIVATION > identifier“\{”motivationBody“\}”\\
motivationBody &::= threshold“;”intensityUpdategoalGenerationmitigation\\
threshold &::=< THRESHOLD > “=” < NUMBER >\\
identifier &::=< ATOM >\\
&\mid < VAR >\\
classname &::= identifier\\
intensityUpdate &::=< INTENSITY\_UPDATE > classname“\{”\\
&(beliefToIntegerMapping“;”) * “\}”\\
beliefToIntegerMapping &::= (log\_expr“->”arithm\_expr)\\
goalGeneration &::=< GOAL\_GENERATION > classname“\{”\\
&(beliefToTriggerMapping“;”) * “\}”\\
beliefToTriggerMapping &::= (log\_expr“->”trigger)\\
mitigation &::=< MITIGATION > classname“\{”\\
&(beliefToIntegerMapping”;”) * “\}”\\
trigger &::= (“+”|“-”)((“!”|“?”))?(literal|var)\\
literal &::= (((< TK\_NEG >)?atom)| < TK\_TRUE > | < TK\_FALSE >)\\
atom &::=< ATOM > (“(”terms“)”)?(list)?\\
terms &::= term(“,”term)*\\
term &::= (literal|list|arithm_expr|string)\\
list &::= “[”(term(“,”term)*\\
&(“|”(< VAR > | < UNNAMEDVAR > |list))?)?“]”\\
log\_expr &::= log\_expr\_trm(“|”log\_expr)?\\
log\_expr\_trm &::= log\_expr\_factor(“\&”log\_expr\_trm)?\\
log\_expr\_factor &::= (< TK\_NOT > log\_expr\_factor|rel\_expr)\\
rel\_expr &::= (arithm\_expr|literal|string)\\
&((“<”|“<=”|“>”|“>=”|“==”|“\backslash\backslash ==”|“=”|“=..”)\\
&(arithm\_expr|literal|string|list))?\\
arithm\_expr &::= arithm\_expr\_trm((“+”|“-”)arithm\_expr)?\\
arithm\_expr\_trm &::= arithm\_expr\_factor(\\
&(“*”|“/”| < TK\_INTDIV > | < TK\_INTMOD >)arithm\_expr\_trm)?\\
arithm\_expr\_factor &::= arithm\_expr\_simple((“**”)arithm\_expr\_factor)?\\
arithm\_expr\_simple &::= (< NUMBER > |“-”arithm\_expr\_simple|“(”log_expr“)”|var)\\
var &::= (< VAR > | < UNNAMEDVAR >)(list)?\\
string &::=< STRING >
\end{aligned}
$$

Table 1: BNF of the motivation language.

```
Motivation feed {
   ...

   IntensityUpdate MyIntensityUpdateFunction {
      hungry & near(food) -> 2;    //This increases intensity
      not hungry -> -1;            //This lowers it a bit
   }

   ...
}
```

Table 5: Example of an intensity update function.

```
Motivation processBay1 {
   ...

   GoalGeneration MyGoalGenerationFunction {
      near(food) -> +!eat(food);
      //true -> +!eat(food); //Another possibility
   }

   ...
}
```

Table 6: Example of a goal generation function.

As we have seen, the functions for updating the intensity of, and mitigating, a motivation need to provide some kind of mapping between perceptual data and an intensity variation. As a result, our *language of motivation* allows the specification of a mapping between beliefs and an arithmetic expression expressing how the intensity level should be modified when the specified beliefs are true. Any specific mapping is represented as $log\_expr- > arithm\_expr$, as shown in Table 1.

An example of such a mapping is shown in Table 5. In this example, the intensity of the motivation to feed is increased by 2 points whenever the agent is hungry and believes food is nearby. It is important to notice that this language deals exclusively with beliefs, both intrinsic ones and those resulting from perception, whereas some motivation models assign values to actions and by doing so conform to a procedural view of reasoning. The mitigation function provides a mapping that is syntactically the same as for the intensity update function but, according to our model of motivations in Section 4.2.1, this function is only invoked when an intention is adopted to satisfy its associated motivation.

Goal Generation .

Aside from mapping beliefs into perceptions, we must also describe the mapping of beliefs into goals. Since the goal generation function is only invoked when the motivation threshold is exceeded as a result of intensity accumulation, our language allows the specification of additional constraints before a goal is generated, or the unconditional generation of goals through the *true* condition. The general form of goal generation functions is illustrated in Table 1, similar to the intensity update function previously described, mappings in the goal generation function start from a logical expression over beliefs, however the target of this mapping are new goals to be achieved as a result of the intensity reaching the threshold in the motivation containing this goal generation function.

This is illustrated in the example of Table 6, below. In this example, the agent generates an event to eat the food located nearby whenever the goal generation function of Table 6 is invoked.

Example .

A complete example of a motivation described using our language is shown in Table 7. This example describes the dynamics of a motivation to feed whenever the agent is sufficiently hungry and close to a

```
Motivation feed {
   Threshold = 10;                  //This is the threshold
   IntensityUpdate MyIntensityUpdateFunction {
      hungry & near(food) -> 2;     //This increases intensity
      not hungry          -> -1;    //This lowers it a bit
   }

   GoalGeneration MyGoalGenerationFunction {
      near(food) -> +!eat(food);
      //true     -> +!eat(food);    //Another possibility
   }

   Mitigation MyMitigationFunction {
      ate(food) -> -20;
   }
}
```

Table 7: Description of a motivation to feed.

```
Motivation avoidPredators {
   Threshold = 10;
   IntensityUpdate MyIntensityUpdateFunction {
      near(predator)   -> 5;        //This increases intensity
      seenBy(predator) -> 10;
      far(predator)    -> -5;       //This lowers it a bit
   }

   GoalGeneration MyGoalGenerationFunction {
      near(predator) -> +!fleeFrom(predator);
   }

   Mitigation MyMitigationFunction {
      far(predator) -> -10;
   }
}
```

Table 8: Description of a motivation to avoid predators.

source of food. The motivational intensity will start to increase as soon as the agent becomes hungry and detects food nearby, until it reaches the threshold of 10. Once the threshold is reached, the goal generation function will add a goal to eat this food. Finally, the agent assumes the motivation is mitigated when it perceives it has eaten the food, diminishing the motivational intensity accordingly.

In this simple example, the goal to eat the food is not activated immediately upon the perception of hunger and food proximity. In an agent with a single isolated motivation, describing goal adoption in terms of motivated reasoning is not significantly more flexible than what could be described directly in AgentSpeak, for example. However, when multiple concurrent behaviours are present, motivational information can be used to identify priorities and gives a rational underpinning for *not adopting* certain goals immediately. If we consider a new motivation to avoid predators, described in Table 8, in which motivational intensity is increased by the proximity of and detection by a predator, the adoption of a goal to eat food might not be rational if it means approaching a predator. In an agent with trigger-activated behaviours, this conflict must be solved by a designer before the agent is deployed in one of two ways: enumerating all possible conflicts and include them in a guard condition, or dropping (or failing) all conflicting goals whenever a higher priority goal is adopted. Both solutions have limitations, as trying to describe guard conditions for a large number of potentially conflicting goals can be challenging, while dropping goals may not be acceptable in certain situations.

4.2.2 Integration with AgentSpeak

In traditional AgentSpeak, plans are adopted as a reaction to events in the environment in a direct sense. That is, plans are expressed so that if event e happens in a certain world state, the agent will always adopt a plan matching that event. Furthermore, since goals in the procedural sense used by AgentSpeak(L) are adopted as part of the execution of plans, the agent does not generate them through deliberation, and they are instead adopted in the process of reacting to some event in the environment. For instance, a plan may be described so that whenever an agent believes that a given block is on a table (*e.g.* `on(block,table)`), a procedure to remove such a block is invoked. This amounts to simple reaction rather than autonomous behaviour. Furthermore, this method of *behaviour selection* fails to properly describe the reasons for goal adoption in a declarative sense. Using the same example of a block on a table, a declarative goal to remove the block from the table could be described as **`not`** `on(block,table)`. The question here is, whether the agent should *always* react to new events and start deliberation immediately even if the agent might be pursuing other more important goals.

Besides using the motivational model to generate goals, it is possible to investigate how motivation information can be used to refine and improve other parts of the agent reasoning cycle. Since a motivated agent acts to mitigate its motivations, the selection of plans to achieve these goals can be optimised by selecting plans that more effectively mitigate their underlying motivations. Therefore, we are currently investigating two modules that take advantage of motivational information for both plan and intention selection, as well as an analysing of the results of using these modules in a prototype agent.

4.3 Discussion

We have defined a simple language of motivations that allows a designer to create agents whose behaviour is activated by a more elaborate mechanism than the triggering mechanisms often used in current agent architectures. This language is supported by an extensible meta-level control module that uses the motivation abstraction as a metric for the agent to evaluate its future behaviour rather than inflexibly reacting to specific events in the environment. Reasoning about motivations serves not only to add flexibility (and thus autonomy) to an agent, but the analysis of future behaviour is crucial for an agent acting to accomplish a declarative goal, since any course of action that achieves such a goal would be acceptable, even if it jeopardises other goals in the process.

By departing from a fixed set of trigger-activated behaviours, we are moving towards agents which are truly autonomous, since they are able to reason about their courses of action at runtime instead of just obeying predefined rules. When operating in a multi-agent scenario, this type of agent might be subjected to a body of norms that regulate the agent society [Dignum, 1999], and a truly autonomous agent may decide that it is worthwhile to violate certain norms to achieve very important goals. In this setting, a motivational component might be used to reason about not only the concrete outcome of certain behaviours, but also the implications for the agent in the context of a normative system.

5 Research Plan

This section outlines the main set of issues we intend to focus our research for the next 9-month period, as well as summarising the results achieved so far. Moreover, we outline a strategy to carry out this research, including the activities we intend to execute as well as their associated deliverables; these activities are then organised in a work plan shown in Table 2.

5.1 Results

Two papers have already been published from this work, detailing the addition of a planning component in BDI architectures in general [Meneguzzi et al., 2007], as well as the extended AgentSpeak described in Section 3 [Meneguzzi and Luck, 2007]. Besides these papers, we have three thesis chapters drafted.

5.2 Workplan: Activities and Deliverables

- **Activity:** Motivated agency
  Description: Continuing the work on motivated agents started on the previous year, we need to develop a scenario in which meta-level control is required, in order to conduct empirical evaluation of the motivated architecture created previously.
  Deliverables: A description of the scenario, to be included in the corresponding thesis chapter.

- **Activity:** Motivation and partner selection.
  Description: Since the proposed research aims to introduce motivations as a mechanism for meta-level control, which includes making decisions regarding the necessity of external assistance, it is necessary to study the existing literature regarding partner selection to modify the motivational model and allow it to effectively exercise control over task delegation. Since many approaches to partner selection already exist in the literature, we intend to survey existing technologies and refine an existing method to take advantage of motivational information.
  Deliverables: The modified architecture, which should include a partner selection mechanism that takes advantage of the motivational model. The description of this architecture, as well as tests and results should form the basis of a thesis chapter, and possible a paper describing a decision procedure for partner selection and task delegation.

- **Activity:** Multiagent planning and coordination.
  Description: As our motivation-based architecture is refined for multi-agent operation, we need to integrate the planning mechanism introduced previously in our work. This involves investigating how to consider commitment in terms of a planning agent, and how to allow multiple planning agents to reach an agreement over joint plans. Moreover, considering shortcomings in the planning prototype when dealing with the simultaneous execution of plans, it is necessary to investigate methods for dealing with concurrent plans, aiming to allow our agent architecture to cope with multiple interacting agents. Depending on time and results, we will consider the possibility of examining how norms may be included in this framework.
  Deliverables: A report or paper outlining a simple agent architecture using the studied concepts. Possibly a prototype including one of the methods investigated.

5.3 Provisional Table of Contents

Period	Activity	Duration	Status
	2007		
	Motivated Agency		
January	- Draft chapter on motivated architecture	4 weeks	Complete
February	- Write upgrade report	2 weeks	Not Complete
March	- Design and run tests for the motivated architecture	2 weeks	Not Complete
	- Submit a paper	2 weeks	Not Complete
	Motivation and partner selection		
April	- Survey existing approaches	4 weeks	Not Complete
May	- Possibility of refinements based on motivation information	4 weeks	Not Complete
June	- Modify motivated architecture for multi-agent scenarios	2 weeks	Not Complete
July	- Design and run tests for the modified architecture	4 weeks	Not Complete
August	- Draft chapter on modified motivated architecture	2 weeks	Not Complete
September	- Submit a paper about the modified motivated architecture	2 weeks	Not Complete
	Multi-agent planning and coordination		
October	- Study existing technology	4 weeks	Not Complete
November	- Integration of multi-agent planning to motivated architecture	4 weeks	Not Complete
December	- Draft chapter on modified planning architecture	4 weeks	Not Complete
	2008		
January			
February			
March			
April	Tentative date for starting thesis write up	48 weeks	Not Complete
May			
...			

Table 2: Workplan

References

Ambros-Ingerson, J. A. and Steel, S. (1988). Integrating planning, execution and monitoring. In *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 83–88, St Paul, MN. American Association for Artificial Intelligence.

Balkenius, C. (1993). The roots of motivation. In *Proceedings of the 2nd International Conference on Simulation of Adaptive Behavior*. MIT Press.

Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E. (2005a). *Multi-Agent Programming: Languages, Platforms and Applications*, volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*. Springer.

Bordini, R. H., Hübner, J. F., and Vieira, R. (2005b). Jason and the golden fleece of agent-oriented programming. In Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors, *Multi-Agent Programming: Languages, Platforms and Applications*, pages 3–37. Springer.

Bratman, M. E. (1987). *Intention, Plans and Practical Reason*. Harvard University Press, Cambridge, MA.

Braubach, L., Pokahr, A., Lamersdorf, W., and Moldt, D. (2004). Goal representation for BDI agent systems. In Bordini, R. H., Dastani, M., Dix, J., and Fallah-Seghrouchni, A. E., editors, *Proceedings of the 2nd International Workshop on Programming Multiagent Systems Languages and tools (PROMAS 2004)*, pages 7–9.

Cañamero, D. (1997). Modeling motivations and emotions as a basis for intelligent behavior. In *AGENTS '97: Proceedings of the first international conference on Autonomous agents*, pages 148–155, New York, NY, USA. ACM Press.

Cohen, P. R. and Levesque, H. J. (1990). Intention is choice with commitment. *Artificial Intelligence*, 42(2-3):213–261.

Dignum, F. (1999). Autonomous agents with norms. *Artificial Intelligence and Law*, 7(1):69–79.

d'Inverno, M. and Luck, M. (1998). Engineering AgentSpeak(L): A formal computational model. *Journal of Logic and Computation*, 8(3):233–260.

d'Inverno, M. and Luck, M. (2004). *Understanding Agent Systems*. Springer Series on Agent Technology. Springer Verlag, Berlin, 2nd edition.

Duff, S., Harland, J., and Thangarajah, J. (2006). On proactivity and maintenance goals. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1033–1040, New York, NY, USA. ACM Press.

Fikes, R. and Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208.

Fox, M. and Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20:61–124.

Georgeff, M. P. and Ingrand, F. F. (1989). Monitoring and control of spacecraft systems using procedural reasoning. In *Proceedings of the Space Operations and Robotics Workshop*, Houston, USA.

Grand, S. and Cliff, D. (1998). Creatures: Entertainment software agents with artificial life. *Autonomous Agents and Multi-Agent Systems*, 1(1):39–57.

Griffiths, N. and Luck, M. (2003). Coalition formation through motivation and trust. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 17–24, New York, NY, USA. ACM Press.

Hübner, J., Bordini, R. H., and Wooldridge, M. (2006). Programming declarative goals using plan patterns. In *Proceedings of the 2006 Workshop on Declarative Agent Languages and Technologies*.

Jennings, N. R. (2000). On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296.

Luck, M., Munroe, S., and d'Inverno, M. (2003). *Autonomy: Variable and Generative*, chapter Chapter 2, pages 9–22. Kluwer.

McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502.

Mele, A. R. (2003). *Motivation and Agency*. Oxford University Press.

Meneguzzi, F. and Luck, M. (2007). Composing high-level plans for declarative agent programming. In *Proceedings of the 5th International Workshop on Declarative Agent Languages and Technologies (DALT 2007)*.

Meneguzzi, F. R. (2006). Motivated declarative agents in multiagent domains: Open issues. Progress report, University of Southampton.

Meneguzzi, F. R., Zorzo, A. F., da Costa Mora, M., and Luck, M. (2007). Incorporating planning into bdi agents. *Scalable Computing: Practice and Experience*, 1.

Meneguzzi, F. R., Zorzo, A. F., and Móra, M. D. C. (2004). Propositional planning in BDI agents. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 58–63, Nicosia, Cyprus. ACM Press.

Morignot, P. and Hayes-Roth, B. (1996). Motivated agents. Technical report, Knowledge Systems Laboratory – Stanford University.

Müller, J. P. (1996). The design of intelligent agents: A layered approach. In *The Design of Intelligent Agents: A Layered Approach*, volume 1177 of *Lecture Notes in Computer Science*. Springer Verlag, Germany.

Munroe, S. J., Luck, M., and d'Inverno, M. (2003). Towards motivation-based decisions for worth goals. In *Proceedings of Multi-Agent Systems and Applications III, Proceedings of the 3rd International Central and European Conference on Multi-Agent Systems*, pages 17–28.

Norman, T. J. and Long, D. (1995). Alarms: An implementation of motivated agency. In *ATAL*, pages 219–234.

Norman, T. J., Preece, A., Chalmers, S., Jennings, N. R., Luck, M., Dang, V. D., Nguyen, T. D., Deora, V., Shao, J., Gray, W. A., and Fiddian, N. J. (2004). Agent-based formation of virtual organisations. *Knowledge-Based Systems*, 17(2-4):103–111.

Raja, A. and Lesser, V. (2004). Meta-level reasoning in deliberative agents. In *IAT '04: Proceedings of the Intelligent Agent Technology, IEEE/WIC/ACM International Conference on (IAT'04)*, pages 141–147, Washington, DC, USA. IEEE Computer Society.

Rao, A. S. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In de Velde, W. V. and Perram, J. W., editors, *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume 1038 of *LNCS*, pages 42–55. Springer, Eindhoven, The Netherlands.

Rao, A. S. and Georgeff, M. P. (1995a). BDI-agents: from theory to practice. In *Proceedings of the First International Conference on Multiagent Systems ICMAS-95*, pages 312–319, San Francisco.

Rao, A. S. and Georgeff, M. P. (1995b). Formal models and decision procedures for multi-agent systems. Technical Report 61, Australian Artificial Intelligence Institute, 171 La Trobe Street, Melbourne, Australia. Technical Note.

van Riemsdijk, B., van der Hoek, W., and Meyer, J.-J. C. (2003). Agent programming in dribble: from beliefs to goals using plans. In *AAMAS '03: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 393–400, Melbourne, Australia. ACM Press.

van Riemsdijk, M. B., Dastani, M., and Meyer, J.-J. C. (2005). Semantics of declarative goals in agent programming. In *AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 133–140, Utrecht, The Netherlands. ACM Press.

Winikoff, M., Padgham, L., Harland, J., and Thangarajah, J. (2002). Declarative & Procedural Goals in Intelligent Agent Systems. In Fensel, D., Giunchiglia, F., McGuinness, D. L., and Williams, M.-A., editors, *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pages 470–481, Toulouse, France. Morgan Kaufmann.

Wooldridge, M. (2000). *Reasoning about Rational Agents*. The MIT Press.