Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

USER UNDERSTANDING OF COGNITIVE PROCESSES IN SIMULATION: A TOOL FOR EXPLORING AND MODIFYING

David Scerri
Sarah Hickmott
Lin Padgham

School of Computer Science and Information Technology
RMIT University
GPO Box 2476
Melbourne 3001, VIC, AUSTRALIA

ABSTRACT

Agent based simulations often model humans and increasingly it is necessary to do this at an appropriate level of complexity. It has been suggested that the Belief Desire Intention (BDI) paradigm is suitable for modeling the cognitive processes of agents representing (some of) the humans in an agent based modeling simulation. This approach models agents as having goals, and reacting to events, with high level plans, or plan types, that are gradually refined as situations unfold. This is an intuitive approach for modeling human cognitive processes. However, it is important that users can understand, verify and even contribute to the model being used. We describe a tool that can be used to explore, understand and modify, the BDI model of an agent's cognitive processes within a simulation. The tool is interactive and allows users to explore options available (and not available) at a particular agent decision point.

1 INTRODUCTION

Developing an Agent Based Model for social simulation requires the modeler to capture people's behaviors. In Padgham et al. (2011) it was argued that simple rules, typical of many agent based modeling platforms such as Repast (North et al. 2006) are inadequate, or at least inconvenient, for modeling the decision making of humans, who often operate using abstract plans over multiple time steps. This work described the integration of the JACK Belief Desire Intention (BDI) agent platform (Winikoff 2005) with Repast, to allow for easier modeling of human decision making processes.

Also, it is becoming widely accepted that to make social simulations effective, with respect to their particular intended use, then end users, stakeholders and/or domain experts, need to be involved in the model specification, design, testing and use (see e.g. Ramanath and Gilbert (2004)). Who should be involved and how depends on the intended use of the simulation, which may range from education to social research exploration to decision support. This paper examines how – in the context of using the BDI paradigm to specify agents within an ABM simulation – to give users the possibility to *understand* the cognitive processes of an agent in the simulation, *interact* with these processes during a simulation, and also to help *specify* what those cognitive processes might be.

Agent based simulations can be used for a range of different purposes around gaining greater understanding of complex situations. We have done some work (and developed a prototype simulation) around evacuation in response to a bushfire (i.e., forest fire or wildfire). In exploring with stakeholders how the tool we have developed may be further refined and used, one aspect which stands out is the need to explore and potentially interact with a representation of the cognitive processes (plans, goals and decisions) of the agent. If the simulation was to be used in community awareness building, we have been told it will be necessary for an individual to identify "their" representative agent in the simulation visualization, and also

978-1-4673-4781-5/12/$31.00 ©2012 IEEE

to examine and have some control over this agent, in order to internalize the knowledge and understanding arising from the simulation.

Similarly, if incident controllers, or others responsible for planning and acting during an emergency, are to be able to trust a simulation sufficiently to learn from it, they must be able to gain some insight and understanding of the modeling, including the modeling of the agent's cognitive processes. Finally, the knowledge of domain experts could more directly (and arguably more accurately) be incorporated into a model using a tool that allows expert users to interact with a simulation in order to see and specify important aspects of agents cognitive processes.

There do exist tools, e.g., Prometheus Design Tool, PDT (Padgham, Thangarajah, and Winikoff 2005) which can provide a view of the structure of plans and goals which enable a user to gain some understanding of the cognitive process represented in the simulation. However, as a relatively compact structure can represent thousands or even millions of different possible ways for something to be accomplished (Padgham and Winikoff 2004), given different situations, it is often necessary or at least helpful to be able to follow the decisions made at various steps, and the reasons for these decisions. By allowing a user to control interactively the decision made at various points in a simulation, a user can also explore the effects of different decision making strategies, in a range of potential scenarios. This can provide a better understanding of the underlying model, leading to a deeper and more informed interpretation of results, as well as potentially better preparation for actual scenarios that arise.

In this paper we describe a tool that we have built, which allows the interrogation (and possible modification) of the cognitive decision making of an agent which is part of a simulation incorporating both cognitive and reactive agents as described in Padgham et al. (2011). The tool currently uses cognitive agents built with JACK (Winikoff 2005), and designed using PDT (Padgham, Thangarajah, and Winikoff 2005) a design tool for BDI agents, in a simulation environment which combines JACK and Repast Simphony as described in Padgham et al. (2011). Our intent however is to incorporate the tool as part of a general purpose BDI plug-in for Repast Simphony. In using the tool, a user identifies at the start of a simulation, which agent decision points they wish to explore. The simulation is then stopped at these points, and the user can explore aspects of the cognitive state (and environment state via standard Repast Simphony tools), to better understand why the agent is behaving as it is. The user can also make decisions at these points for the specific agent being observed, thus influencing the continuation of the simulation. Moreover, they can make permanent modifications, or suggestions for modifications, to the underlying agent decision making structure, thus participating in the ongoing development of a simulation application. Some such modifications can be made automatically and are incorporated as soon as the program is recompiled. Others require a programmer/modeler to make the necessary changes to the code.

In summary, there are three levels at which a user can interact with the tool:

- to explore the way that a simulation plays out, stopping at points of interest and exploring both system state and agent cognitive structures to understand what decisions an agent is taking and why;
- to interactively control a simulation to observe the effects of certain decisions, which are not those which the agent is necessarily programmed to take. This immersion via controlling of the agent can be powerful for learning (Shute et al. 2009), as well as providing broader understanding; and
- to participate in the development of the simulation by providing expert advice as to what decisions should be made by an agent (type) in various situations, including which plans would most likely be chosen under which conditions, by the humans being modeled.

In the next section we briefly describe BDI representation, and introduce an example we will use throughout to illustrate the tool. In Section 3 we describe the interface to the tool and its different display panes. Following this we describe how the tool is used in the three different ways discussed above: understanding what the system is doing (from the perspective of the decisions of a particular agent); controlling decisions made by an agent to understand what effect different choices would have on how a

scenario plays out; and to contribute to more accurate modeling of the decisions of certain types of agents in the simulation, as understood by either a representative of the group being modeled, or a stakeholder who has a detailed understanding of the behavior to be represented. Finally we describe some difficulties and issues that require further work in order to fully realize the potential of this tool.

2 BDI REPRESENTATION

BDI agents are agents that use mental constructs such as beliefs, desires, plans and commitments, in their internal representations, e.g., (Cohen and Levesque 1990; Rao and Georgeff 1995) and are typically programmed using an agent programming language or agent development platform (e.g., JACK (Winikoff 2005), JASON (Bordini, Hübner, and Wooldridge 2007), Jadex (Pokahr, Braubach, and Lamersdorf 2005)). These agent programs consist primarily of a library of *plans* which describe, often at an abstract level, how to achieve a particular *goal* (or alternatively respond to an *event* for more reactive behavior), given some situational *context*, which is evaluated according to knowledge (or beliefs) about the *current state*.

Goals and events: Goals and events are the driving force behind BDI agent programs. Agent goals cause the agent to pro-actively select plans that will achieve their goals, and if one plan fails, they will attempt to find and execute an alternative plan. Important events (those the agent is programmed to respond to) similarly cause the agent to select a plan to respond to the event. Events may also lead to the agent updating its beliefs about the world.

Plans: Plans are procedural descriptions or recipes for how to act. Often they are at an abstract level, which is gradually refined as the agent makes choices as to how to accomplish the various plan steps. Plan steps are either *actions* which are the low-level behaviors of the agent, or sub-goals which are themselves associated with plans. We say a plan is *relevant* to a goal or event if it is specified as a plan for achieving that goal. (Plans may be pro-active for achieving a goal, or reactive for reacting to an event. We will in future include reacting to an event when we discuss achieving a goal, as the representations and mechanisms are the same.)

Plan types and instances: The plans that are in the agent's plan library are actually plan types. Before they can be used they must be *instantiated*; that is, any variables must be bound. For example a plan to take a flight may involve driving to the airport, checking in, boarding the plane, and so on. Before this plan can be used in practice, the flight must be bound to a particular flight, leaving from a specified airport at a certain time. Any variables that need to be bound in order to instantiate a plan are mentioned in what is called the plan's *context condition*. If there are choices of bindings for some variables, a plan may have multiple possible instances. If a plan's variables cannot all be bound satisfactorily, then the plan has no instance available in the current situation.

Context conditions: A context condition of a plan has two purposes. It specifies the variables that need to be bound, and any conditions relating to those variables, and it also specifies conditions that must be true in order for the plan to be appropriate for the situation. The context condition is generally a logical formula, which must evaluate to "true" for the plan to be *applicable*, or usable, in the current situation. A context condition is generally a conjunction of clauses, but may also contain disjunctions. An example of a context condition that could give multiple (or no) instances is: `[neighbor($N) AND has-car($N)]` as there may be multiple neighbors who meet this criterion. If there are no values of `$N` which make this clause `True`, then the plan is not currently applicable.

Decisions, intentions and execution: Each time the agent arrives at a goal it must evaluate which of the plans are applicable, and select one for execution. This selection may be based on a fixed priority or preference (a priority number associated with the plan (type), that ranks it in relation to other applicable plans), or it may be calculated by a procedure often referred to as a *meta-plan* which reasons about which

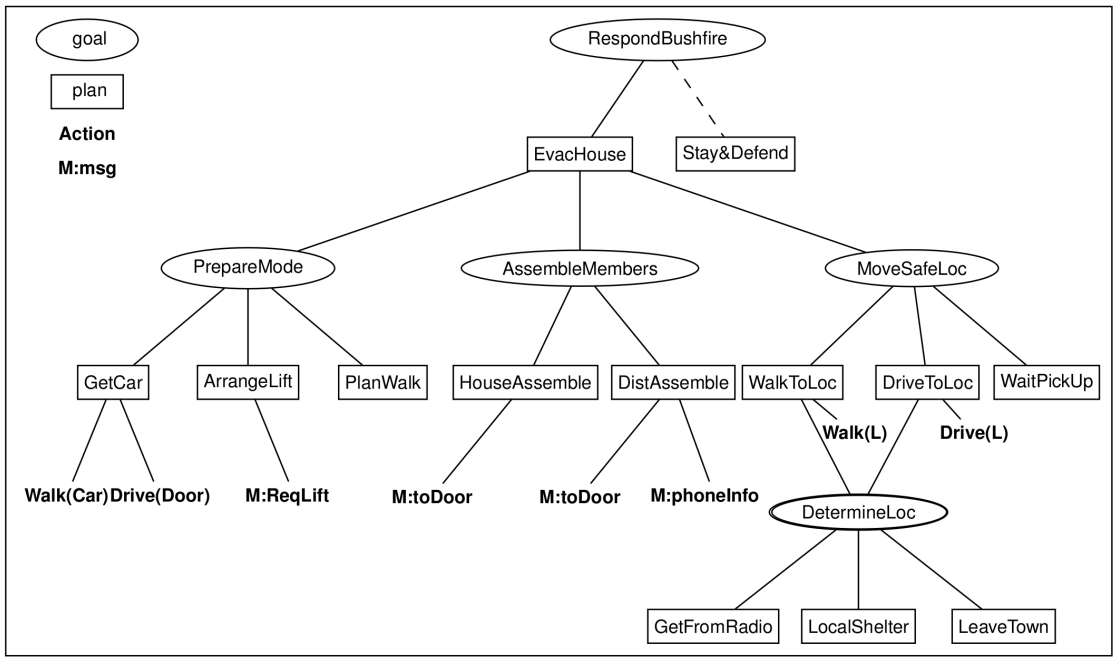

Figure 1: Example goal-plan hierarchy: Bushfire response.

of the applicable plans to prefer, based on the beliefs about the world. Beliefs about the world are thus used in two ways: to determine whether a plan is applicable - i.e., usable in the current situation; and to reason about which applicable plan might be preferred over another, given the current situation. Once an applicable plan is selected to achieve a particular goal (or sub-goal) it is placed into an *intention* structure, which is executed by either doing the next action specified in the plan, or selecting a plan to achieve the next sub-goal specified in the plan. As a plan is being executed, it may be the case that it *fails* at some point. This can happen because an action fails to achieve what is intended, or it can happen because there is no applicable plan available for achieving the current sub-goal, given the state of the world. When a plan fails, the agent will reconsider which plans are applicable and will, if possible, select an alternative plan for achieving the sub-goal.

2.1 Example

The set of plans is stored in the agent's plan library and can be represented as a set of and-or hierarchies of plans and goals as shown in Figure 1. This figure shows a partial structure of goals and plans for an agent's decision making behavior in response to a bushfire. We will use this example throughout the paper to illustrate our tool.

The top goal `RespondBushfire` can be considered as arising from some environmental information regarding an update about an impending bushfire, generating this goal. What is shown here is that the first decision the agent must make is whether to stay and defend (`Stay&Defend`) or evacuate (`EvacHouse`). Each of these options may have a context condition, such as for example `FireDistance` $>$ `10km` for evacuate, or `WaterTankLevel` $>$ `0.5` for stay and defend. This figure shows only the details of the plan to evacuate, which consists of three subgoals: to decide mode of evacuation; then assemble family members; then move to a safe location. Each of these subgoals in turn has a number of alternative plans that can be chosen to achieve that sub-goal in the higher level plan. At the bottom level of the hierarchy the plans consist of actions that affect the environment (e.g., `GetCar` does `Walk` and `Drive` actions),

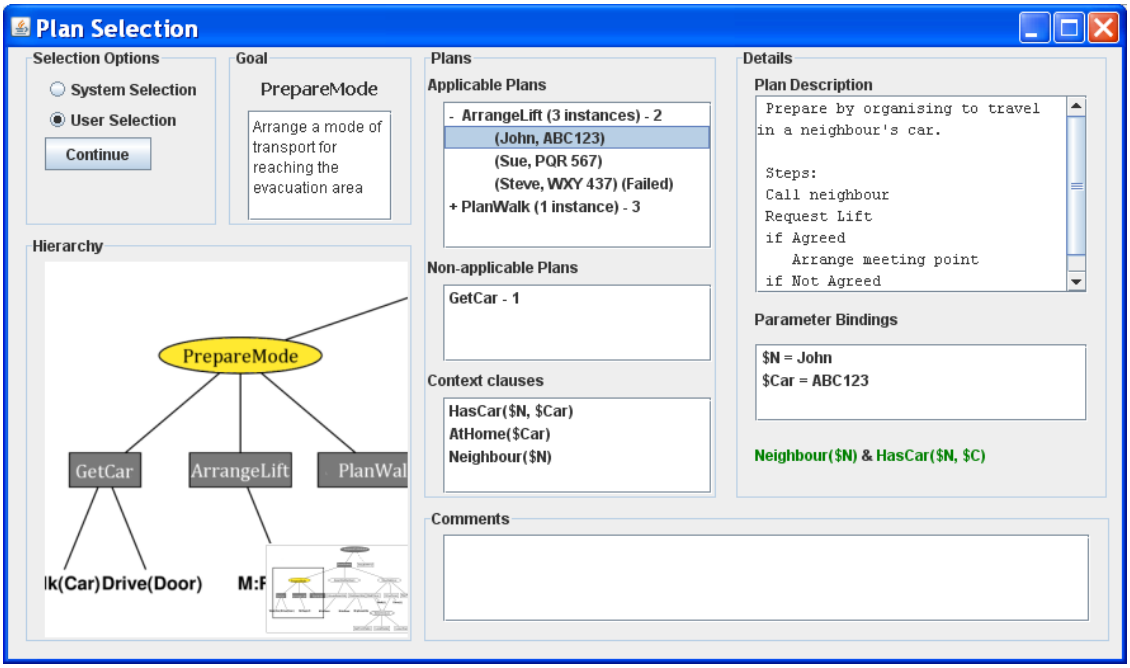

Figure 2: Screenshot of Tool Interface.

sending a message to another agent (e.g., `ArrangeLift` sends a lift request message) or some internal processing that is not visible in the diagram, but would be part of the plan description (as in `PlanWalk`).

In describing the tool we will focus on the decision as to which plan to choose for the goal `PrepareMode`.

3 TOOL DESCRIPTION

Before running a simulation the user must specify the decision points when the simulation should be interrupted and the tool interface populated with information. Potential decision points are any points where a goal or subgoal needs to be achieved and there are multiple plans available in order to achieve that goal. Figure 2 shows a screen-shot with the simulation paused at the point that the agent must decide which plan to select to achieve the subgoal `PrepareMode`. We describe now the different panes of the tool and the information they are providing regarding the agent's cognitive structure. Following this we provide an explanation of how to use the tool for the different tasks of exploring, controlling and modifying the cognitive structure of a particular agent. Two of our considerations in designing the interface to the tool were:

- in order to understand the plan selection decision being made (or to make that selection if in interactive mode), the user needs to have an overview of **all** relevant plans (both applicable and non-applicable) with their variables and context conditions, and must know which plans are applicable/non-applicable.
- to avoid incorporating in the interface to the tool, information which is already available via existing Repast Simphony tools (which need to be used in conjunction with our tool to obtain a full understanding) such as the current state of the environment.

3.1 Hierarchy

A significant benefit of using the BDI paradigm is that the goal-plan hierarchy captures a large amount of information in a clear and easy to understand way. This goal-plan hierarchy of the relevant agent, shown

in the Hierarchy pane on the left hand side, (Figure 2) shows the current plan choice decision with the goal to be achieved and the alternative plans available to achieve that goal. The user can pan and zoom in this figure to understand how the current plan selection and goal are related to the higher level goal that the agent is trying to achieve, and the plans that have been selected previously that have led to this point. (A UI improvement we are looking to implement is to make the tree expandable and collapsible so as to not overwhelm the user.) In our example the agent has chosen previously `EvacHouse` to respond to the goal `RespondBushfire` and is now at the point of deciding how to achieve the first subgoal in that plan, namely `PrepareMode`.

3.2 Goal Details

The goal pane, immediately above the right hand side of the hierarchy, shows the name of the current goal type (`PrepareMode`) and a textual description of this goal. If the goal has parameters or information carried within the goal object, this is also shown within this description pane.

3.3 Plans

The Plans frame, in the top middle of the screen has three sections: Applicable Plans, Non-applicable Plans and Context clauses. This area provides the information about the full set of plans which the agent has for fulfilling the current goal. The **Applicable Plans** are those which can be used in the current situation and are shown as the plan type, annotated with the number of instances (i.e., possible variants of the plan type). Clicking on the type shows the specific instances, as is seen for the plan `ArrangeLift` in Figure 2. Each plan in this list will have at least one instance. Some instances may have been tried in a previous attempt to achieve this goal instance, in which case they will be marked as "failed" as with the third instance of `ArrangeLift`. Each instance is identified by the bindings of the variables in the context condition. For example, the first instance of `ArrangeLift` is identified as `John, ABC 123` using the identifier of the neighbor variable and the car variable. When an instance is selected these parameter bindings are also shown in the parameter bindings section of the Details pane (discussed below). The plan types are ordered by priority which is shown beside the name of the plan. Unless there is a meta-plan to choose between applicable plans, the agent will randomly choose amongst the applicable plans with the highest priority. We see that of the applicable plans, `ArrangeLift` has the highest priority and `PlanWalk` the lowest. However `GetCar`, which is not applicable has higher priority than both these.

The **Non-applicable Plans** section shows the type and priority of any plans that are defined as relevant for this goal but which are not applicable. Non-applicable plan types by definition have no instances. As with the applicable plans, the non-applicable plan types are organized by priority, highest first.

The **Context Clauses** section shows the list of all atomic context condition clauses used across the set of relevant plans. This is to enable the user to understand what information is being considered across the full set of plans, in order to make a selection. Beside this, in the next pane is shown the context condition of the currently selected plan, enabling the user to examine which (atomic) clauses are used in the particular plan, and how they are logically combined. In order to provide additional information, clauses in the context condition of the selected plan are color coded in red or green, to indicate whether they evaluate to True or False, if they have been evaluated. (Some clauses will not have been evaluated. If an atomic clause within a disjunction evaluates to True, no further clauses will be evaluated, and conversely, if an atomic clause within a conjunction evaluates to False, no further clauses will be evaluated. In the case with variable bindings, it may be that there is no binding which makes all clauses True, in which case the plan is not applicable, but it may not be meaningful to indicate the truth value of particular atomic clauses.)

3.4 Details Pane

The details panel contains information specific to the particular plan instance or type selected. The **Plan Description** is a natural language description of what the currently selected plan (type) will do, with

reference to the use of any variables/parameters. This description is provided by the modeler at design time, and is generic to all instances of the plan type. In the **Parameter Bindings** section is a listing of each of the parameters of the plan and their bindings for the selected instance. If the plan is not applicable, no bindings are shown. At the bottom of this pane is the color coded context condition of the selected plan.

3.5 Comments

The comments section allows the user to enter any natural language comments. These are recorded along with the details of the goal, plans, and current world state, and can be later used by the modeler to extract relevant information.

3.6 Selection Options

At the top left of the screen the user has two different options when selecting a plan:

- User Selection
- System Selection

After selecting one of these options, clicking the *Continue* button will allow the simulation to progress.

3.7 State of the World

The plan selection interface does not (currently) include any explicit representation of the world, since this is covered by standard interfaces in ABMS platforms. For example, in Repast Simphony, it is possible for the modeler to have multiple displays of the world and for the user to access any of the attributes of an individual agent by selecting that agent in the displays.

4 USING THE TOOL

As discussed previously there are several levels at which a user may wish to use the interactive tool, ranging from simply understanding the programmed execution of a particular agent, through controlling the execution of that agent for exploration of possibilities, and finally specifying appropriate behavior of agents as a participant in the modeling process. We describe each of these separately below. In all cases the first step is to specify at which decision points in an agent the user wishes to pause the simulation, for exploration, and possibly control or modification. In our example there is a pause point at the goal `PrepareMode`.

4.1 Exploration and Understanding

As shown in Figure 2 the user can see from the hierarchy pane that `PrepareMode` is a subgoal of the plan `EvacHouse`, and can see from the plans pane that the plan `GetCar` is not applicable, and that there are three applicable instances of `ArrangeLift` as well as a single instance of the `PlanWalk` plan. The user can also see that the `ArrangeLift` plans have highest priority among the applicable plans, and by selecting among these instances can see the particular neighbors the agent could try to arrange a lift with (as the value of `$N` in the Parameter Bindings section of the Details pane). The user can also see (in the list of instances for `ArrangeLift` in the Applicable Plans section) that there has been a previous attempt to achieve the current goal `PrepareMode` by using the `ArrangeLift` plan with neighbor Steve, but it has failed.

In order to further understand what is going on, the user can select the non-applicable plan `GetCar`, in which case they would see that the context condition for this is `HasCar(self,$C) AND AtHome($C)` (i.e., that the agent has a car and that the car is at home). By exploring the world state (using the standard Repast Simphony tools) the user can then see that neither of the family cars belonging to this agent are

currently `AtHome`. They can also see that this plan has higher priority than the `ArrangeLift` plans, so would be preferred, if it was applicable. The user can also explore the context condition of `PlanWalk` and ascertain that this plan is always applicable. They can however see that its priority is lower than any of the other plans, so it will always be chosen as a last resort, if nothing else is applicable, or if all other options have failed.

Once the user has explored the situation sufficiently, they can select Continue and the system will choose one of the highest priority applicable plans (in this case one of the `ArrangeLift` plans) to execute. When the next identified decision point to pause at is reached this interface will be repopulated with the new decision information.

4.2 Interactive Selection

Our tool can also be used to allow the user to interact with and control the agent's decision making process by manually selecting a plan instance that would not, or may not, be chosen automatically by the system. This is a powerful mechanism for allowing users to feel in control of a particular agent, and immerses them more fully into the simulation. It is well understood that control is an important aspect of immersion, which improves the likelihood of learning (Shute, Ventura, Bauer, and Zapata-Rivera 2009) in serious games. For community education in high risk bushfire areas, this feature allows people to "try out" how they could behave in different contexts in a high-level and intuitive way and then understand some of the implications of those actions. For example, a community member who chooses to evacuate from their home when an approaching fire threat is already too close, may realize that this has severe and undesirable consequences.

By selecting the mode User Selection (rather than System Selection) the selected instance (as long as it is from the applicable set) will be executed once the Continue button is selected. Doing this will cause a dialog box to appear which prompts the user to specify whether this selection is relevant for this time only, or whether such a selection should always be made. If the user indicates that the selection is relevant to future decisions, they are asked to comment on this using free text in the comments box. There are three situations which are relevant to consider and these are described for the user. The first situation is that the selected plan should *always* (i.e., not only when the world state is as it is now.) be the preferred plan (if applicable) for achieving the current goal. In this case its priority should simply be set appropriately, as described below, ensuring that priorities of other plans, including those not currently applicable, are also modified if needed to give the desired precedence. A second possibility is that in the current situation (and possibly some others), this plan should be preferred. This preference will need to be realized by a meta-plan which reasons about which applicable plan to select, and so it is necessary to specify what factors about the current state of the world are relevant for making this the preferred choice. A third situation is similar to this but is when a particular instance of a plan type with multiple instances is preferred not only over instances of other plan types, but over other instances of the same plan type. Here the meta-plan will need to reason using information related to the variables bound in the context condition. For example in the plans of type `ArrangeLift` we may prefer to first try neighbors who have a smaller family, or a larger car, or live closest. These are all properties of the entity referenced by the `$N` binding in the context condition and can be compared by a meta-plan which has a reference to all the applicable plans and can use additional information to select between them. (The metaplan is not currently available for inspection by the tool, although the way it works can be described as part of the goal description if desired. We are exploring a more structured declarative representation of the metaplan which could be inspected, and even edited)

Some of the information which the user is asked to explain in natural language could potentially be obtained via a more complex structured interface. However, we are waiting to do some user evaluation before we add additional interface complexity. Some of the issues in further automation are discussed in Section 6. Once the user has entered their comments, or indicated the selection is only relevant to this case, the selected plan instance will be placed in the intention structure for execution and processing continues. If priorities have been changed, these will take effect for all other plan selections related to this goal in the

current simulation, and will also be integrated into the code for further executions once the program has been recompiled. Although the comments field does not support any direct changes to program code, the comments may be used by the modeler to incorporate the suggested changes. It is also possible for the user to make some limited changes directly, as outlined in the following section.

4.3 Modifying the Model

End users can be a valuable resource in modeling, and can also be more comfortable with using the results if they have been involved in the modeling process. The tool provides some ability for users to assist in design of the cognitive structures of agents in the simulation. There are essentially three kinds of modifications which can be made: the preferences regarding which plans should be chosen in which situation, the conditions under which a plan should (or should not) be available for use (i.e., the context condition), and the actual plans which are available - i.e., what the agent could do at any point. We describe below the support available for each of these.

Modifying plan priorities: As we have seen in the previous sections, the plan types are listed in the interface in order of priority, with their priority value clearly specified. When the system makes a plan selection, it chooses a plan instance from the applicable plan type with the highest priority, or in the case where there are multiple plan types with the same priority, it will choose randomly between them. (If a meta-plan has been specified, this can override both priorities and the randomness of the choice.) This priority applies to all instances of a particular plan type. Since the ordering of priorities can have a significant impact on the outcome of a simulation, it is important that the user is able to adjust these priorities and explore how this changes the simulation output. The priority of all plans can be directly edited via the tool user interface, resulting in immediate automated re-ordering of plans to reflect this change (if necessary). The updated priorities are then used for future system plan selections in the current simulation and (after re-compilation) in future runs.

Modifying the context conditions of plans: It is important to allow the user the ability to modify the context conditions of plans, in order to adjust the information as to when certain plans may, or should not, be used. However, this introduces complexities as some aspects of context conditions are used to bind variables that are required in order to execute the plan. For example, in our plan `ArrangeLift`, there is a context condition `Neighbor($N) AND HasCar($N,$C)` which serves the dual purpose of binding `$N` to some particular neighbor, as well as ensuring that the clause returns `True` (i.e., there is some neighbor that has a car). When the plan executes, one of the steps is to retrieve the phone number of the particular neighbor that `$N` is bound to, and call them. If this clause is removed from the context condition, there will be no value for `$N`, and as a result the plan cannot be executed. Other context conditions may be used only to indicate the plan's applicability and these can potentially be changed without causing problems with the plan's execution. For example, the `EvacHouse` plan includes a clause `fireDistance` $>$ `10km` which specifies that this plan is only applicable when the fire is a certain distance away. The user can safely increase or decrease the distance or remove that clause completely without affecting the plan's execution.

Our current approach is to allow the user to modify any context condition clause, but with a warning regarding the removal of variables that are likely to be used in the plan's execution. (Recall that a programmer provided description of what the plan does is available at the interface.) To modify a context condition, the user must first select a plan type, and then right click on the desired clause, which allows them to either edit or remove the clause. Any changes will be reflected after the model is recompiled. (If the user has removed context conditions containing variables needing to be bound, this may result in the program crashing when run, with an exception. It could also lead to strange results.) We plan to evaluate with users before making further refinements to perhaps automatically detect and disallow certain modifications which remove bindings of required variables.

Other suggestions: While other parts of the model and cognitive processes of the agents may be more difficult to modify in a way suitable for a non-programmer, it is still important that the user can indicate other modifications particularly within the context of a particular decision point. For example, the user

may want to suggest the inclusion of a new plan to handle a goal in a specific context, or indicate that there are other parts of the world which should be modeled as they are relevant to the decision making at some point. While future work will look at ways to capture this information automatically, at present we allow the user to enter comments about suggested modifications at any decision making point, and then store these comments along with a record of the agents beliefs, the current goal, the current applicable plan set and any other important information which can then be used by the modeler to make changes at a later point. We believe that allowing these comments to relate to a specific context improves the user's ability to provide meaningful feedback.

5 ISSUES AND CHALLENGES

As with any piece of software there are competing requirements which must be balanced in some way. The key competing requirements in this case are between the desire for the tool to be simple and intuitive for an end-user, at the same time as being sufficiently powerful to allow exploration and understanding of complex nuances, and potential ability to modify the agent's decision making in non-trivial ways. We have tried to follow the principle that conceptually simple things should be simple to do/see, with additional complexity experienced only when needing to do more complex things. Initially it might seem that the simplest approach to allowing a user to participate in defining what plans are viable in a particular situation and what the preferences over them are, is simply to show them all plans at each decision point and ask for information about which plans are possible and which are preferred. However this is problematic from at least two aspects. The first is that it cannot be assumed that the reason for selecting a particular plan belongs with the description of that plan, the second is that the reasons for a choice in the current situation must be appropriately generalized to sufficiently similar future situations. We explain each of these a little more fully.

5.1 Applicability of Plans

Initially it might seem that the reason for choosing a particular plan at a certain point in the simulation, could be obtained from the user and then simply coded into the plan's context condition, or applicability. We illustrate the problem with this approach using our example. Suppose that at a certain point in the simulation the user says the correct decision (in our example decision point) is to choose `PlanWalk` and the reason is that there is no car available and none of the neighbors have a car. This is a perfectly valid reason (and indeed in our example would result in this plan being selected). However this reason does not belong in the PlanWalk plan. Rather the reasons are part of unfulfilled conditions for using otherwise preferable plans. We would not want to attach such a reason to this plan as it would also unnecessarily exclude it in a situation where, for example, the car was at home, but we had failed to get it because the battery was flat. So, in order to understand, and to make, decisions about plan selection, it is necessary to be aware of all the potential plans for achieving the goal, and the aspects of the state that affect the applicability of any of them. For this reason, we show in the tool the set of context clauses from all plans, as well as both applicable and non-applicable plan types.

5.2 Generalizing Selection Criteria

There may well be aspects of the world state that a user wants to indicate are important for a particular preference, or should be added as a criteria for flexibility. It might at first consideration seem that such conditions could somehow be selected from the world state and then encapsulated in a context condition clause to be added to a plan, or a preference criterion. However this is not entirely straightforward. For example, a user might choose a `PlanWalk` plan instance (in our `PrepareMode` example) because of the location of the evacuation point and the agent's current location. However it is not these exact locations that is of interest, but rather the underlying reason that the one is not far from the other. It is this that must be captured before the condition can be used either for a context condition or as a preference. Any

support tool attempting to assist the user in specifying such conditions and adding them automatically is quite complex. Our current choice is to allow users to express this in natural language which must then be interpreted by a programmer.

6 CONCLUSION AND FUTURE WORK

We have presented a tool which supports a user in understanding and controlling the decisions of a particular agent in a simulation, and also allows a user to contribute as a domain expert in helping to specify the desired modeling of the agents. While our aim is to allow a user to fully participate in the cognitive processes of a BDI agent, and to provide feedback which can be used to improve the model in an automated way, there is still significant work to be done before this aim is fully realized. However, our next step is to work with end users to obtain their feedback on the usability of the tool, prior to adding greater flexibility and/or modifying the current interface or functionality.

We are also aiming to port the work we have done using JACK as the BDI agent development platform, to a system using an open source freely available platform, and to do this as a plug-in to Repast Simphony.

This work can be considered to lie in the area of participatory design and simulation. There is a large body of work in this field, ranging from involving stakeholders in a real life role-playing game in order to test or inform a related a computer model, e.g., Guyot, Drogoul, and Honiden (2006), to directly controlling an agent during a simulation in order to understand, validate or design a model, e.g. Chu et al. (2012). For example, Taillandier and Chu (2009) record the decisions of an expert-user, acting on behalf of an agent during a simulation, to define the agent's behavior in terms of a utility function. To our knowledge however no work has been done which allows users to observe, interact with, and specify an agents cognitive process, during a simulation, in a manner that leverages the BDI perspective. The BDI paradigm provides a level of abstraction that is accessible with respect to both understanding and specifying agent behavior, and we believe that our interactive tool will further assist both modelers and users in simulating agents with cognitive abilities.

ACKNOWLEDGMENTS

This work was carried out with financial support from the Australian Government (Department of Climate Change and Energy Efficiency) and the National Climate Change Adaptation Research Facility and ARC grant DP1093290. The views expressed herein are not necessarily the views of the Commonwealth, and the Commonwealth does not accept responsibility for any information or advice contained herein. The authors would also like to thank the staff of the Country Fire Authority (CFA) for their assistance and feedback.

REFERENCES

Bordini, R. H., J. F. Hübner, and M. Wooldridge. 2007. *Programming Multi-agent Systems in AgentSpeak Using Jason*. Wiley. Series in Agent Technology.

Chu, T.-Q., A. Drogoul, A. Boucher, and J.-D. Jucker. 2012. "Towards a Methodology for the Participatory Design of Agent-Based Models". In *Principles and Practice of Multi-Agent Systems*, edited by N. Desai, A. Liu, and M. Winikoff, Volume 7057 of *Lecture Notes in Computer Science*, 428–442. Springer.

Cohen, P. R., and H. J. Levesque. 1990. "Intention Is Choice with Commitment". *Artificial Intelligence* 42:213–261.

Guyot, P., A. Drogoul, and S. Honiden. 2006. "Power and negotiation: lessons from agent-based participatory simulations". In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, edited by H. Nakashima, M. P. Wellman, G. Weiss, and P. Stone, 27–33. ACM.

North, M. J., N. T. Collier, and J. R. Vos. 2006. "Experiences creating three implementations of the repast agent modeling toolkit". *ACM Trans. Model. Comput. Simul.* 16 (1): 1–25.

Padgham, L., D. Scerri, G. Jayatilleke, and S. Hickmott. 2011, December. "Integrating BDI Reasoning into Agent Based Modelling and Simulation". In *Proceedings of the 2011 Winter Simulation Conference*,

edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Padgham, L., J. Thangarajah, and M. Winikoff. 2005. "Tool Support for Agent Development using the Prometheus Methodology". In *Proceedings of the Fifth International Conference on Quality Software*, QSIC '05, 383–388. Washington, DC, USA: IEEE Computer Society.

Padgham, L., and M. Winikoff. 2004. *Developing Intelligent Agent Systems: A practical guide*. Wiley Series in Agent Technology. John Wiley and Sons.

Pokahr, A., L. Braubach, and W. Lamersdorf. 2005. "Jadex: A BDI Reasoning Engine". In *Multi-Agent Programming: Languages, Platforms and Applications*, edited by R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, Volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, 149–174. Springer.

Ramanath, A. M., and N. Gilbert. 2004. "The Design of Participatory Agent-Based Social Simulations". *Journal of Artificial Societies and Social Simulation* 7 (4).

Rao, A. S., and M. P. Georgeff. 1995. "BDI-agents: from theory to practice". In *Proceedings of the First Intl. Conference on Multiagent Systems*, edited by V. R. Lesser and L. Gasser, 312–319. San Francisco: AAAI.

Shute, V. J., M. Ventura, M. I. Bauer, and D. Zapata-Rivera. 2009. "Melding the power of serious games and embedded assessment to monitor and foster learning: Flow and grow". In *Serious games: Mechanisms and effects*, edited by U. Ritterfeld, M. J. Cody, and P. Vorderer, 295–321. Routledge, Taylor and Francis.

Taillandier, P., and T.-Q. Chu. 2009, oct.. "Using Participatory Paradigm to Learn Human Behaviour". In *Knowledge and Systems Engineering, 2009. KSE '09. International Conference on*, edited by N. T. Nguyen, T. D. Bui, E. Szczerbicki, and N. B. Nguyen, 55 –60. IEEE.

Winikoff, M. 2005. "JACK Intelligent Agents: An Industrial Strength Platform". In *Multi-Agent Programming: Languages, Platforms and Applications*, edited by R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni, Volume 15 of *Multiagent Systems, Artificial Societies, and Simulated Organizations*, 175–193. Springer.

AUTHOR BIOGRAPHIES

LIN PADGHAM is Professor of Artificial Intelligence in the School of Computer Science and I.T. at RMIT University, Melbourne, Australia. Her research interests include modeling, building and understanding intelligent agents for complex application areas requiring a balance between goal directed long-term behavior and reactive response to a dynamic environment. Her email address is lin.padgham@rmit.edu.au.

DAVID SCERRI is a PhD candidate in the Intelligent Systems group at RMIT University, Australia. He is currently researching the validation and analysis of Agent Based Models. His email address is david.scerri@rmit.edu.au.

SARAH HICKMOTT is a post doc in the Intelligent Systems group at RMIT University, Australia. She is interested in supporting decision making around sustainable futures and climate adaptation, with the use of agent oriented modeling and simulation, and automated planning. Her email address is sarah.hickmott@rmit.edu.au.