Goals in Agent Systems: A Unifying Framework

M. Birna van Riemsdijk
LMU Munich
Germany
riemsdijk@pst.ifi.lmu.de

Mehdi Dastani
Utrecht University
The Netherlands
mehdi@cs.uu.nl

Michael Winikoff
RMIT University, Melbourne
Australia
michael.winikoff@rmit.edu.au

ABSTRACT

In the literature on agent systems, the proactive behavior of agents is often modeled in terms of goals that the agents pursue. We review a number of commonly-used existing goal types and propose a simple and general definition of goal, which unifies these goal types. We then give a formal and generic operationalization of goals by defining an abstract goal architecture, which describes the adoption, pursuit, and dropping of goals in a generic way. This operationalization is used to characterize the discussed goal types.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents, languages and structures*; I.2.5 [**Artificial Intelligence**]: Programming Languages and Software; F.3.3 [**Logics and Meaning of Programs**]: Studies of Program Constructs; D.3.3 [**Programming Languages**]: Language Constructs and Features

General Terms

Theory, Languages

Keywords

Agent Programming, Goals, Formal Semantics

1. INTRODUCTION

A key property of software agents is that they are *proactive*, and consequently *goals* are a vital concept for agents. There has been much work in recent years on how goals can be used in agent programming frameworks to allow an agent to be proactive (see, e.g., [30, 11, 27, 20, 23, 2, 12, 15, 7, 4]). Agent programming frameworks in which goals play a central role, are generally so-called *cognitive* agent programming frameworks [26]. In these frameworks, agents are endowed with high-level mental attitudes such as beliefs, goals, and plans. The idea is then that an agent should try to reach its goals by executing appropriate plans, given its beliefs. Beliefs thus form the informational component of the agent, goals form the motivational component, and plans are the means for achieving the goals.

Various aspects of goals in agent systems have been investigated in recent years, with many different variations of goals appearing. We therefore believe that it is now useful to try and structure this collection of approaches. We do this by investigating commonalities among the various kinds of goals and proposing a *unifying perspective*. Such a unifying perspective provides a basis for a common understanding of goals in agent systems, and has, to the best of our knowledge, not yet been proposed in the literature. It facilitates a more structured and systematic study of goals in future research, and allows comparing different variations of goals by investigating how these instantiate the unifying framework. Moreover, it helps to identify aspects of goals that have been overlooked so far in the literature.

Cite as: Goals in Agent Systems: A Unifying Framework, M. Birna van Riemsdijk, Mehdi Dastani, Michael Winikoff, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.

Our investigation of the notion of goal focuses on two key aspects[1]:

1. What is a goal, and what types of goals are there?
2. How are goals operationalized in agent systems, i.e., how are goals pursued by agents?

We address the first issue by proposing a unifying abstract definition for the concept of goal (Section 2). While most people seem to have an intuitive idea of what a goal is, such a unifying view that encompasses the various kinds of goals has not been proposed yet. We arrive at this generic definition through analyzing important existing goal types and relating them by providing a taxonomy of goal types. The operationalization of goals is investigated in Section 3 by proposing an abstract goal architecture in which the most important aspects of operationalizing goals are captured. On the basis of this abstract goal architecture, we propose a new generic goal construct with formal semantics that can be used to model existing goal types.

Our work is from a language and architecture perspective, that is, we are interested in how goals are defined and used in agent programming, i.e., in languages and architectures. We do not address how goals are used in agent logics (see, e.g., [3]) or in philosophy [1]. While we thus do not investigate agent logics for defining goals, we do make use of simple logics as a tool for illustrating our ideas and making them more precise.

2. AN ABSTRACT VIEW OF GOALS

In this section, we briefly discuss what types of goals have appeared in the literature (Section 2.1), we propose a unifying abstract definition of a goal that aims at capturing the essential aspects of what a goal is (Section 2.2), and we conclude with a discussion thereof (Section 2.3). In this section, we are interested solely in getting a better high-level understanding of the notion of goal. In Section 3 we will address in more detail how goals are used in agent systems by investigating how agents pursue goals.

[1]For reasons of space, we will in each case provide only some example references, rather than trying to be complete.

2.1 Goal Types

Existing approaches to goals typically focus on one or several *types* of goals. One of the main distinctions among goals that has been made is the one between *declarative* and *procedural* goals [30, 11, 27, 12, 15]. Roughly speaking, a *procedural goal* is the goal to execute actions, and a *declarative goal* is the goal to reach certain states of affairs, i.e., it describes desired situations [26, Chapter 5]. Declarative goals have also been termed goals-to-be and procedural goals have been called goals-to-do [10, 27]. Declarative goals thus focus on the result of the execution of actions, while procedural goals focus on the actions themselves.

Other frequently used goal types are *achievement* goals [18, 30, 11, 27, 2, 6, 15], *maintenance* goals [2, 7, 6, 9], *perform* goals [2, 6] and *query* or *test* goals [18, 2]. Broadly speaking, an achievement goal is the goal to achieve a certain state of affairs (which is typically not reached yet), and a maintenance goal is the goal to maintain a certain state of affairs. A perform goal is the goal to execute actions, and a query or test goal is the goal to have a certain piece of information. Often, a query goal reduces to a simple test on the state of the agent. This is why some agent programming languages, which have a construct for testing [10, 27], do not call this a test *goal*. Achievement goals have received the most attention in the literature.

A further distinction between kinds of goals has appeared in work on goal-oriented requirements engineering: the distinction between *system* goals and *individual* goals [25]. System goals represent high-level goals the software system needs to achieve to fulfil the system requirements, while individual goals are goals of single actors in the system.

In order to get a better understanding of these goal types, and to pave the way for a unifying definition of what a goal is, we will discuss these goal types and relate them. We first consider the distinction between system goals and individual goals. A system goal can be viewed as an organizational goal, while an individual goal would be the goal of a single agent. We believe that the investigation of organizational goals is important, but in this paper we restrict our attention to individual goals and will from now on, when using the term "goal", mean "individual goal".

We continue our analysis by investigating whether there is a relation between the goal types that have appeared in the agent programming literature. As explained above, declarative goals are about situations or states, and procedural goals about actions. From the definitions of achievement and maintenance goals, we can see that these are declarative goals as both refer to situations, rather than to actions. A query goal can also be viewed as a declarative goal, as it is the goal to be in a situation in which a piece of information is available. Looking at the definitions of the procedural goal and the perform goal, we can see that they are identical, i.e., both are defined as "the goal to execute actions". Nevertheless, the notion of procedural goal is generally used in contrast with declarative goals, while the perform goal is a particular kind of concrete goal type. We thus have goals referring to states and goals referring to actions, i.e., referring to transitions between states, whereby the former can again be subdivided and contain goals referring to single states (achievement goals and query goals) and goals referring to multiple states (maintenance goals).

These considerations lead to the taxonomy of Figure 1. We depict the perform goal as a kind of procedural goal, to reflect that generally they are used on different abstraction levels. Also, extending this taxonomy with several kinds of perform goal is more natural than if we would have equated procedural goals and perform goals.

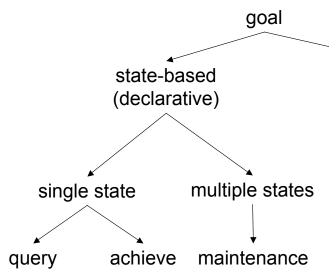

Figure 1: Goal Taxonomy

2.2 A Unifying Definition

Based on the discussion of Section 2.1, we propose the following as a first attempt to define what a goal is: a goal is *a mental attitude representing preferred progressions of a particular multi-agent system*. Defining a goal as a mental attitude accounts for the fact that we are interested in individual goals. Further, as the notion of progression encompasses both states and transitions between states,[2] defining a goal as representing preferred progressions accounts both for goals referring to actions as well as to those referring to states. Moreover, it accounts for goals referring to single states (in which case the preferred progressions would contain a state in which the goal is achieved), and for goals referring to multiple states. Although goals as used in agent programming commonly represent preferred progressions of the agent itself and possibly of its environment, goals may in general also refer to other agents. For example, an agent may have the goal that another agent believes a certain proposition.[3] Therefore, we define a goal as progressions of *a particular multi-agent system*, where this multi-agent system may consist of the agent itself, but may also contain an environment and possibly other agents.

In order to make the definition abstract and general enough, we have chosen the wording such that no technical terms are used that refer to a particular multi-agent system formalization or implementation. Nevertheless, if the multi-agent system is a computational system, the definition can be applied in a natural way. In that case, the preferred progressions are formed by the preferred computations of the multi-agent system. A goal is then a set of computations that the agent wants to realize. These computations can be represented by, e.g., computation traces or computation trees.

Our definition of goal naturally leads to a notion of goal fulfillment. As goals are *preferred* progressions of a system, they can be compared to *actual* progressions of the system, whereby a goal would be fulfilled, loosely speaking, if the actual progression is one of those that the agent prefers. This brings us to an aspect of goals that is not reflected in the discussion of goal types and that is not yet captured by our definition of goals.

If an agent has a goal, it will typically not sit still and hope for the system to progress as it prefers. On the contrary, having a goal generally means that the agent tries to realize this goal. In particular if the goal refers to the agent itself, the agent will typically have to do something to realize its goal. In order to take this into account, we refine our definition of goals as follows: a goal is *a mental*

[2]On Dictionary.com, a progression is defined as "a passing successively from one member of a series to the next", i.e., a repeated passing from one to the next.

[3]Note that a goal about another agent is still an individual goal and not a system goal, as a particular agent wants another agent to behave in a certain way.

attitude representing preferred progressions of a particular multi-agent system that the agent has chosen to put effort into bringing about. This is also what distinguishes a goal from a specification of desired behavior as used commonly in software verification. A goal, in contrast with a behavior specification, is something which an agent pursues *proactively*, which is facilitated by the fact that goals are a mental attitude which the agent can use in its decision making. We believe that this definition captures the essential aspects of what a goal is.

2.3 Discussion

Our definition of a goal aims to unify the various goal types that have appeared in the literature. However, this is not the only aim: now that we have a general definition, we can investigate whether there are other goal types than the ones discussed above that fit this definition.

As we consider a goal as a set of computation traces or a computation tree, it can naturally be specified by a logical formula. For example, we may consider (a variant of) Linear Temporal Logic (LTL) [8], which has computation traces as models, for the specification of goals. A trace in LTL is a sequence of system states $s_0, s_1, \ldots$, which can be enriched with the representation of which actions are executed to get from a state s_i to the next state s_{i+1}. A state then represents a configuration of the multi-agent system, typically consisting of a representation of the environment and of the local states of the agents.

Loosely speaking, an achievement goal for ϕ can then be represented by the LTL formula $\Diamond\phi$, specifying that the agent prefers computations in which ϕ holds eventually. The formula ϕ may be an atomic proposition p, indicating that the property should hold in the environment, but might also refer to the beliefs of the agent itself or of other agents (which could be expressed by a formula $\Diamond\mathbf{B}_i p$, where i is the name of an agent[4]). A maintenance goal for ϕ can be represented by $\Box\phi$, specifying that ϕ should always hold. A perform goal of agent i to do the action a could be represented by $\Diamond done_i(a)$,[5] specifying that the agent wants to have executed the action a eventually, and a query goal for ϕ could be represented by $\Diamond(\mathbf{B}_i\phi \vee \mathbf{B}_i\neg\phi)$, specifying that agent i wants to know (believe) the truth of ϕ, i.e., wants to believe that ϕ holds or to believe that ϕ does not hold.

By looking at goals in this way one can see that in principle *any* property characterizing sets of computations can be a goal. In particular, one may consider other variants of the goal types of Figure 1. For example, an agent may want to maintain a certain situation for a limited time period, rather than throughout the execution of the agent. This type of maintenance goal could be expressed by $\phi \mathsf{U} \psi$, which informally means that ϕ should be maintained until ψ becomes true. Another type of maintenance goal is what is called a reactive maintenance goal [7]: the agent waits for things to go wrong and then fixes them. This type of maintenance goal can be expressed by a formula of the form $\Box(\neg\phi \rightarrow \Diamond\phi)$.

Furthermore, one may consider various variants of procedural goals, e.g., expressing that if an action a is executed by agent i, action b should eventually also be executed ($done_i(a) \rightarrow \Diamond done_i(b)$), rather than just saying that some (sequence of) actions should be executed. A final interesting goal type that has not been addressed often in the literature so far, is goals about other agents. We see this aspect as orthogonal to the distinction between declarative and procedural goals, i.e., both "internal" goals (goals referring only to the agent itself and possibly to its environment) and "external" goals (goals referring to other agents) may be procedural or declarative.

While in principle any property characterizing sets of computations can be viewed as a goal, this does not mean that it is feasible to endow the agent with mechanisms for the effective realization of each of these goal types. In Section 3, we present an abstract goal architecture for operationalizing goals, that is general enough to capture a range of goal types that have been discussed here.

Another point for discussion is the relation between plans and procedural goals. In some literature, plans have been put on a par with procedural goals [30; 26, Chapter5]. However, taking into account our analysis in Sections 2.1 and 2.2, we suggest that procedural goals are not necessarily the same as plans, as the former are in general more high-level. A procedural goal does not necessarily represent a particular complete plan, i.e., it may, for example, express that if some action is executed, another action should also be executed, or it may express that either action a or action b should be executed ($\Diamond(done(a) \vee done(b))$, etc. Such procedural goals can then be fulfilled, just like declarative goals, through the adoption of a concrete plan. That is, a plan should be viewed as a means for fulfilling a goal, be it a procedural or a declarative goal.

3. OPERATIONALIZING GOALS

In Section 2, we have discussed what a goal essentially is, and provided a unifying abstract definition of a goal. The purpose of a proactive agent is to put effort into bringing about its goals. In principle, an agent may consist of only a set of plans that are programmed such that they bring about its goals, i.e., to realize a preferred progression of the system. The goals are then nothing more than a specification of the desired behavior of the agent.

However, such a simple model of goal-directed behavior is often not flexible enough (see also [26, Chapter 5]) if agents are required to operate in dynamic environments that may also contain other agents. For example, it may be the case that a plan fails, e.g., due to environmental circumstances that were not foreseen at design time.

More flexible goal-directed behavior can be obtained if goals are represented *explicitly* in agent programs. In this section, we discuss how explicitly represented goals can be used effectively to guide the agent's behavior towards the realization of its goals. That is, the goals should be operationalized such that the progression of the agent fulfills the goals, if at all possible.

In Section 3.1, we discuss aspects of operationalizing goals, and give a simple and generic abstract goal architecture. Section 3.2 presents a formalization of this architecture, and in Section 3.3 we show how the goal construct proposed in 3.2 can be used to model a range of goal types.

3.1 An Abstract Architecture

Many different approaches for explicitly representing goals in agent programming frameworks have been proposed, with each of them focussing on particular aspects of the issue. Here, we propose an abstract view on how goals are used in agent programming frameworks. Our aim is to capture the essential aspects of operationalizing goals in agent programming frameworks, abstracting away from particular goal types.

The basic schema that most goal-oriented agent programming frameworks adhere to, is that goals are *adopted* for some reason, they are used for *generating* the agent's *plans*, and then they may

[4]We will sometimes omit the name i, here and in other kinds of formulas, for reasons of presentation.

[5]Informally, the formula $done_i(a)$ holds in a state s_i if the action a has been executed by agent i to get from s_{i-1} to s_i. This might also be extended to arbitrary plans (sequences of actions) π, where the formula $done_i(\pi)$ holds in a state s_i if the actions of π have been executed one by one to get from s_{i-k} (where k is the length of π) to s_i.

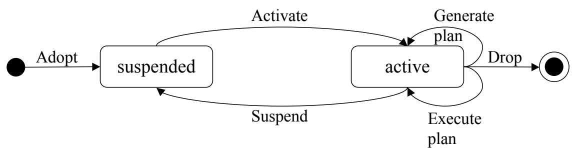

Figure 2: Goal Abstract Architecture

be *dropped* for several reasons. This schema does not follow immediately from our discussion of goals in Section 2, and one might also consider other ways of operationalizing goals. However, existing agent programming frameworks typically implement (parts of) this schema, and using these kinds of mechanisms for operationalizing goals has proven to be effective.

An explicit representation of goals allows the agent to adopt and handle new goals that may present themselves because of, e.g., requests of other agents or of the owner of the agent. This thus provides for added flexibility, compared to a situation in which goals are not represented explicitly.

Once a goal is adopted, it can be used by the agent to generate plans that are aimed at fulfilling the goal. However, if a goal is adopted, the agent will in general not start pursuing the goal immediately. It can, for example, be the case that the agent is currently pursuing other goals, and does not have the resources to pursue the adopted goal right away. We call goals that the agent is not currently pursuing *suspended* goals (see also [2]).[6] Once an agent starts pursuing a suspended goal, the goal becomes *active*. If a goal is active, it can be used for generating plans for fulfilling it. Once a plan is generated, it can be executed. In some cases, e.g., because a more important goal has just been adopted that needs to be pursued, an active goal may have to be suspended again. Alternatively, an agent may drop an active goal, e.g., because it is deemed to be impossible or because it is no longer relevant (e.g., because it has been achieved). This abstract architecture is depicted in Figure 2.

A similar goal architecture has been described by [2] in the context of the Jadex framework. A key difference between our work and theirs is that our work is more abstract and aims at unifying various approaches to operationalizing goals that have appeared in the literature. By contrast, the Jadex framework was proposed as one particular approach to the operationalization of goals. Moreover, we provide a formalization of this abstract architecture that can be instantiated to model various goal types. In the Jadex framework, no generic formalization is provided, and the goal architecture is specialized for each goal type separately without investigating a common representational framework of goals.

We believe that the goal architecture of Figure 2 captures the essential aspects of operationalizing goals in agent programming frameworks, as existing goal-oriented agent programming frameworks generally instantiate parts of this architecture, or refine it. An aspect of the architecture which is not always provided in existing agent programming frameworks, is a mechanism for goal adoption. If this is the case, the agent is endowed with a set of (suspended) goals at start-up (see, e.g., [27, 5]). An example of a framework that does allow goal adoption through communication is [14].

Also, the distinction between suspended and active goals is not always made explicit. In some frameworks, goals are suspended automatically if a plan for pursuing the goal has finished and the goal cannot be dropped [27, 5]. An example of a framework that does incorporate suspension explicitly, is the Jadex framework [2]. We believe that making the distinction between suspended and active goals explicit in the goal architecture, highlights and clarifies a mechanism that is often implemented in agent programming frameworks in some way, but is generally not explicitly addressed as such. In Jadex, also an additional distinction is made between so-called option goals, and suspended goals. In that framework, suspended goals are goals that should not be activated. In order to activate a suspended goal, it should first become an option, and then it may be activated. However, for our purposes this distinction is not necessary: it is simpler to just consider all suspended goals as candidates for being activated.

In order to generate a plan for trying to fulfill a goal, the agent typically performs means-end reasoning. Several mechanisms can be used for performing means-end reasoning. A common mechanism, which is used in many agent-oriented programming languages, including JAM, JACK, AgentSpeak(L), 3APL, CAN, and Jadex, is to have a library of plan "recipes" where each plan recipe specifies in which circumstances the plan may be used, and for the fulfillment of which goal the plan can be used. Another mechanism that may be used is planning from first principles. The two mechanisms can also be used in combination [19]. In the abstract framework that we consider in this paper, we do not require any particular representation of the plan, nor do we constrain how it is derived. The plan could be a single action, a sequence of actions (a plan in the classical sense), a program, or a BDI plan set.

3.2 Formalization

In this section, we make the ideas presented in Section 3.1 more precise by providing a formalization of the abstract goal architecture. The basic idea is that a goal contains a set of conditions that govern the transitions SUSPEND, ACTIVATE, and DROP (as depicted in Figure 2) which change the state of a goal. This representation allows for the framework to be instantiated with a range of different goals types, by varying the conditions under which the transitions can take place. The instantiation of the framework with particular goal types will be discussed in Section 3.3.

As will become clear in Section 3.3, it is useful to make a distinction between conditions that are checked during plan execution, i.e., when a non-empty plan is associated with a goal, or only when a plan completes or has not been generated yet. In order to be able to check this, we have to represent which plan aimed at fulfilling the goal is currently associated with the goal. Also, we need to represent in which state a goal currently is (in the ACTIVE or SUSPENDED state).

We thus define a generic goal construct as $\mathsf{g}(C, E, S, \pi)$, where C and E are sets of conditions of the form $\langle condition, action\rangle$ with $action \in \{\textsc{Suspend}, \textsc{Activate}, \textsc{Drop}\}$, $S \in \{\textsc{Active}, \textsc{Suspended}\}$ represents the state of the goal, and π is the plan of the goal. The conditions in E are tested when the plan is empty, and those in C are tested only when the plan is not empty. If a condition should be tested in both situations it is placed in C *and* in E. We use ϵ to denote the empty plan. We do not define a particular language for expressing conditions, but we assume that conditions can be checked on the beliefs of the agent. In order to express that a condition c is true with respect to the agent's beliefs B, we use the notation $B \models c$.

As explained in Section 3.1, we do not require a particular mechanism for doing means-end reasoning (*mer* for short), nor do we require a particular representation of plans. In our formalization,

[6]The notion of suspended goals might seem contradictory with our definition of goals of Section 2.2, since goals are mental attitudes that the agent *puts effort into bringing about*. However, the idea is that suspended goals become active as soon as possible and needed/useful. That is, suspended goals are in principle also goals that the agents puts effort into bringing about, but temporarily suspending them is useful for operationalization.

we only require the existence of a function *mer* which takes a goal and the agent's beliefs, and returns a plan $\pi \neq \epsilon$. All we require of π is that we can execute it (step by step) and that executing it one step results in a residual plan π' (where π' may be the empty plan ϵ).

We now give the operational semantics for this goal construct in terms of a set of transition rules. In these transition rules, the state of an agent is a tuple $\langle B, G\rangle$, where B is the agent's belief base and G is its goal base consisting of a set of goals. A transition rule represents how the state of the agent can evolve one step as represented by the transition below the line, under the conditions expressed above the line.

We define how a goal base containing a set of goals evolves, by means of defining how single goals evolve. That is, we define when transitions $\langle B, g\rangle \rightarrow \langle B', g'\rangle$ where g is a single goal may occur, and lift this to transitions over the goal base by means of the following transition rule. This rule indicates that if an agent can make a transition by processing one of its goals, then the agent can process its goal base in which only the single goal is processed.

$$\frac{\langle B, g\rangle \rightarrow \langle B', g'\rangle \quad g \in G \quad G' = (G \cup \{g'\}) \setminus \{g\}}{\langle B, G\rangle \rightarrow \langle B', G'\rangle}$$

In this paper, we abstract away from the mechanism used for adopting new goals. We define goal adoption formally by assuming that the initial goal base of an agent includes a set of goals of the form $\mathsf{g}(C, E)$, representing the goals that the agent may adopt during its execution. If such a goal is adopted, its state will become SUSPENDED as suggested in Figure 2, and the plan assigned to it will be an empty plan.

$$\frac{}{\langle B, \mathsf{g}(C, E)\rangle \rightarrow \langle B, \mathsf{g}(C, E, \text{SUSPENDED}, \epsilon)\rangle}$$

The next two rules specify how a goal can move from SUSPENDED to ACTIVE. The first rule indicates that a suspended goal of an agent can be activated if the plan associated with the goal is non-empty and the agent believes the corresponding activation condition c in $\langle c, \text{ACTIVATE}\rangle \in C$. The second rule is analogous, but covers the case for a condition c in $\langle c, \text{ACTIVATE}\rangle \in E$, in which case the plan should be empty.[7] The result of activating a suspended goal is that the goal is in the active state and can be executed, as also suggested in Figure 2.

$$\frac{\pi \neq \epsilon \quad \langle c, \text{ACTIVATE}\rangle \in C \quad B \models c}{\langle B, \mathsf{g}(C, E, \text{SUSPENDED}, \pi)\rangle \rightarrow \langle B, \mathsf{g}(C, E, \text{ACTIVE}, \pi)\rangle}$$

$$\frac{\langle c, \text{ACTIVATE}\rangle \in E \quad B \models c}{\langle B, \mathsf{g}(C, E, \text{SUSPENDED}, \epsilon)\rangle \rightarrow \langle B, \mathsf{g}(C, E, \text{ACTIVE}, \epsilon)\rangle}$$

The next two rules specify how a goal can move from ACTIVE to SUSPENDED. They are analogous to the rules for moving in the other direction. The result of suspending a goal is that the goal is in a suspended state and cannot be executed, as also suggested in Figure 2.

$$\frac{\pi \neq \epsilon \quad \langle c, \text{SUSPEND}\rangle \in C \quad B \models c}{\langle B, \mathsf{g}(C, E, \text{ACTIVE}, \pi)\rangle \rightarrow \langle B, \mathsf{g}(C, E, \text{SUSPENDED}, \pi)\rangle}$$

$$\frac{\langle c, \text{SUSPEND}\rangle \in E \quad B \models c}{\langle B, \mathsf{g}(C, E, \text{ACTIVE}, \epsilon)\rangle \rightarrow \langle B, \mathsf{g}(C, E, \text{SUSPENDED}, \epsilon)\rangle}$$

Note that when a goal is suspended, the associated plan is also suspended (a detailed mechanism for goal suspension and resumption is described in [21]). When a (suspended) goal is activated again, the associated plan is also activated, which allows the agent to continue executing the plan where it stopped when it was moved to the SUSPENDED state. It might also be the case that the suspended plan is no longer useful. In this case, the agent will want to drop the plan and generate another one. This will be addressed in the sequel.

[7]Note that merging transition rules for C and E is not possible as we also need to check whether the plan is empty or not.

The following two transition rules specify dropping of an active goal. An agent can drop its active goal if it believes the corresponding drop condition c in $\langle c, \text{DROP}\rangle \in C$ or in $\langle c, \text{DROP}\rangle \in E$. Note that if a condition in C is used to drop a goal, the plan related to the goal is not executed completely, in which case the goal together with its not fully executed plan is removed from the agent's goal base. Moreover, note that the conditions for dropping a goal determine the level of commitment an agent has towards its goal [3, 30]. Typically, an agent is expected to have some commitment to its goals, which means that it should have a good reason for dropping a goal (e.g., having achieved the goal or believing it is not achievable anymore).

$$\frac{\pi \neq \epsilon \quad \mathsf{g}(C, E, \text{ACTIVE}, \pi) \in G \quad \langle c, \text{DROP}\rangle \in C \quad B \models c}{\langle B, G\rangle \rightarrow \langle B, G \setminus \{\mathsf{g}(C, E, \text{ACTIVE}, \pi)\}\rangle}$$

$$\frac{\mathsf{g}(C, E, \text{ACTIVE}, \epsilon) \in G \quad \langle c, \text{DROP}\rangle \in E \quad B \models c}{\langle B, G\rangle \rightarrow \langle B, G \setminus \{\mathsf{g}(C, E, \text{ACTIVE}, \epsilon)\}\rangle}$$

Finally, we come to transition rules for plan generation and plan execution. These rules need to be given a lower priority than the condition rules above, as an agent should not continue executing the plan of goal that should be dropped or suspended. This is done by adding an additional condition to the premise of the rules below that specifies that the goal has no conditions that hold. This formalization makes sure that plan generation and plan execution can occur for active goals, unless one of the conditions of the goal specifies that it should move to another state.

The first rule states that if an agent has an active goal with empty plan (ϵ) and there is no suspend or drop action for which the agent believes the corresponding condition, then the agent can perform means-end reasoning to generate a plan for the active goal. We assume that means-end reasoning takes into consideration the existing goals and their assigned plans when it generates a new plan, in order to ensure that the newly generated plan does not interfere with the plans of the existing (active) goals.

$$\frac{\neg\exists\langle c, a\rangle \in E \ .\ (B \models c) \wedge (a \neq \text{ACTIVATE})}{\langle B, \mathsf{g}(C, E, \text{ACTIVE}, \epsilon)\rangle \rightarrow \langle B, \mathsf{g}(C, E, \text{ACTIVE}, \mathit{mer}(g, B))\rangle}$$

Our next transition rule states that the plan assigned to an active goal can be executed.

$$\frac{\langle B, \pi\rangle \rightarrow \langle B', \pi'\rangle \quad \neg\exists\langle c, a\rangle \in C \ .\ (B \models c) \wedge (a \neq \text{ACTIVATE})}{\langle B, \mathsf{g}(C, E, \text{ACTIVE}, \pi)\rangle \rightarrow \langle B', \mathsf{g}(C, E, \text{ACTIVE}, \pi')\rangle}$$

Note that when a plan completes, a new plan is generated by default, because the goal is only moved out of the active state if the relevant conditions apply. This is important to ensure that an agent keeps putting effort into reaching its goals, if this is still useful. This will be discussed in more detail in Section 3.3. Also, note that we assume that an additional transition system is defined to derive transitions for the execution of plans $\langle B, \pi\rangle \rightarrow \langle B', \pi'\rangle$ (see, for example, [30]). We also assume that it is not possible to derive $\langle B, \epsilon\rangle \rightarrow \langle B', \pi\rangle$, i.e., once a plan has completed it cannot progress any further.

A final transition rule deals with the case that a plan for an active goal is no longer useful. This can be the case if a suspended goal with a non-empty plan is activated, as already mentioned above.

Also, it may be the case that a plan that is being executed gets stuck, e.g., because the first action of the plan cannot be executed due to environmental circumstances. A typical way of dealing with this, is to drop a plan so that a new plan can be generated [28]. This is modeled in the next transition rule. The rule expresses that a plan can be dropped if it cannot be executed any further. In some cases, an agent may want to drop its plan even if it can still be executed. This can be incorporated by adapting the premise of the transition rule.

$$\frac{\pi \neq \epsilon \quad \langle B, \pi \rangle \not\to \langle B', \pi' \rangle}{\langle B, \mathsf{g}(C, E, \textsc{Active}, \pi) \rangle \to \langle B, \mathsf{g}(C, E, \textsc{Active}, \epsilon) \rangle}$$

Summarizing, we have specified for each transition depicted in Figure 2 one or two transition rules. These express exactly when a goal can move from one state to another (as governed by the conditions of the goal), and when a plan can be generated or executed.

3.3 Instantiation with Goal Types

In this section, we investigate how the proposed goal construct of Section 3.2 can be used to model a range of goal types. The goal types we consider are those of Figure 1. They can be modeled by making particular choices for the conditions of the goal. As will become clear, the instantiations are such that the agent puts effort into bringing about its goals, thereby doing what it can to make sure that its actual progression is one that is preferred.

3.3.1 Achievement Goals

The first goal type we address is the achievement goal. In Section 2.1, we have defined an achievement goal as the goal to achieve a certain state of affairs ϕ ($\Diamond\phi$ in LTL). The idea is that the instantiation of the abstract goal architecture with an achievement goal to reach a state of affairs ϕ, should facilitate the effective fulfillment of this goal. For this, we at least need ϕ to be purely propositional, i.e., ϕ should not contain temporal operators.

In the literature, an achievement goal in agent programming frameworks typically contains at least a description of the state of affairs that is to be achieved. This description is often called the *success condition* s (which would be equal to ϕ, in case of the example above). In addition, achievement goals are sometimes endowed with a so-called *failure condition* f [30, 2, 6]. If the failure condition comes to hold, the success condition is assumed not to be achievable anymore.

The purpose of the success condition is to allow the agent to generate plans for reaching the state of affairs represented by the condition. Moreover, the success condition is used to monitor the execution of plans generated for achieving the success condition. In particular in dynamic environments, it may be the case that a plan that was meant to achieve the success condition, fails to achieve it. The fact that the success condition is represented, then allows the agent to generate another plan for trying to achieve it. This idea has been referred to as the decoupling between plan execution and goal achievement [30]. If a plan *is* successful in reaching the success condition, the goal has been achieved (i.e., the agent has produced a trace in which $\Diamond\phi$ holds). In that case, the goal is no longer of use, and is typically dropped. Another reason to drop the goal is if the failure condition becomes true. In that case, the goal is assumed not to be achievable anymore, which means it does not make sense to hang on to the goal.[8]

In our abstract goal architecture, we model an achievement goal with success condition s and failure condition f as follows.

$$\mathsf{A}(s, f) \equiv \mathsf{g}(\{\langle s \vee f, \textsc{Drop}\rangle\}, \{\langle s \vee f, \textsc{Drop}\rangle, \langle true, \textsc{Activate}\rangle\})$$

The conditions express that if the failure condition or the success condition come to hold (either during plan execution or afterwards), the goal should be dropped. Moreover, we need to add a condition $\langle true, \textsc{Activate}\rangle$ in order to activate an adopted achievement goal. The activation is not governed by any special conditions on the belief base, which is why we use the condition $true$. We add it to the E component, as an adopted achievement goal which is added to the SUSPENDED state will have an empty plan.

Modeling achievement goals in this way makes sure that the agent keeps trying to realize its achievement goal (once it has been activated). It will only stop trying if it has reached the goal (note that it is dropped immediately after activation if the success or failure condition is already true upon activation), or if it is no longer reachable. In this way, it does what it can to make sure that its actual progression is one that is preferred, i.e., one in which the success condition of the achievement goal is fulfilled. However, whether the agent indeed does fulfill the achievement goal also depends on the effectiveness of the mechanism for plan generation and on environmental circumstances.

In the above definition of an achievement goal, the goal is dropped regardless of whether the associated plan is executed completely (as the condition $\langle s \vee f, \textsc{Drop}\rangle$ occurs both in C and in E). One may redefine the achievement goal to force the complete execution of plans even when an agent comes to believe the success or failure conditions during execution of a plan for the achievement goal. This version of achievement goals can be defined in a simple way in our framework as follows.

$$\mathsf{A'}(s, f) \equiv \mathsf{g}(\{\}, \{\langle s \vee f, \textsc{Drop}\rangle, \langle true, \textsc{Activate}\rangle\})$$

3.3.2 Perform Goals

The second goal we address is the perform goal, which was defined in Section 2.1 as the goal to execute actions. This type of goal is typically used in combination with plan recipes for means-end reasoning. The plan recipes then specify which plan may be executed for a particular perform goal. The perform goal itself is usually only a name by means of which the appropriate plan recipe can be selected. In such a setting, it is natural to interpret the perform goal as having succeeded in case one of the plans of the corresponding plan recipes executes completely (see also [2]).[9] If P is the perform goal under consideration and $\pi_1, \ldots, \pi_n$ are the plans of the plan recipes corresponding to P, this perform goal could be expressed in LTL as $done(\pi_1) \vee \ldots \vee done(\pi_n)$ (assuming that the logic allows us to use complex plan statements as the arguments of the $done$ operator).

This behavior can be modeled in our abstract goal architecture by assuming that an agent's belief base is updated with $succeeded(\mathsf{g})$ or $failed(\mathsf{g})$ after the plan associated to a goal g has been executed completely, expressing that the plan has been executed completely without exceptions occurring, or that it has not, respectively. We also assume that these formulas are removed from the agent's belief base as soon as a new plan for the same goal is generated. We can then specify by means of the condition $\langle succeeded(\mathsf{g}), \textsc{Drop}\rangle$ that a perform goal should be dropped when a plan has been executed successfully for this goal.[10] This condition is placed in the

[8]It is the job of the programmer to choose appropriate failure conditions.

[9]This is a different interpretation than the one of [6], in which a perform goal is considered to have been successful as soon as a plan is selected for it. However, considering the discussion in Section 2, we maintain that our interpretation is more appropriate.

[10]Note that if a perform goal has been adopted twice, both goals

set E, as it should not be checked during plan execution (in that case the condition $\mathit{succeeded}(\mathsf{g})$ would always be false). The transition rules for plan generation and plan execution make sure that new plans are generated until the goal is dropped because a plan has been executed completely. As in the case of achievement goals, the agent does what it can to make sure that its actual progression is one that is preferred, i.e., one in which a plan corresponding to the perform goal is executed completely.

$$\mathsf{P} \equiv \mathsf{g}(\{\}, \{\langle \mathit{succeeded}(\mathsf{g}), \textsc{Drop}\rangle, \langle \mathit{true}, \textsc{Activate}\rangle\})$$

Note that perform goals have no special argument associated with it because the only thing that needs to be done is to drop the goal when its generated plan is completely executed. Further, a perform goal is a special case of the second version of the achievement goal, where the failure condition is just false and the success condition is $\mathit{succeeded}(\mathsf{g})$, i.e., where $\mathsf{P} \equiv \mathsf{A'}(\mathit{succeeded}(\mathsf{g}), \mathit{false})$.

3.3.3 Maintenance Goals

In Section 2.1, a maintenance goal has been defined as the goal to maintain a certain state of affairs, which could be expressed by the LTL formula $\Box\phi$ (if ϕ is the state of affairs to be maintained). The ideal situation, given this goal, would thus be that the agent prevents ϕ from becoming false. This may not only involve taking action, but the agent may also need to refrain from executing certain actions. Realizing this kind of behavior involves advanced reasoning mechanisms [7, 9] that are not easily represented by our conditions of goals.

A kind of maintenance goal that can be operationalized easily within our framework, is the reactive maintenance goal. In the case of a reactive maintenance goal, the agent takes action if a maintenance goal is violated in order to re-establish the condition that is to be maintained. This can be represented by the LTL formula $\Box(\neg\phi \rightarrow \Diamond\phi)$, and modeled in our goal architecture as follows.

$$\mathsf{M}(\phi) \equiv \mathsf{g}(\{\}, \{\langle \phi, \textsc{Suspend}\rangle, \langle \neg\phi, \textsc{Activate}\rangle\})$$

According to this definition of the maintenance goal, the agent should activate its maintenance goal when the agent believes the condition ϕ is false. An agent can suspend a maintenance goal only if its corresponding plan is executed completely and the agent believes that condition ϕ now holds. The fact that maintenance goals are not dropped, makes sure that they can be used throughout the execution of the agent. This is in line with the $\Box$ operator of the LTL property, which expresses that a property should hold throughout the execution of the agent.

The above definition of maintenance goal can be easily adapted to make it correspond to a maintenance goal $(\Box(\neg\phi \rightarrow \Diamond\phi))\mathsf{U}\psi$, which expresses that ϕ should be re-established if it becomes false, until ψ comes to hold (see Section 2.3). All we need to do is to add a condition $\langle\psi, \textsc{Drop}\rangle$ both to C and E, and to change $\langle\phi, \textsc{Suspend}\rangle$ to $\langle\phi \wedge \neg\psi, \textsc{Suspend}\rangle$. We also need to have $\langle\psi, \textsc{Activate}\rangle$ in C and in E. This is because we cannot drop a suspended goal, so if ψ becomes true while the goal is suspended, we activate it first, and then drop it.

3.3.4 Query Goals

In Section 2.1, a query/test goal is the goal to have a certain piece of information. This can be concretized in LTL as $\Diamond(\mathbf{B}\phi \vee \mathbf{B}\neg\phi)$, which expresses that the agent wants to find out whether a formula ϕ is true or false. In order to model this goal, we define $B \models \mathit{known}(\phi)$ as $B \models \phi$ or $B \models \neg\phi$ to indicate that an agent knows the truth of ϕ. A query/test goal can then be modeled as a kind of achievement goal, where $\mathit{known}(\phi)$ is the success condition.

will be dropped if a plan for one of them has been executed. If this is not desired, it would require distinguishing between instances when adopting goals, e.g., by introducing a unique identifier during adoption.

4. CONCLUSION AND FUTURE RESEARCH

In this paper, we have investigated the notion of goal as used in agent systems by focussing on two issues: what is a goal, and how are goals operationalized? In order to address the first issue, we have proposed a unifying abstract definition of the notion of goal on the basis of an analysis of commonly-used existing goal types. We have addressed the second issue by proposing a unifying abstract goal architecture with a formalization that we have used to model a range of goal types. Through this, we have gotten a better understanding of what are essential aspects of goals in agent systems. In the following, we discuss an important aspect of goals that we have not covered in detail in this paper, and point to directions for future research.

Our formalization of the abstract goal architecture uses simple conditions on the beliefs of the agent to govern the state changes of the goals. We have shown that this framework is general enough to model a range of goal types. In future work, we will explore the limits of our framework in more detail, e.g., by investigating whether variants of these goal types are also easily expressible. Examples of such variants are a repetitive achievement goal of achieving ϕ every day, or a combination of a perform or achievement goal and a maintenance goal where one should maintain ψ while achieving ϕ (expressible in LTL as $\Diamond\phi \wedge ((\Box\psi)U\phi)$). An apparent limitation of the framework in its present form is that it does not capture more advanced usages of goals in which sophisticated reasoning techniques are used to determine how to pursue goals. In the sequel, we briefly mention some of these kinds of reasoning techniques and indicate how they could be incorporated into our formal framework. In future work, we will investigate the incorporation of these techniques in more detail.

In general, various advanced reasoning techniques could be used at any stage of goal pursuit. In particular, in order to determine whether a goal can move from SUSPENDED to ACTIVE, an agent can use reasoning to determine whether the goal that is to be activated is "compatible" with the existing active goals. A goal is compatible with a set of other goals, if the agent can simultaneously pursue all of these goals without undesirable interferences occurring. For example, interference may occur because of resource conflicts, or because actions of different plans interfere by, e.g., undoing each other's results.

In order to represent whether goals (and their corresponding plans) interfere, an agent can be endowed with advanced representational structures that are typically not part of the belief base (see, e.g., [23, 29, 17, 24]). Reasoning then involves these structures and existing active goals, i.e., modeling it as a simple check on the belief base does not faithfully capture these kinds of mechanisms. However, they could be introduced in our framework by adding a requirement for compatibility to the premise of the transition rules that govern activation.

A kind of reasoning that may be used to govern plan execution, is reasoning about maintenance goals. In order to prevent a maintenance goal from being violated, an agent sometimes has to refrain from doing a particular action (see [7, 9]). Moreover, an agent could reason about priorities among goals for determining whether a goal needs to be suspended. If a goal is adopted that has a higher priority than some active goal and is incompatible with this goal, this can be reason to force the lower priority goal to be suspended in favor of the higher priority one (see [17]). Capturing this in our abstract goal architecture would require the addition of a transition rule for goal suspension.

Besides the incorporation of advanced reasoning techniques in our formal framework, we see several other opportunities for future research. In particular, the analysis in Section 2 has shown that it can be interesting to consider other (variants of) goal types than have been considered in the literature so far. Extending agent-oriented programming languages with richer forms of goals will thus be an important issue for future research. Moreover, as pointed out in Section 2.1, we have in this paper only considered individual goals. In future work, we aim to explore the relationship between individual, group, and system/organization goals. We envisage that these are potentially useful as "stepping stones" between goal-oriented requirements engineering and the design of individual agents. Another important area that we did not cover in this paper, is the investigation of the use of goals in agent development. Although some aspects (such as goal-oriented requirements engineering) have been explored, other aspects remain to be explored in more detail, such as goal-oriented design [13], and the use of goals as a basis for modularization in agent programming [28].

Acknowledgements

We would like to thank the RMIT agent group (in particular Lin Padgham, Jennifer Sandercock, Sebastian Sardina, and John Thangarajah) for their useful comments. M. Birna van Riemsdijk has been sponsored by the project SENSORIA, IST-2005-016004.

5. REFERENCES

[1] M. E. Bratman. *Intention, plans, and practical reason*. Harvard University Press, Massachusetts, 1987.

[2] L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal representation for BDI agent systems. In *ProMAS'04*, volume 3346 of *LNAI*, pages 44–65. Springer, Berlin, 2005.

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

[4] M. Dastani and J.-J. Meyer. A practical agent programming language. In *Proc. of ProMAS '07*, volume 4908 of *LNAI*. Springer, 2008.

[5] M. Dastani, M. B. van Riemsdijk, F. Dignum, and J.-J. Ch. Meyer. A programming language for cognitive agents: goal directed 3APL. In *ProMAS'03*, volume 3067 of *LNAI*, pages 111–130. Springer, Berlin, 2004.

[6] M. Dastani, M. B. van Riemsdijk, and J.-J. Ch. Meyer. Goal types in agent programming. In *Proc. of ECAI'06*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 220–224. IOS Press, 2006.

[7] S. Duff, J. Harland, and J. Thangarajah. On proactivity and maintenance goals. In *Proc. of AAMAS'06*, pages 1033–1040, Hakodate, 2006.

[8] E. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 996–1072. Elsevier, Amsterdam, 1990.

[9] K. Hindriks and M. B. van Riemsdijk. Satisfying maintenance goals. In *Proc. of DALT'07*, 2007.

[10] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming in 3APL. *Int. J. of Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[11] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming with declarative goals. In *Proc. of ATAL'2000)*, Lecture Notes in AI. Springer, Berlin, 2001.

[12] J. F. Hübner, R. H. Bordini, and M. Wooldridge. Declarative goal patterns for AgentSpeak. In *Proc. of DALT'06*, 2006.

[13] J. Khallouf and M. Winikoff. Towards goal-oriented design of agent systems. In *Proc. of ISEAT'05*, 2005.

[14] A. F. Moreira, R. Vieira, and R. H. Bordini. Extending the operational semantics of a BDI agent-oriented programming language for introducing speech-act based communication. In *Proc. of DALT'03*, volume 2990 of *LNAI*, pages 135–154, London, UK, 2004. Springer-Verlag.

[15] V. Nigam and J. Leite. A dynamic logic programming based system for agents with declarative goals. In *Proc. of DALT'06*, volume 4327 of *LNAI*, pages 174–190, 2006.

[16] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: a BDI reasoning engine. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, Berlin, 2005.

[17] A. Pokahr, L. Braubach, and W. Lamersdorf. A goal deliberation strategy for BDI agent systems. In *MATES'05*, volume 3550 of *LNAI*, pages 82–93. Springer-Verlag, 2005.

[18] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. van der Velde and J. Perram, editors, *Agents Breaking Away (LNAI 1038)*, pages 42–55. Springer-Verlag, 1996.

[19] S. Sardina, L. P. de Silva, and L. Padgham. Hierarchical planning in BDI agent programming languages: A formal approach. In *Proc. of AAMAS'06*, pages 1001–1008, Hakodate, Japan, 2006. ACM Press.

[20] S. Sardina and S. Shapiro. Rational action in agent programs with prioritized goals. In *Proc. of AAMAS'03*, pages 417–424, 2003.

[21] J. Thangarajah, J. Harland, D. Morley, and N. Yorke-Smith. Suspending and resuming tasks in intelligent agents. In *Proc. of AAMAS'08*, 2008.

[22] J. Thangarajah, M. Winikoff, L. Padgham, and K. Fischer. Avoiding resource conflicts in intelligent agents. In *Proc. of ECAI'02*, Lyon, France, 2002.

[23] J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and avoiding interference between goals in intelligent agents. In *Proc. of IJCAI'03*, 2003.

[24] N. Tinnemeier, M. Dastani, and J. Meyer. Goal selection strategies for rational agents. In *Proc. of LADS'07*, 2007.

[25] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proc. of RE'01*, pages 249–263. IEEE, 2001.

[26] M. B. van Riemsdijk. *Cognitive Agent Programming: A Semantic Approach*. PhD thesis, 2006.

[27] M. B. van Riemsdijk, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming in Dribble: from beliefs to goals using plans. In *Proc. of AAMAS'03*, pages 393–400, 2003.

[28] M. B. van Riemsdijk, M. Dastani, J.-J. Ch. Meyer, and F. S. de Boer. Goal-oriented modularity in agent programming. In *Proc. of AAMAS'06*, pages 1271–1278, Hakodate, 2006.

[29] M. B. van Riemsdijk, M. Dastani, and J.-J. Ch. Meyer. Semantics of declarative goals in agent programming. In *Proc. of AAMAS'05*, pages 133–140, Utrecht, 2005.

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