Goal Types in Agent Programming

[Extended Abstract] *

Mehdi Dastani
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands
mehdi@cs.uu.nl

M. Birna van Riemsdijk
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands
birna@cs.uu.nl

John-Jules Ch. Meyer
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands
jj@cs.uu.nl

ABSTRACT

This paper discusses three types of declarative goals and motivates their integration in logic-based agent-oriented programming languages. These goal types are perform goals, achieve goals, and maintain goals. A goal type is considered as a specific agent attitude towards goals. The semantics for each goal type is explained from an operational perspective. It is argued that the suggested semantics of the goal types ensure some desirable and expected properties.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial IntelligenceIntelligent agents, Languages and structures; I.2.5 [**Artificial Intelligence**]: Programming Languages and Software

General Terms

Theory, Languages

Keywords

Declarative goals, Agent programming languages, Agent types

1. INTRODUCTION

An essential characteristic of autonomous agents is their pro-active behaviour [9]. Such agents are assumed to have goals (or objectives) based on which they decide and perform actions. According to this view, an agent's goal denotes a state that the agent desires to realize by means of actions available to it. Different logics have been proposed to characterize goals, to represent and reason about them, and to specify their relations to other agent concepts such as beliefs and actions [2, 3, 6, 5]. These logics, which are mainly used to specify the behavior of autonomous agents, allow the specification of agents that have a certain attitude towards their goals. These attitudes, which are specified as logical axioms, are also known as agent types [3, 6, 4]. For example, an agent can be specified either to hold its goal until it has achieved it (blindly-committed agent type), or to drop the goal if it believes that it can *never* achieve it (single-minded agent type). It is important to note that the specification of many interesting agent types involves temporal aspects, such as in the case of the single-minded agent type, which involves the belief of an agent that it can *never* achieve some goal.

Motivated by these logics, many agent-oriented programming languages have been proposed to implement agents that can represent and reason about goals and are capable of generating plans for their goals [1]. One problem related to the implementation of agent types in agent programming languages is the implementation of the agent types that involve temporal aspects. Some agent types, such as the single-minded agent type, require that the agent is able to reason about its future execution steps to conclude that it believes the goals can *never* be achieved. Defining a programming construct that enables agents to perform such a reasoning task is, however, not an easy task, if not impossible. Moreover, the idea of implementing an agent with *one* specific attitude towards its goals is not attractive, especially when agents can have more than one goal. A more general agent-oriented programming language should therefore allow the implementation of agents that can have different attitudes towards different goals.

In order to implement goal-related agent types and to allow agents to have different attitudes towards different goals, existing programming languages have introduced the notion of goal types. For example, JACK [1] provides programming constructs to implement, among others, test, achieve, insist, and maintain goals, Jason [1] allows the implementation of achieve and test goals, and Jadex [1] provides programming constructs for achieve, query, perform and maintain goals. It is important to note that goal types with the same name captured by these agent-oriented programming languages may have a different meaning across these languages. Moreover, goals in Jadex are represented in XML using a label as the name of the goal, and a number of other parameters. In Jason and JACK, goals are created by generating a particular type of event. Such a notion of goal does not support

*A full version of this paper is available at `http://www.cs.uu.nl/~mehdi/publications.html`

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.

(logical) reasoning and therefore does not allow the derivation of subgoals. As we will explain, the derivation of goals is important in the context of agent-oriented programming. Finally, neither JACK nor Jadex provides the formal semantics of their goal types. This is in contrast with our proposal, as we aim at providing a formal semantics for the goal types. Below, we will explain our idea of these goal types and we will briefly address differences and similarities with comparable goal types from other agent-oriented programming languages. A detailed comparison between the goal types of the various languages is beyond the scope of this paper.

In this paper, we discuss three types of goals: perform goals, achieve goals, and maintain goals. In contrast to some other approaches, goals are here assumed to be represented as logical formulas with declarative semantics. This supports reasoning about goals which facilitates the generation of plans for subgoals. Moreover, we explain the *type* of a goal as an agent's attitude towards the goal. We do this with the aim of integrating them in logic-based agent-oriented programming languages. Therefore, the semantics of a goal type will be explained from an operational perspective. Note that the operational semantics perspective can directly be used to design a programming language that allows the implementation of the goal types. Unfortunately, the space limitation does not allow us to present the formal operational semantics of the goal types. Finally, we will discuss some properties of the goal types and show that they have certain desirable and expected properties.

The basic idea in this paper is that agents can have a set of goals based on which planning rules can be selected and applied to generate plans for the goals. In our framework, a planning rule indicates that a specific plan can achieve a certain goal. The application of a planning rule will add the plan of the planning rule to the plan base. Although it is possible to use a specific kind of planning rule for each goal type, we take a different approach and assume *one* kind of planning rule for every goal type. The advantage of such an approach is that it can reduce the number of rules which a programmer has to specify; a planning rule for a certain goal might be used in case this goal is an achieve goal or a maintain goal. As noted, the goals we consider in this paper are declarative goals, i.e., a goal can be represented as a logical formula denoting a desired state. An important advantage of declarative goals is that they allow reasoning, facilitating the generation of subgoals. This is especially advantageous in the context of the planning rules.

Consider, for example, "`FC and CC`" as the goal of an agent, where `FC` stands for having-fuel-in-car and `CC` for having-clean-car. We consider logical consequences of this goal as its subgoals, i.e., "`FC`" and "`CC`" are in this case the subgoals. The idea is, that for this complex goal "`FC and CC`", an agent programmer can implement either one single planning rule or two planning rules, i.e., one for each subgoal. The latter is however more generic, as the two rules can also be applied if the agent has, e.g., the goal `CC`, rather than the complex goal. The mechanism of deriving subgoals of a complex goal can be realized through logical inference mechanisms [7].

2. PERFORM GOALS

The idea of the perform goal is to allow the generation of a set of plans without demanding that the plans must reach the states denoted by the goals. The attitude of an agent toward a perform goal is thus to generate plans after which the goal will be dropped. For example, consider "`FC and CC`" as a perform goal of an agent that can generate the plan to refuel at the gas station `gs1` and the plan to clean the car in the car wash `cw1`. In this case, the agent will generate both plans after which it drops the goal entirely, regardless of the effect of those plans. However, if the agent does not have a planning rule to clean the car and can only generate the plan to refuel, then it will only generate the refuel plan after which the entire goal is dropped. This idea to drop a goal when plans are generated is practical, and similar ideas have been implemented in most agent-oriented programming languages, although sometimes under different names. For example, in JACK and Jason this type of goal is known as achieve goal.

As noted, in our approach the perform goals are represented as logical formulas. If the logical and declarative nature of the perform goals is not essential, then the question arises why we should not use *labels* instead of *logical formulas* to generate a set of plans. The reason to use logical formulas is to allow reasoning about the goals to decide which planning rule to apply. In our approach, the planning rules, which can be applied for all goal types, can be applied only if they are relevant, i.e., if the goal in their heads is logically entailed by an agent's goal. Using labels to represent perform goals requires different types of planning rules and a different procedure for rule application. Moreover, using logical formulas to represent a goal, in general, allows us to design planning rules for atomic or simple goals such that plans for more complex goals can be generated by means of plans for subgoals.

The perform goal can be enriched with a *redo* flag which may be true or false, as is also proposed in Jadex [1]. If the flag is false, then the perform goal will be removed after examining and applying all relevant planning rules. The removal of the perform goals is therefore independent of whether they are achieved or not. However, if the flag is true, then the perform goal remains in the goal base such that the applicable planning rules can be applied once again. This ensures that the plans associated with the perform goal will be generated and performed again. For example, consider a vacuum cleaner that has to clean a number of rooms repeatedly without the ability to check if a room is clean. The perform goal with a true flag can be used to model this type of behaviour. A consequence of this view is that an agent with a perform goal to which a false redo flag is assigned, can drop the goal if the relevant planning rules have tautologies as belief conditions. This is because in our framework a planning rule has a belief condition. A planning rule can only be applied if its belief condition holds.

3. ACHIEVE GOALS

The idea of an achieve goal is to reach the state denoted by it. The emphasis here is that the goal will not be dropped until the state denoted by it is achieved. This is in contrast with perform goals, which can be dropped once the plans generated for it have been executed, regardless of whether the perform goal was achieved. As for perform goals, an agent with an achieve goal will apply relevant planning rules to generate and execute plans. If the achieve goal is not reached after the execution of these plans however, it applies the planning rules again, hopefully generating different

plans, since the circumstances might have changed. Once the achieve goal is reached, it will stop generating and executing plans for this goal. For example, consider an agent with the achieve goal `FC` and two planning rules to generate either the plan to refuel at the gas station `gs1` or the plan to refuel at the gas station `gs2`. This agent will apply one rule and execute the plan to refuel at `gs1`. If the agent succeeds to refuel, the achieve goal will be dropped. Otherwise, it will apply the second rule to generate the plan to refuel at `gs2`. If both plans do not achieve the goal, then it will apply the rules again.

As suggested in [8] and implemented in Jadex, we assign a failure condition to each achieve goal to indicate when the agent should stop trying to achieve the goal and thus drop the goal. For example, `no-fuel-at-gs1-and-gs2` can be the failure condition of the achieve goal `FC`. A similar kind of goal type is introduced in Jadex and in JACK, though in JACK under the name *insist* goal. An achieve goal can thus be dropped under two circumstances: either if its failure condition becomes true, or if the state denoted by it is reached. In both cases, besides the removal of the achieve goal, the plans associated with it will be removed. A planning rule can be applied if the goal in the head of the rule is not achieved yet, if there is no plan for the same subgoal in the plan base, and if the failure condition does not hold.

Following this view, a goal with an achieve type and an impossible failure condition will not be removed from the goal base unless the state denoted by the goal is reached. Moreover, a goal with an achieve type such that the state denoted by the goal cannot be reached and its failure condition never becomes true can be replaced by the same goal with the perform type and a true redo flag. In both cases, relevant planning rules will be applied repeatedly.

4. MAINTAIN GOALS

The idea of a maintain goal is to ensure that a state holds and continues to hold. Plans should be generated and executed if the state denoted by the maintain goal is *threatened* not to hold, rather than waiting and taking action once the state does not hold. The condition under which the maintain goal is threatened not to hold will be called the maintain condition. The agent starts to generate and execute plans when the maintain condition becomes true. This triggering condition can be considered as an alarm to act in order to ensure the maintenance of the state denoted by the maintain goal. It should be noted that there might be no logical relation between the maintain goals and their triggering conditions. This relation depends usually on the application domain. However, we assume that in all agent configurations, if the triggering condition does not hold, then the maintain goal holds. Note that a maintain goal cannot be removed from the goal base.

For example, consider an agent with `FC` as a maintain goal. This means that the agent wants to maintain having a fueled car. In such a case, the maintain condition is the illuminated lamp warning of a shortage of fuel. The agent should in this case generate and execute a refuel plan. If the maintain condition continues to hold after the execution of the plan, because for example the tank station had no fuel, the agent may try to generate and execute another plan, e.g., to go to another tank station to refuel. If all plans are generated and the maintain condition still holds, then there are two options. The agent can either stop generating and executing plans since it has tried all plans, or it can continue to apply the planning rules once again, hopefully generating new plans this time. In order to allow both options, we add a *retry* flag, like in Jadex [1], to the maintain goals. If the flag is true, the agent will try to re-apply planning rules, otherwise it does not. Note that if the maintain condition holds, the state denoted by the maintain goal can either hold or not. A similar idea of maintain goals is implemented in Jadex and JACK, although in JACK the idea is that the maintain condition should hold during the execution of its related task. As a consequence, a goal with the maintain type for which the maintain condition is a tautology and the retry flag is set to false can be replaced by the same goal with the perform type having a false flag. In both cases, the relevant planning rules will be applied once.

5. CONCLUSION

In this paper, we have discussed three types of declarative goals for which we argued that they should be integrated in logic-based agent-oriented programming languages. We have already designed a simple agent-oriented programming language that allows the implementation of these goal types. This language uses a propositional language to represent the agent's goals and beliefs. The formal semantics of the programming language, and thus of the goal types, are defined and it is shown that this semantics has the properties as we have discussed in this paper. Unfortunately, because of the space limitation we could not present the details in this paper.

We plan to extend the logic-based agent-oriented programming language using a (computational) subset of the first-order predicate language instead of using a propositional language. In this way, declarative goals become more expressive. We also have a plan to start implementing an interpreter for an extended version of the programming language which we hope to be available soon.

6. REFERENCES

[1] R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, Berlin, 2005.

[2] C. Boutilier. Toward a logic for qualitative decision theory. In *Proceedings of the KR'94*, pages 75–86, 1994.

[3] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42, 1990.

[4] M. Dastani and L. van der Torre. Programming BOID-Plan agents: deliberating about conflicts among defeasible mental attitudes and plans. In *Proc. of AAMAS'04*, pages 706–713, New York, USA, 2004.

[5] J.-J. Ch. Meyer, W. van der Hoek, and B. van Linder. A logical approach to the dynamics of commitments. *Arificial Intelligence*, 113:1–40, 1999.

[6] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In *Proc. of KR'91*, 1991.

[7] M. B. van Riemsdijk, M. Dastani, F. Dignum, and J.-J. Ch. Meyer. Dynamics of declarative goals in agent programming. In *Proc. of DALT'04*, 2004.

[8] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proc. of KR'02*, 2002.

[9] M. Woolridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., 2002.