Explaining Simulations through Self Explaining Agents

Maaike Harbers, John-Jules Meyer, and Karel van den Bosch

M. Harbers, K. van den Bosch and J.-J. Meyer. (2010) Explaining Simulations Through Self Explaining Agents. *Journal of Artificial Societies and Social Simulation*, 13(1)4.
http://jasss.soc.surrey.ac.uk/13/1/4.html

Abstract

Several strategies are used to explain emergent interaction patterns in agent-based simulations. A distinction can be made between simulations in which the agents just behave in a reactive way, and simulations involving agents with also pro-active (goal-directed) behavior. Pro-active behavior is more variable and harder to predict than reactive behavior, and therefore it might be harder to explain. However, the approach presented in this paper tries to make advantage of the agents' pro-activeness by using it to explain their behavior. The aggregation of the agents' explanations form a basis for explaining the simulation as a whole. In this paper, an agent model that is able to generate (pro-active) behavior and explanations about that behavior is introduced, and the implementation of the model is discussed. Examples show how the link between behavior generation and explanation in the model can contribute to the explanation of a simulation.

Keywords

Explanation, agents, goal-based behavior, virtual training

Introduction

Emergent interaction patterns in agent-based simulations provide insight into the processes that are being simulated. In some simulations, the rules according to which agents behave are completely reactive and rather simple. For instance, the famous Boids program (Reynolds 1987) simulating the flocking behavior of birds only consists of three simple rules: separation (steer to avoid crowding local flockmates), alignment (steer towards the average heading of local flockmates), and cohesion (steer to move toward the average position of local flockmates). Although the single agents are not complex, interesting interaction patterns arise. To explain these macro properties, e.g. the shape or direction of a flock of birds, all reactions of the agents in the simulations have to be monitored. Besides the rules according to which the entities behave, their starting positions, number and the environment play a crucial role in the outcome of the simulation (see e.g. the experiments in artificial life). Therefore, the reactive rules of the agents by themselves are not sufficient as a basis for explaining the system. Using the particular configuration of agents and environment also does not lead to deeper understanding per se. Explaining the behavior of the whole simulation might even become more difficult when the agents themselves do not only have reactive behavior, but also pro-active behavior that is not directly explainable from their interaction with the environment. In contrast to simulations with

simple agents, the single agents' behavior is more variable, and harder to predict and understand. However, we do believe that using exactly this pro-active behavior in the form of the goals underlying the agents' actions will lead to a natural explanation of the agent behavior and thus might form a good basis for explaining the simulation as a whole.

If pro-active agents would be able to explain their actions, e.g. because of a particular plan or to achieve a certain goal, the understanding of the emergent processes arising from their interactions would be facilitated. The similarities or contradictions in the explanations of several self-explaining agents would help to construct an explanation of the overall processes. For instance, the explanations of agents which are part of a fleeing group would help to understand the reasons of the flight behavior of the group as a whole. Agent explanations like *I'm trying to get away from the fire* result in a different overall explanation than agent explanations such as *I just followed the others*. The example shows that an aggregation of (abstract) micro level explanations could add insight in the macro behavior over and beyond a definition of the rules that created it.

Although this idea is appealing, it needs pro-active agents that can explain themselves at the micro level. Although there are some first attempts at building this type of self explaining agents (see section 2 for related work), no satisfactory solution has been given yet. In this paper, we therefore concentrate on the way in which pro-active agents can explain themselves first. We used a simulation-based training system for firefighting as an application involving intelligent agents with self-explaining capabilities. The trainees have to fulfill a task or mission while they are surrounded by and interact with virtual characters which can be their team-members, opponents, and neutral participants. In order to create realistic training scenarios, these characters have to perform believable behavior which can be complex. In order for the trainee to understand the reaction of the simulation to his own behavior, it is important that those characters can explain themselves. This problem is exacerbated when the agents in the simulation interact (as a team or with neutral bystanders). Without knowing the motivation of a character's actions, it is harder for a trainee to understand the situation and learn from the training session. This application therefore seems a good stepping stone for developing self explaining simulations in general.

The outline of this paper is as follows. In section 2 we discuss the requirements of explanations in general, and more specifically in simulation-based training systems. Some current approaches of self-explaining agents in simulation-based training systems are discussed. In section 3, we introduce an agent model for the generation of behavior and explanations. Then, in section 4 we discuss issues and considerations arising from the implementation of such a model. Finally, we give a conclusion and directions for future research in section 5.

Explanation in simulation-based training

A single event can be explained in many ways. For example, the fact that a firefighter dropped one of his tools on the floor can be explained by the fact that *he stumbled*, *he hit something on the floor*, or *it was dark and he could not see anything*. A whole other type of explanation about the falling tool is that it fell *because of the gravitation force*. All of the examples are explanations of the same event and one explanation is not by definition better than another. Each explanation uses part of the elements that may be said to have 'caused' the action. The desired explanation to be used depends on the context in which it is given and on the person to whom it is given.

For making simulation entities capable of explaining their actions, we should consider what types of explanation are most useful. An agent giving explanations like *I went to the fire truck because I was designed in such a way by the programmer*, will not improve understanding of the single entity, leave alone the complete simulation. The explanation *I went to the fire truck to take an explosion meter* would probably be more in the right direction. Besides the type of explanation, the extensiveness has to be determined. Additional information to the last example could be *I want to measure explosion danger*, *I believe there is an explosion meter inside the fire truck*, and *explosion meters measure explosion danger*. Probably not all information is necessary, thus a proper selection should be made.

The examples in the last paragraph revealed the agent's goals to go to the fire truck (to measure explosion danger and to get a meter), and beliefs related to these goals (there is an explosion meter and such a device measures explosion danger). It has been demonstrated that BDI agents, agents based on the Belief Desire Intention model (Rao and Georgeff 1991), do provide characters in computer games with believable behavior (Norling 2003). Although it only matters whether agents behave as if they had beliefs and desires to understand the behavior of agents, for the generation of understandable explanations, it is important that the agents have actual beliefs and desires and reason with them. This thesis is supported by research on explaining expert systems, which found that users of expert system often not only want to know how, but also why a conclusion has been reached (Ye and Johnson 1995). An important difference between expert systems and BDI agents is that the BDI agents are proactive, i.e. they have goals. Therefore, the creation of why-explanations is much more natural than in expert systems; the trace of steps behind one action already answers why-questions. We will therefore concentrate on the use of BDI agents for explanation in simulations in the rest of this paper.

Related work

Only few simulation-based training systems involve intelligent agents that are able to explain their own behavior. The first account of self-explaining agents is called Debrief (Johnson 1994). Debrief has been implemented as part of a fighter pilot simulation and allows trainees to ask an explanation about any of the artificial fighter pilot's actions. To generate an answer, Debrief modifies the recalled situation repeatedly and systematically, and observes the effects on the agent's decisions. With the observations, Debrief determines what factors were responsible for ('causing') the decisions. A possible shortcoming of Debrief is that it derives what *must have been* the agent's underlying beliefs. This might result into proper explanations, but sometimes different reasons can be responsible for the same action. For instance, a firefighter might search for a victim in a house's basement on his own initiative or because his commander told him to do so. The different motives are not observable from the firefighter's behavior, but might improve understanding of the complete situation. If the firefighter came up with the idea to go the basement himself, the commander might not know about it and give commands that would bring the man into danger. By making beliefs and goals in the reasoning process explicit instead of deriving them from observable behavior, the actual reasons for executing an action can be given.

A more recently developed account of self-explaining agents is the XAI explanation component (Van Lent et al 2004). The XAI system has been incorporated into a simulation-based training for commanding a light infantry company. After a training session, trainees can select a time and an entity, and ask questions about the entity's state. However, the questions involve the entity's physical state, e.g. its location or health, but not its mental state. Thus no explanations about the reasons for its actions are given. A second version of the XAI system (Gomboc et al 2005; Core

et al 2006) was developed to overcome the shortcomings of the first; it claims to support domain independency, modularity, and the ability to explain the motivations behind entities' actions. This second XAI system is applicable to different simulation-based training systems, and for the generation of explanations it depends on information that is made available by the simulation. It was found that most simulations do not represent agents' goals, and preconditions and effects of actions, so still no explanations of agents' reasons could be given.

Although a well functioning modular explanation tool would be desirable, we think that the generation and explanation of behavior are closely connected. The reasoning steps taken to generate an action should be reproducible in an explanation about it. This reproducibility should be taken into account while designing the generation of behavior. In the next section we present an agent model in which such a connection between action generation and explanation is made.

A model for the generation and explanation of behavior

As argued in the previous section, we aim to develop self-explaining agents based on a BDI model, and closely connect the generation and explanation of behavior in this model. We use the BDI model to obtain agents with explicit goals, and actions generated based on these goals. The goals also serve to explain the actions. To go from goals to actions, the agents use beliefs and the ability to make plans. For the creation of plans, we have looked at planning methods based on hierarchical task networks (HTNs) (Russell and Norvig 2003). In HTN planning, an initial plan describing the problem is a high-level description of what is to be done. Plans are refined by action decompositions, which reduces a high-level action to a set of lower-level actions. Actions are decomposed until only primitive actions remain in the plan. BDI concepts fit well in these structures, e.g. an initial plan corresponds to a desire or goal, and a plan corresponds to an intention. Moreover, as action decompositions are explicated, primitive actions can be explained by the high-level actions they originated from.

We have inspired our agent model on the GPGP/TEAMS approach (Lesser et al 2004), which is currently one of the most extensive accounts of general (HTN) planning. GPGP (generalized partial global planning) is a framework for the coordination of small teams of agents, and therefore it makes use of local (from the perspective of one agent) and non-local task structures. TAEMS (task analysis, environment modeling and simulation) is the language used to represent these task structures. In the GPGP approach, coordination and scheduling are distinguished, where coordination refers to planning among agents and scheduling to the planning within an agent. The underlying model of the GPGP approach can be represented conceptually as an extended AND/OR goal tree where the leaves of a tree are primitive (non-decomposable) actions. Between the nodes of the tree there exist coordination relationships: task-subtask relations and non-local effects. Non-local effects are relations between two goals or tasks at any place in the goal tree and involve things like one goal facilitating the achievement of another goal.

As in the GPGP approach, our model also consists of a tree with goal and plan decompositions. In this section we first describe how beliefs, goals, plans and actions relate to each other in our approach. Next, we demonstrate how our model is used to generate behavior. Last, we discuss how it generates the corresponding explanations.

The BDI-based agent model

Figure 1 shows an abstract plan tree based on the BDI-model of a firefighting agent. Later the example is discussed more extensively, and here it only serves to facilitate reading. In general, the top of the tree is a goal describing the main objective of an agent, which is possibly divided into sub-goals. Goals are abstract descriptions of (parts of) desired world states, i.e. they describe what an agent wants to achieve. Lower in the tree, goals are divided into plans, which are composed of (abstract) actions and describe how a goal can be reached. Plans are divided into sub-plans, sub-sub-plans and eventually actions. Actions are no further decomposed. All of the agent's possible goals and plans are part of its BDI-model and depending on the current world state, one or more of them become active. An agent's knowledge about the current world state is represented in its beliefs. Beliefs relate to the connections between the agent's goals, plans and actions, and determine which goals and plans become active. For example, a fire-fighter would only adopt the goal to extinguish a fire, if he believed that there (possibly) was one. Goals and plans relate to their descendants in several ways; either all sub-goals/plans must be achieved or at least one or at most one, and the order of execution does or does not matter for the achievement of the main goal or plan.

The agents' beliefs can be classified into general and observation beliefs. The truth of general beliefs does not depend on a particular situation. An example in the firefighter domain would be the belief that combustion requires oxygen. Observation beliefs are context-specific and their truth depends on an agent's current situation, for example, the belief that there is somebody in the burning house. Observation beliefs can be interpretations of observations. For example, the belief that a house is burning is an interpretation of observations such as smoke, heat, sounds of fire, etc. Such interpretation relations are also represented in our model.

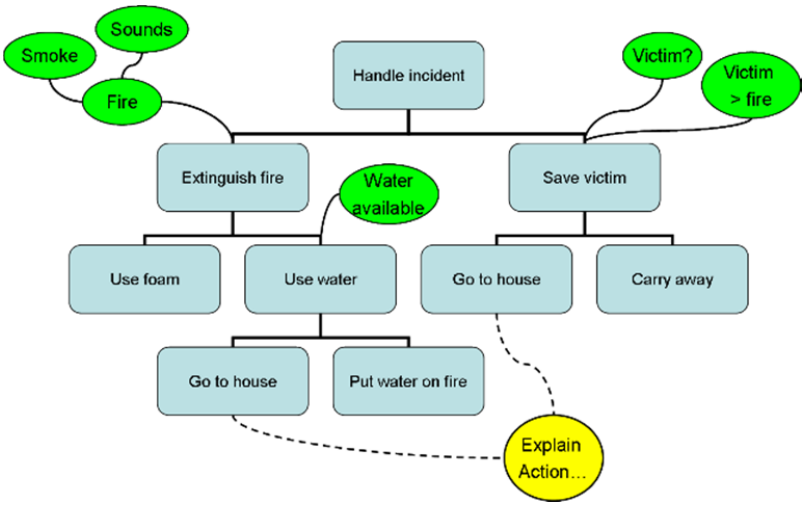

Figure 1. BDI-model of a firefighter agent

Generation of behavior

A character in a simulation-based training system has a main goal or mission. To achieve this goal, it has to adopt proper sub-goals and plans, and finally execute the corresponding actions. In

Figure 1, the firefighting agent's main goal is to *handle the incident* successfully, and it is divided into the two sub-goals *extinguish the fire* and *save the victim*. The agent checks its beliefs to determine which of the sub-goals currently apply and only these become active. Our agent hears sounds of fire and sees smoke, and derives that there must be a fire. To achieve the goal *extinguish the fire* it has to choose between two plans: extinguish the fire by *using foam* or *using water*. Because the agent has the belief that *water is available* it uses water, and it has to *go to the house* and *put water on the fire* sequentially. However, while extinguishing the fire, our agent suddenly sees a victim and the goal *save the victim* becomes active. Ideally, extinguishing the fire and saving the victim are aspired simultaneously, but this is not always possible due to limited resources (e.g. there are insufficient firefighters available). The agent's model contains preferences for sub-goals in order to make choices between sub-goals. In our example, the agent believes that saving the victim is more urgent than saving the building. It thus interrupts extinguishing the fire, and starts carrying away the victim.

Generation of explanations

In a (training) simulation, only an agent's actions are observable, but nothing of the reasoning steps behind them. Whereas the agent's model is used from main goal to plans to actions for the generation of behavior, it will start with the actions for explanation generation. The beliefs, plans and goals that initiated an action all partly explain that action, however providing the complete list would result into quite extensive explanations, especially in bigger models. Out of the many plans, goals and beliefs, the information that is most useful or has most explanatory power needs to be selected.

The algorithm used to select information for explanations traverses the tree bottom up and gives relevant information about the decisions along the way. The basics are simple; to explain the action *Use foam*, we get the following reasons:

```
I use foam, because
   I want to Handle incidents and I believe there is Fire
   therefore I want to Extinguish fire and
      because there is no Water available I use foam
```

In general, giving all reasons for all decisions along the path through the tree to the action might result into of an overload of information. We therefore apply a kind of filter such that presumed known information is left out, e.g. in the example it is no use to state that you want to handle incidents. This goal is known to all involved. If it is clearly perceivable that there is a fire, that part is also left out. This would lead to an explanation that *I use foam, because there is no water available*.

Another heuristic that is used to filter information out of the explanations is to find possible places where ambiguity on the agent's goal arises. This can be nicely illustrated with a possible explanation for the action *go to the house* of our firefighting agent. In Figure 1 it can be seen that this action is part of two plans, it can be executed to extinguish a fire or to save a victim. Possible confusion about which plan the action aims to execute can be solved by going up in the tree, starting with the action till both lines meet. In this case, they meet at the top goal, where it has to be decided whether to first extinguish the fire or save the victim. The beliefs that determined the outcome of this decision are *there is a victim* and *saving victims has priority over extinguishing fires*. The explanation of the action *go to the house* is thus:

```
I Go to house because I believe
   there is a Victim and Safe Victim > Extinguish fire
```

Note that explanations generated by this algorithm are useful when plans are not interrupted. However, problems in explanation generation arise when plans are interrupted, e.g. in the above example where the firefighter switches from extinguishing the fire to saving the victim. Suppose he started executing the plan for extinguishing the fire and went to the house to achieve that goal. At that moment he sees the victim and starts saving the victim. If the firefighter is asked why he went to the house, he would respond that it was to extinguish the fire. That explanation might give the wrong impression that he did not see the victim (yet). The complicated explanation should be that going to the house was done to extinguish the fire, but the goal was switched and the action was also useful for saving the victim.

Implementation of the agent

After defining an agent model in which the agent's reasoning concepts are explicitly represented, we discuss some of the issues that arise when implementing the model. In this case the implementation is relevant for the theory as it can quite easily interfere with the intuition that generation and explanation of actions can be nicely linked in BDI agents. First, we discuss requirements on the implementation of explanation capabilities in section. For instance, the agent needs to have access to its own beliefs, goals and plans, i.e. it requires the ability of introspection. Then, a proper (agent) programming language has to be chosen. Finally, the BDI-model must be translated to constructs in the chosen programming language.

Implementation of explanation facilities

The implementation of self-explaining agents should fulfill four requirements. First, the BDI concepts should be explicitly represented, such that one can easily refer to the goals and beliefs that explain an action. Second, the operationalization of these concepts (i.e. how actions are generated from the goals, beliefs and plans) should be available for reproduction in explanations to indicate why actions are performed in a certain order and plans and goals have priorities. Third, a self-explaining agent should be able to introspect. An agent needs to have knowledge about its own states and processes in order to explain them. The fourth and last requirement is that a self-explaining agent needs to have memory. To explain its actions, an agent needs to know about its states and processes not only at the time they occur, but also at later points in time.

The third requirement is that agents need to introspect in order to explain the reasons for which they executed their actions. However, in the current agent programming languages the information required for explanations is either implicit in the program or is present in the interpreter, but not available for introspection. (The only possible exception is DESIRE (Brazier et al 1997), although this platform does not explicitly provide introspection on the decision mechanism, which is exactly what is needed for the explanations). Fortunately, the lack of total introspection is not a major drawback as we do not need its full power. We do not use the introspection as input for the agent's deliberation process, but only as input for an explanation facility that can be explicitly invoked. The introspection needed for explanation therefore can be

either created artificially within the program, or as a separate module or program for handling the explanations.

The first solution yields logging an agent's reasoning steps within its own program, e.g. by adding update actions to it in which the agent stores its own processes as beliefs. However, this method has as a consequence that the bookkeeping necessary for the explanation can interfere with the generation of 'real' plans and actions. If the agent needs to be able to react to new circumstances, this might cause problems. We illustrate the possible problems with the fire-fighting agent of section 3 that sees a victim while it is extinguishing a fire. If logging is part of the agent program, update actions can be executed just before or after executing an action. If updating is done after execution, it could happen that the agent finishes the action of putting water on the fire, but then starts saving the victim without making the update that it extinguished the fire. After it saved the victim, the agent wants to continue extinguishing the fire although it is not necessary anymore. Its other actions, e.g. informing the head quarter might fail. On the other hand, if updating is done before execution of an action and there is an interruption, the agent unjustly believes it executed a particular action and will encounter problems if it tries to execute the next action of the plan.

The second solution, building a separate explanation module, also has a disadvantage. Although the logging and explanation generation will not interfere with the actual program of the agent, action generation and explanation are no longer intrinsically connected. In agent implementations in which the interpreter provides the operationalization of the BDI model, the connection can be kept by changing the interpreter such that it provides the logging of the decisions during the action generation. The downside of this approach is that one would have to modify existing agent platforms for this purpose. In agent implementations in which the operationalization is explicitly programmed, the logging can be added in the agent program itself. A danger however is that the programmer alters some decision making module without changing the explanation parts belonging to it, which is inefficient and undesirable.

As long as there are no programming languages offering explanation facilities, a balance between both options has to be found. Explanation generation should not hinder the planning of the agent, but generation and explanation should also not be completely separated from each other. In such a solution the agent will not have general introspection, but only the introspection necessary to generate explanations. At the end of this section a possible solution implemented in 2APL (Dastani 2008) is discussed.

Agent programming languages

Agent programming languages can be divided into languages that do and do not explicitly represent BDI concepts. Languages such as 2APL (Dastani 2008) and ConGolog (Giacomo et al 2000) have explicit representations of the agent's beliefs and goals. Agent programming languages in which the concepts are not explicitly represented are for example Soar (Rosenbloom et al 1993) and Jade (Bellifemine 2007).

Furthermore, agent programming languages can be classified according to whether the agent's flow-of-control is explicit in the program or implicit in the interpreter (or deliberation cycle). Languages with an implicit flow-of-control represent the elements with which the agent reasons, e.g. beliefs, goals and plans. The operationalization of these concepts is done through the design of the agent and its deliberation process. An example of such an agent programming language is Jason/AgentSpeak (Bordini et al 2007). Languages with an explicit flow-of-control in contrast

explicitly represent the relations between the agent's reasoning elements, usually by if-then-else rules. In such languages the identity of the elements is not explicit, and can only be known if they are implemented in separate files called belief or goal bases. Jade is the prime example of an agent programming language with a more explicit flow-of-control.

The consequences for explanation generation of explicit and implicit representations of BDI-concepts and flow-of-control can be illustrated by a firefighter agent's choice between extinguishing a fire and saving a victim. The rule the agent follows is that normally saving a victim has priority over extinguishing a fire, except if there is explosion danger. In that case the agent extinguishes the fire before it saves the victim (otherwise the fire might initiate an explosion which results into many more victims). Reasoning elements in this example are the goal *handle incident*, the sub-goals *extinguish fire* and *save victim*, and the beliefs *fire*, *victim* and *explosion danger*. The relations between the elements are the priority rules: without explosion danger saving a victim is more urgent than extinguishing a fire, with explosion danger vice versa. The reasoning elements and the relations between them both partly explain the agent's choice, thus both should be available for introspection.

Representations with an implicit flow-of-control usually consist of rules like Goal <– Belief | Plan. If the agent has the specified goal and belief in the rule in its goal and belief base, respectively, the plan in the rule is added to its plan base. The interpreter of the program tries rules till it finds one of which the conditions are fulfilled and then applies that rule. The rules are considered in the order at which they are represented. An implicit flow-of-control representation of the above example would be as follows.

```
handle incident <– explosion danger | extinguish fire
handle incident <– victim | save victim
handle incident <– fire | extinguish fire
```

The goals, beliefs and plans are explicitly represented and available to explain the agent's actions. For instance, if the agent has the beliefs *victim* and *explosion danger*, the first rule would be applied, and the goal *handle incident* and the belief *explosion danger* can be used to explain the agent's action of extinguishing the fire. However, information about priorities is only implicitly present in the representation. The following piece of code consists of the same three rules, but put in a different order.

```
handle incident <- fire | extinguish fire
handle incident <- victim | save victim
handle incident <- explosion danger | extinguish fire
```

If again the agent has the beliefs *victim* and *explosion danger*, the second rule would apply and the agent would (wrongly) start saving the victim. Thus, the order in which the rules are represented matters because it contains information about priorities. A programmer includes priority information into the program by properly ordering the rules, but priorities are nowhere explicitly represented. Just because the interpreter considers rules in a specific order, the desired behavior is generated. However, information about priorities might be needed in explanations, so it has to be made available elsewhere.

The same example could be represented with if-then-else rule, to make the flow-of-control explicit, for example as follows.

```
if (not explosion danger) then
   if (victim) then
```

```
      save victim
   end
   if (fire) then
      extinguish fire
   end
else if (explosion danger) then
   extinguish fire
   if (victim) then
      save victim
   end
end
```

Here, the priority rules are defined. If there is no explosion danger, saving a victim is executed before extinguishing the fire, and if there is, the fire is extinguished before saving the victim. The rule condition that applies to the current situation corresponds exactly to the priority rule used by the agent, which it thus has available for explanation. However, information about the agent's goals and beliefs can not directly be deduced from the behavior generation process, and needs to be documented elsewhere.

As said before, for the generation of agent behavior it is not necessary to explicitly represent both the agent's reasoning concepts (declarative) and their operationalization (imperative). However, explanations need information about both aspects. So to equip an agent programming language with explanation capabilities, the explicit part can easily be logged for explanation, but the implicit part has to be made explicit as well.

An agent programming language that tries to bridge the two paradigms is 2APL. It has declarative elements on the belief and goal level and imperative elements on the plan level. Therefore it seems a good candidate to implement the agent model. However, the example given above on rule priorities between plan generation rules is also applicable to 2APL and thus also in 2APL we have to model the agent carefully and make some decisions that are implicit in the deliberation cycle explicit.

Goals, plans and explanations in 2APL

In the abstract BDI model agents have both goals and plans. A goal describes a desired world state and a plan is a sequence of actions (or sub-plans). An agent needs to have at least one goal in order to generate actions, i.e. to make it proactive. In the example of section 3, the agent's main purpose is to handle the incident it is confronted with, so the node *handle incident* driving its behavior should be represented as a goal. The two descendant nodes of this main goal, *extinguish fire* and *save victim*, can be viewed as goals or plans, and their descendants as sub-goals and sub-plans, etc. In simple scenarios in which all of the agent's actions succeed and no unexpected events occur the difference between (abstract) plans and goals does not really matter, because their operationalization is similar in those situations. But in more challenging scenarios in which actions fail or plans are interrupted it does. In general, a goal is only dropped when it is achieved. In exceptional cases it can be explicitly dropped by the agent (as part of a plan). Plans, however, can be changed whenever circumstances require. If *extinguish fire* is represented as a goal, the goal should thus be explicitly dropped when the agent starts saving a victim because an agent can not simultaneously pursue the goals to extinguish a fire and save a victim. If it is represented as a plan, the (remaining) steps of the plan can remain in the plan base, but the execution will be postponed.

For instance in 2APL, which we used for our implementation, the differences between plans and goals become clear from the deliberation cycle. Goals are used in PG-rules, which have the form Goal <− Belief | Plan. In Figure 2 can be seen that they are tried in each deliberation cycle of the agent. Plans or actions that are part of a plan are only executed when they are in the plan-base of an agent. For plans, the interpreter knows which plan should currently be executed and manages the order in which plans are executed. But because all goals are tried to be achieved at every deliberation step in the program, for goals it has to be explicated in the program which ones the agent adopts and drops.

Thus if we implement most part of the model using goals, it should be carefully considered which goals the agent adopts and drops. The relation between goals has to be administered well as they tend to persist and interfere unexpectedly. However, because the goals are kept explicitly in the goal base they are very amenable to be used for explanations. In contrast, if we implement most part of the model as plans, we might easily loose track for which plan an action is performed. This is especially true when plans are interrupted. The difficulty of dealing with interrupted plans can be illustrated by the following example.

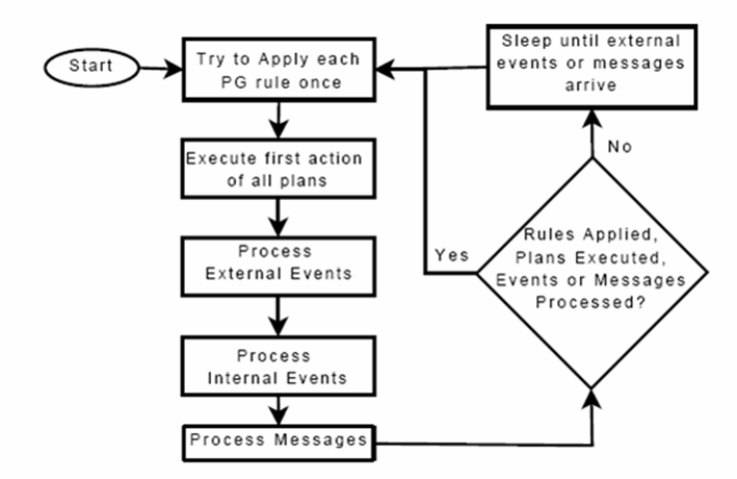

Figure 2. The deliberation cycle of a 2APL agent

In 2APL the interruption of a plan can be modeled by plan revision rules (PR-rules). A PR-rule that postpones the execution of a plan has the form Plan1 <− Belief | Plan2; Plan1. The given example could be modeled as follows:

```
OriginalPlan <- victim |
   saveVictim ;
   OriginalPlan
```

The rule states that whatever is the current plan of a firefighter, if it believes that there is a victim, he will first save the victim and then continue with its original plan. Notice that the plan *saveVictim* should also make sure that the belief that there is a victim is made false, otherwise the same rule is applied over and over again. In order to overcome the problems with updating, *atomic plans* can be used. The example would become:

```
OriginalPlan <- victim |
   <saveVictim; UpdateBelief(not victim)> ;
```

```
OriginalPlan
```

The plan between square brackets is interpreted as an atomic plan, which should be executed at once ensuring that the execution of this plan is not interleaved with the execution of the actions of other plans of the same agent. Through this expression the actual and the updating action can be connected to each other such that even an interruption can not separate them.

Given the above considerations, the 2APL rules can be used quite straightforward to both generate the desired behavior of the firefighter and keep track of its decisions through logging actions for the explanations.

Conclusions

In the present paper we have introduced an agent model able to generate behavior and explanations. We have shown that the generation of both behavior and explanations can be realized by implementing the model as a BDI agent in 2APL. The generated explanations aim to improve understanding of the agent's behavior, but also serve as the basis for explaining the interaction emerging in a group of agents.

The implementation of a self-explaining agent yielded some difficulties. To provide explanations introspection is required, but the current agent programming languages do not provide such possibilities. It is possible to artificially create introspection, but that results in only ad-hoc solutions. For a grounded solution, explanation facilities should be part of the platform design. In the platform, both concepts and their operationalization should be explicit. Moreover, the platform should ensure that explanation generation does not interfere with the actual program. Nevertheless, the paper shows that self-explaining agents based on a BDI theory can be developed using 2APL.

In order to use single agent explanations for explaining the complete simulation, a way to examine the set of explanations of individual agents is needed. Such an approach could distinguish different categories of explanation sets. For example, sets of individual behavior explanations can contain similar, different or contradictive explanations. One could then speak of agreement or cooperation, not understanding, and disagreement or hostility, respectively. Based on the categorization of a set of explanations, an explanation of the whole simulation can be constructed. Such simulation explanations distinguish between situations in which equal overall behavior is based on different individual motivations.

In the future, we will work on such an approach. We will also develop an explanation module in 2APL where an agent can update its executed actions with the corresponding explanatory elements. This enables the 2APL agent to reason about its goals and beliefs without interference with the execution of its actual action generating program. Furthermore, we want to make a complete implementation of a firefighting agent based on the input of experts. Several instantiations of this BDI-agent can then be connected to a simulation-based training system. Such a system can serve as a test-bed for approaches that explain simulations through self-explaining agents. Experiments could give insight into the usefulness of explanations for trainees, both of single agents and of teams of agents.

Acknowledgements

This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

References

BELLIFEMINE F, Caire G, and Greenwood D (2007) *Developing Multi-Agent Systems with JADE*. Wiley.

BORDINI R, Hubner J, and Wooldridge M (2007) *Programming multi-agent systems in AgentSpeak using Jason*. Wiley.

BRAZIER F, Dunin-Keplicz B, Jennings N, and Treur J (1997) Modelling multi-agent systems in a compositional formal framework. *Cooperative Information Systems* 6 (1).

CORE M, Traum T, Lane H, Swartout W, Gratch J, and Van Lent M (2006) Teaching negotiation skills through practice and reflection with virtual humans. *Simulation* 82 (11).

DASTANI M (2008) 2APL: a practical agent programming language. *Autonomous agents and multi-agent systems* 16 (3), pp. 214-248.

DE GIACOMO G, Lesperance Y, and Levesque H (2000) ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121 (1-2), pp. 109-169.

GOMBOC D, Solomon S, Core M G, Lane H C, and Van Lent M (2005) Design recommendations to support automated explanation and tutoring. *Proceedings of the Fourteenth Conference on Behavior Representation in Modeling and Simulation*, Universal City, CA.

JOHNSON W L (1994) Agents that learn to explain themselves. *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp 1257-1263.

LESSER V, Decker K, Carver N, Garvey A, Neiman D, Prasad N M, and Wagner T (2004) Evolution of the GPGP/TAEMS domain-independent coordination framework. *Autonomous agents and multi-agent systems* 9, pp 87-143.

NORLING E (2003) Capturing the quake player: using a bdi agent to model human behaviour. In Rosenschein J S, Sandholm T, Wooldridge M, and Yokoo M (Eds.) *Proceedings of AAMAS 2003*. pp 1080-1081.

RAO A and Georgeff M (1991) Modeling rational agents within a BDI-architecture. In Allen J, Fikes R, and Sandewall E (Eds.) *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, San Mateo, CA, USA, Morgan Kaufmann publishers Inc. pp. 473-484.

REYNOLDS C (1987) Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics* 21 (4) pp 25-34.

ROSENBLOOM P, Laird J, and Newell A (1993) *The Soar Papers: Readings on Integrated Intelligence*. MIT Press, Cambridge.

RUSSELL S and Norvig P (2003) *Artificial Intelligence A Modern Approach*. Pearson Education, Inc., New Jersey, USA, second edition.

VAN LENT M, Fisher W, and Mancuso M (2004) An explainable artificial intelligence system for small-unit tactical behavior. *Proceedings of IAAA 2004*, Menlo Park, CA, AAAI Press.

YE R, and Johnson P (1995) The impact of explanation facilities on user acceptance of expert systems advice. *Mis Quarterly* 19 (2) pp. 157-172.

Appendix

The following (pseudo-)code is an implementation of the firefighter agent in Figure 1:

```
fire :- smoke, sounds.

handle incident <- victim | save victim

handle incident <- explosion danger | extinguish fire

extinguish fire <- (water and fire) | use water

extinguish fire <- (no water and fire) |
   <use foam; UpdateBelief(no fire)>

use water <- (water and fire) |
   <go to house; UpdateBelief(at house)> ;
   <water on fire; UpdateBelief(no fire)>

save victim <- true |
   <go to house; UpdateBelief(at house)> ;
   <carry away; UpdateBelief(no victim)>

OriginalPlan <- victim |
   <saveVictim; UpdateBelief(not victim)> ;
   OriginalPlan
```