A Dynamic Logic Programming Based System for Agents with Declarative Goals

Vivek Nigam* and João Leite

CENTRIA, New University of Lisbon, Portugal
vivek.nigam@gmail.com,
jleite@di.fct.unl.pt

Abstract. Goals are used to define the behavior of (pro-active) agents. It is our view that the goals of an agent can be seen as a knowledge base of the situations that it wants to achieve. It is therefore in a natural way that we use Dynamic Logic Programming (DLP), an extension of Answer-Set Programming that allows for the representation of knowledge that changes with time, to represent the goals of the agent and their evolution, in a simple, declarative, fashion. In this paper, we represent agent's goals as a DLP, discuss and show how to represent some situations where the agent should adopt or drop goals, and investigate some properties that emerge from using such representation.

1 Introduction

It is widely accepted that *intelligent agents* must have some form of *pro-active* behavior [20]. This means that an intelligent agent will try to pursue some set of states, represented by its *goals*. Generally, to determine these states, agents must reason for example, with their beliefs, capabilities or with other goals. It is therefore our perspective that the goals of an agent can be seen as a *knowledge base* encoding the situations it wants to achieve. Consider the following program, containing one rule, as an example of an agent's *goal base*:

$$goal(write\_paper) \leftarrow not\ deadline\_over$$

the agent will consider to write a paper ($goal(write\_paper)$), if the deadline is believed not to be past ($not\ deadline\ \_over$).

Programming with a declarative knowledge representation has demonstrated several advantages over *classical programming*. For instance, explicitly encoded knowledge can easily be *revised* and *updated*. Recently, an increasing amount of research [19,7,14,17,18,16] has been devoted to the issue of programming agents with a declarative representation of goals. The declarative side of goals intimately related to the need to check if a goal has been achieved, if a goal is impossible, if a goal should be dropped, i.e., if the agent should stop pursuing a goal, if there is

* Supported by the Alβan Program, the European Union Programme of High Level Scholarships for Latin America, no. E04M040321BR.

M. Baldoni and U. Endriss (Eds.): DALT 2006, LNAI 4327, pp. 174–190, 2006.

interference between goals [19,16]; and also to the need to construct agents that are able to *communicate* goals with other agents [14]. In [19,16,14] the reader can find examples illustrating the need for a declarative aspect to goals.

Furthermore, agents, due to changes in the environment, have the need to *drop* goals (maybe because the goal has been achieved, or a failure condition is activated [19]), *adopt* new goals [6,17,16,19], or even change the way they reason to determine their goals. Consider, in the previous example, that the deadline to submit the paper has been postponed. Clearly, the previous rule is not valid, since the previous deadline in no longer a condition to drop the goal of writing the paper, hence the rule should be updated. This means that the goals of an agents are *dynamic knowledge bases*, where not only the *extensional part* (i.e., the set of facts) change, but also their *intentional part* (i.e., the set of rules).

In this paper, we will address the problem of representing and reasoning about dynamic declarative goals using *Dynamic Logic Programming (DLP)*.

In [13,9], the paradigm of *DLP* was introduced. According to *DLP*, knowledge is given by a series of theories, encoded as generalized logic programs[1], each representing distinct states of the world. Different states, sequentially ordered, can represent different time periods, thus allowing DLP to represent knowledge that undergoes successive updates. Since individual theories may comprise mutually contradictory as well as overlapping information, the role of *DLP* is to employ the mutual relationships among different states to determine the declarative semantics for the combined theory comprised of all individual theories at each state. Intuitively, one can add, at the end of the sequence, newer rules (arising from new or reacquired knowledge) leaving to *DLP* the task of ensuring that these rules are in force, and that previous ones are valid (by inertia) only so far as possible, i.e. that they are kept for as long as they are not in conflict with newly added ones, these always prevailing.

There has been, in the past years, an intense study of the properties of DLP to represent knowledge bases that evolve with time [2,9,12]. However, up to now, there hasn't been much investigation of how DLP could be used to represent, in a declarative manner, the goals of an agent. Since DLP allows for the specification of knowledge bases that undergo change, and enjoys the expressiveness provided by both strong and default negations, by dint of its foundation in answer-set programming, it seems a natural candidate to be used to represent and to reason about the declarative goals of an agent, and the way they change with time.

For our purpose, we will use a simple agent framework to be able to clearly demonstrate the properties obtained by using DLP. The agents in this framework are composed of data structures representing their beliefs (*definite logic program*), goals (*DLP*), and committed goals (*intentions*). The semantics of these agents are defined by a *transition system* composed of *reasoning rules*. We propose three types of reasoning rules: **1)** Intention Adoption Rules: used to *commit* to a goal by adopting plans to achieve it; **2)** Goal Update Rules: used to *update* an agent's goals using the DLP semantics; **3)** Intention Dropping Rules: used to *drop* previously committed goals. We show that agents in this frame-

[1] Logic programs with default and strong negation both in the body and head of rules.

work are able to express achievement and maintenance goals, represent failure conditions for goals, and are able to adopt, drop or change their goals.

The remainder of the paper is structured as follows: in the next Section we are going to present some preliminaries, introducing Dynamic Logic Programming. In Section 3, we introduce the agent framework we are going to use. Later in Section 4, we discuss some situations related to when to drop and adopt new goals, and how to use the DLP semantics to represent these situations. In Section 5, we give a *simple* example illustrating how DLP could be used to represent goals, to finally draw some conclusions and propose some further research topics in Section 6.

2 Preliminaries

In this section, we are going to give some preliminary definitions that will be used throughout the paper. We start by introducing the syntax and semantics of goal programs. Afterwards, we introduce the semantics of *Dynamic Logic Programming*.

2.1 Languages and Logic Programming

Let $\mathcal{K}$ be a set of propositional atoms. An *objective knowledge literal* is either an atom A or a strongly negated atom $\neg A$. The set of objective knowledge literals is denoted by $\mathcal{L}_{\mathcal{K}}^{\neg}$. If $\{L_1, \ldots, L_n\} \subseteq \mathcal{L}_{\mathcal{K}}^{\neg}$ then $goal\,(L_1, \ldots, L_n)$, $def\,(L_1, \ldots, L_n)$, $maintenance\,(L_1, \ldots, L_n) \in \mathcal{L}_{\mathcal{G}}$[2]. We dub the element of $\mathcal{L}_{\mathcal{G}}$ *objective goal literals*. An *objective literal* is either an objective knowledge literal or an objective goal literal. A *default knowledge (resp. goal) literal* is an objective knowledge (resp. goal) literal preceded by *not*. A *default literal* is either a *default knowledge literal* or a *default goal literal*. A *goal literal* is either an objective goal literal or a default goal literal. A *knowledge literal* is either an objective knowledge literal or a default knowledge literal. A *literal* is either an objective literal or a default literal. We use $\mathcal{L}_{\mathcal{K}}^{\neg,not}$ to denote the set of knowledge literals and $\mathcal{L}_{\mathcal{G}}^{not}$ to denote the set of goal literals.

The set, $\mathcal{L}_{\mathcal{G}}$ also known as the goal language, uses a special symbol, *goal(.)* to represent the conjunction of achievement goals; the special symbol, *maintenance(.)*, to represent maintenance goals; the special symbol, *def(.)*, to represent defeasible goals.

A *goal rule* r (or simply a rule) is an ordered pair $Head\,(r) \leftarrow Body\,(r)$ where $Head\,(r)$ (dubbed the head of the rule) is a goal literal and $Body\,(r)$ (dubbed the body of the rule) is a set of literals. A rule with $Head\,(r) = L_0$ and $Body\,(r) = \{L_1, \ldots, L_n\}$ will simply be written as $L_0 \leftarrow L_1, \ldots, L_n$. A *goal program* (GP) P, is a finite or infinite set of rules. If $Head(r) = A$ (resp. $Head(r) = not\,A$) then $not\,Head(r) = not\,A$ (resp. $not\,Head(r) = A$). If $Head\,(r) = \neg A$ (resp.

[2] We will consider that there is a total order over the set of objective literals, $\mathcal{L}_{\mathcal{K}}^{\neg}$, and that the order in which the objective literals appear in the symbols of the goal language are based in this predefined ordering.

$Head(r) = A$), then $\neg Head\,(r) = A$ (resp. $\neg\, Head(r) = \neg A$). By the *expanded goal program* corresponding to the GP P, denoted by $\mathbf{P}$, we mean the GP obtained by augmenting P with a rule of the form $not\, \neg Head\,(r) \leftarrow Body\,(r)$ for every rule, in P, of the form $Head\,(r) \leftarrow Body\,(r)$, where $Head\,(r)$ is an objective goal literal[3]. Two rules r and r' are conflicting, denoted by $r \bowtie r'$, iff $Head(r) = not\, Head(r')$.

An *interpretation* M is a set of objective literals that is consistent i.e, M does not contain both:

- A and $\neg A$;
- $goal(L_1, \ldots, L, \ldots, L_n)$ and $goal(L_1, \ldots, \neg L, \ldots, L_n)$;
- $maintenance(L_1, \ldots, L, \ldots, L_n)$ and $maintenance(L_1, \ldots, \neg L, \ldots, L_n)$;
- $maintenance(L_1, \ldots, L, \ldots, L_n)$ and $goal(L_1, \ldots, \neg L, \ldots, L_n)$.

An objective literal L is true in M, denoted by $M \vDash L$, iff $L \in M$, and false otherwise. A default literal $not\, L$ is true in M, denoted by $M \vDash not\, L$, iff $L \notin M$, and false otherwise. A set of literals B is true in M, denoted by $M \vDash B$, iff each literal in B is true in M. Only inconsistent sets of objective literals (In), will entail the special symbol $\bot$ (denoted by $In \models \bot$). $\bot$ can be seen semantically equivalent to the formula $A \wedge \neg A$. An interpretation M is an *answer set* (or stable model) of a GP P iff $M' = least\,(\mathbf{P} \cup \{not\, A \mid A \notin M\})$, where $M' = M \cup \{not\_A \mid A \notin M\}$, A is an objective literal, and $least(.)$ denotes the least model of the definite program obtained from the argument program by replacing every default literal $not\, A$ by a new atom not_A.

For notational convenience, we will no longer explicitly state the alphabet $\mathcal{K}$. And as usual, we will consider all the variables appearing in the programs as a shorthand for the set of all their possible ground instantiations.

2.2 Dynamic Logic Programming

A *dynamic logic (goal) program* (DLP) is a sequence of goal programs. Let $\mathcal{P} = (P_1, ..., P_s)$ be a DLP and P' a GP. We use $\rho\,(\mathcal{P})$ to denote the multiset of all rules appearing in the programs $\mathbf{P}_1, ..., \mathbf{P}_s$, and $(\mathcal{P}, P')$ to denote $(P_1, ..., P_s, P')$.

The semantics of a DLP is specified as follows:

Definition 1 (Semantics of DLP). *[9,1] Let $\mathcal{P} = (P_1, \ldots, P_s)$ be a dynamic logic program A be an objective literal, $\rho\,(\mathcal{P})$, M' and $least(.)$ be as before. An interpretation M is a (goal dynamic) stable model of $\mathcal{P}$ iff*

$$M' = least\,([\rho\,(\mathcal{P}) - Rej(M, \mathcal{P})] \cup Def(M, \mathcal{P}))$$

Where:

$$Def(M, \mathcal{P}) = \{not\, A \mid \nexists r \in \rho(\mathcal{P}), Head(r) = A, M \vDash Body(r)\}$$
$$Rej(M, \mathcal{P}) = \{r \mid r \in \mathbf{P}_i, \exists r' \in \mathbf{P}_j, i \leq j \leq s, r \bowtie r', M \vDash Body(r')\}$$

[3] Expanded programs are defined to appropriately deal with strong negation in updates. For more on this issue, the reader is invited to read [10,9]. From now on, and unless otherwise stated, we will always consider generalized logic programs to be in their expanded versions.

We will denote by $SM(\mathcal{P})$ the set of stable models of the DLP $\mathcal{P}$. Further details and motivations concerning DLPs and its semantics can be found in [9].

The next example illustrates how a DLP could be used to represent the goals of an agent.

Example 1. Consider the goals of a young agent that reached a point in her life that she is interested in having a boyfriend. However, to find a boyfriend (as many know) may not be the easiest task, but she knows that being pretty helps to achieve it. We can represent this situation by the following program:

$$\begin{aligned} P_1 : \mathit{goal}(\mathit{boyfriend}) &\leftarrow \mathit{not}\, \mathit{boyfriend} \\ \mathit{goal}(\mathit{pretty}) &\leftarrow \mathit{not}\, \mathit{pretty}, \mathit{goal}(\mathit{boyfriend}) \end{aligned}$$

As she is not pretty and doesn't have a boyfriend, her initial goals would be to have a boyfriend and to be pretty, represented by the unique stable model of P_1, $\{\mathit{goal}(\mathit{boyfriend}), \mathit{goal}(\mathit{pretty})\}$. Her mother, noticing the desires of her daughter and as usual looking for the best for her child, immediately tells her that she should study. As the agent respects her mother, she updates her goals with the program P_2, stating the incompatibility between studying and having a boyfriend:

$$\begin{aligned} P_2 : \mathit{not}\, \mathit{goal}(\mathit{boyfriend}) &\leftarrow \mathit{goal}(\mathit{study}) \\ \mathit{goal}(\mathit{study}) &\leftarrow \end{aligned}$$

Since, P_2 is a newer program than P_1, the rule $\mathit{goal}(\mathit{boyfriend}) \leftarrow \mathit{not}\ \mathit{boyfriend}$ will be rejected, according to the semantics of DLP, by the rule $\mathit{not}\, \mathit{goal}(\mathit{boyfriend}) \leftarrow \mathit{goal}(\mathit{study})$. Furthermore, the goal of being pretty will no longer be supported. Hence, the DLP (P_1, P_2) has a unique stable model, $\{\mathit{goal}(\mathit{study})\}$. After sometime, the agent grows and becomes more confident (to a point that she can question her mother). As she is tired of studying and attending the boring math classes, she decides that studying is no longer her objective. She then, updates her goals with the program P_3:

$$P_3 : \mathit{not}\, \mathit{goal}(\mathit{study}) \leftarrow$$

As a result, the rule $\mathit{goal}(\mathit{study}) \leftarrow$ will be rejected and she will once more have as a goal to find a boyfriend. The DLP (P_1, P_2, P_3) has the unique stable model $\{\mathit{goal}(\mathit{boyfriend}), \mathit{goal}(\mathit{pretty})\}$. However, discussing with some of her friends (or maybe reading some women magazine), she discovers that to be pretty either she has to wear nice clothes (go to the shopping) or have a nice body (fitness), therefore she updates her goal with the program, P_4:

$$\begin{aligned} P_4 : \mathit{goal}(\mathit{shopping}) &\leftarrow \mathit{not}\, \mathit{shopping}, \mathit{goal}(\mathit{pretty}), \mathit{not}\, \mathit{goal}(\mathit{fitness}) \\ \mathit{goal}(\mathit{fitness}) &\leftarrow \mathit{not}\, \mathit{fitness}, \mathit{goal}(\mathit{pretty}), \mathit{not}\, \mathit{goal}(\mathit{shopping}) \end{aligned}$$

The DLP (P_1, P_2, P_3, P_4) has two stable models, one representing that she has the goal of shopping $\{\mathit{goal}(\mathit{boyfriend}), \mathit{goal}(\mathit{pretty}), \mathit{goal}(\mathit{shopping})\}$, and another of getting fit $\{\mathit{goal}(\mathit{boyfriend}), \mathit{goal}(\mathit{pretty}), \mathit{goal}(\mathit{fitness})\}$.

As illustrated in the example above, a DLP can have more than one stable model. But then, how to deal with these stable models and how to represent the semantics of a DLP? This issue has been extensively discussed and three main approached are currently being considered [9]:

Skeptical - $\models_\cap$ According to this approach, the *intersection* of all stable models is used to determine the semantics of a DLP. Therefore, a formula φ is entailed by the DLP $\mathcal{P}$, denoted by $\mathcal{P} \models_\cap \varphi$, iff it is entailed by all the program's stable models;

Credulous - $\models_\cup$ According to this approach, the *union* of all stable models is used to determine the semantics of a DLP. Therefore, a formula φ is entailed by the DLP $\mathcal{P}$, denoted by $\mathcal{P} \models_\cup L$, iff it is entailed by one of the program's stable models;

Casuistic - $\models_\Omega$ According to this approach, one of the stable models is selected by a selection function Ω, to represent the semantics of the program. Since, the stable models can be seen as different consistent options, or possible worlds, by adopting this approach the agent would commit to one of these options. We use $\mathcal{P} \models_\Omega L$, to denote that formula φ is entailed by the stable model of the DLP $\mathcal{P}$, selected by Ω;

3 Agent Framework

In this Section, we define the agent framework[4] that we will use to demonstrate the properties obtained by using Dynamic Logic Programming to represent the goals of an agent. An agent in this framework is composed by a *belief base* representing what the agent believes the world is; a *goal base* representing the states the agent wants to achieve; a set of *reasoning rules*; and a set of *intentions* with two associated *plans* representing the goals that the agent is currently *committed* to achieve. We are considering that the agent has, at its disposal, a *plan library* represented by the set of plans $Plan$. A plan can be viewed as a *sequence of actions* that can modify the agent's beliefs or/and the environment surrounding it, and is used by the agent to *try* to achieve a committed goal, as well as to do the cleaning up actions.

The idea behind associating two plans to an agent's intention, is that one of the plans, $\pi_{achieve}$, will be used to try to achieve the intention, and the second plan, π_{clean}, is used to do all the *cleaning up* actions after the goal is dropped, or when there are no more actions to be performed in $\pi_{achieve}$. For example, if an agent's intention is to *bake a cake*, it would execute an appropriate plan to achieve its goal ($\pi_{achieve}$), gathering the ingredients, the utensils, and setting up the oven. After the cake is baked, the agent would still have to wash the utensils and throw the garbage away, these actions could be seen as clean up actions (π_{clean}).

[4] The agent framework defined in this section could be seen as a modified (simplified) version of the agent framework used in the 3APL multi-agent system [5].

Our main focus in this paper is to investigate the properties of representing the goal base as a Dynamic Logic Program. We are not going to give the *deserved* attention to the belief base. We consider the belief base as a simple definite logic program. However, a more complex belief base could be used. For example, we could represent the belief base also as a Dynamic Logic Program and have some mechanism such that the agent has an unique model for its beliefs[5]. Elsewhere, in [15], we explore the representation of 3APL agent's belief base as a DLP.

Since, the reasoning rules of an agent don't change, it is useful to define the concept of *agent configuration* to represent the variable state of an agent.

Definition 2 (Agent Configuration). *An agent configuration is a tuple* $\langle \sigma, \gamma, \Pi \rangle$*, where* σ *is a definite logic program over* $\mathcal{K}$*, representing the agent's belief base,* $\gamma = (P_1, \ldots, P_n)$ *is a DLP representing it's goal base, such that every* P_i *is a goal program and that the DLP* (γ, σ) *has at least one stable model, and* $\Pi \subseteq Plan \times Plan \times \mathcal{L}_{\mathcal{G}}$ *the intention base of the agent.*

As the goals of an agent might be dependent on its beliefs, to determine its goals it will be necessary to integrate the agent's belief base (σ) and its goal base (γ). We straightforwardly use the DLP semantics to do this integration by considering the DLP (γ, σ) to determine the agent's goals. Consider the following illustrative example:

Example 2. Consider an agent with a goal base, $\gamma = (P)$, consisting of the following program P, stating that the agent will try to have a girlfriend if it has money, and if it doesn't have the goal of saving money:

$$P : goal(girlfriend) \leftarrow have\_money, not\, goal(save\_money)$$

And with its belief base, σ, stating that the agent will have money if it has low expenses and a high income, and that currently this is the case:

$$\begin{aligned} \sigma : \; & have\_money \leftarrow low\_expenses, high\_income \\ & high\_income \leftarrow \\ & low\_expenses \leftarrow \end{aligned}$$

To determine if $goal(girlfriend)$ will be a goal of the agent we update γ with σ, and clearly having a girlfriend will be a goal of the agent, since $goal(girlfriend)$ will be entailed by the DLP (γ, σ). Note that since σ is a definite logic program, and goal rules in γ do not have knowledge literals in their heads, the update of γ with σ amounts to determine the unique model of σ and use it to perform a partial evaluation of γ.

Moreover, we only consider the agent configurations, $\langle \sigma, \gamma, \Pi \rangle$, where the DLP (γ, σ) has at least one model, since an agent without semantics for its goal base wouldn't be of much interest in this work.

[5] For example, a *belief model selector* that would select one of the stable models of the belief base to represent the agent's beliefs.

We assume that the semantics of the agents is defined by a *transition system*. A transition system is composed of a set of *transition rules* that transforms one agent configuration into another agent configuration, in one computation step. It may be possible that one or more transition rules are applicable in a certain agent configuration. In this case, the agent must decide which one to apply. This decision can be made through a *deliberation cycle*, for example, through a priority among the rules. In this paper, we won't specify a deliberation cycle. An unsatisfied reader can consider a *non-deterministic* selection of the rules.

We now introduce the *intention adoption rules*. These rules are used to adopt plans to try to achieve goals of the agent. Informally, if the agent has a goal, $goal(L_1, \ldots, L_n)$, that currently doesn't have plans in the agent's intention base (Π), the rule will adopt a couple of plans, $\pi_{achieve}, \pi_{clean}$, by adding the tuple $(\pi_{achieve}, \pi_{clean}, goal(L_1, \ldots, L_n))$ to the agent's intention base. However, as argued by Bratman in [4], agent's shouldn't pursue at the same time contradictory goals. Therefore, similarly as done in [18], we check if by adopting a new goal, the intentions of the agent are consistent.

Definition 3 (Intention Adoption Rules). *Let $\langle \sigma, \gamma, \Pi \rangle$ be an agent configuration and $goal(L_1, \ldots, L_n) \in \mathcal{L}_{\mathcal{G}}$, where*

$$\Pi = \{(\pi^1_{achieve}, \pi^1_{clean}, goal(L^1_1, ..., L^1_i)), ..., (\pi^m_{achieve}, \pi^m_{clean}, goal(L^m_1, ..., L^m_j))\}$$

such that $\{(\pi_{achieve}, \pi_{clean}, goal\,(L_1, \ldots, L_n))\} \nsubseteq \Pi$, and $x \in \{\cap, \cup, \Omega\}$.

$$\frac{\begin{array}{c}[(\gamma, \sigma) \models_x goal\,(L_1, \ldots, L_n) \vee (\gamma, \sigma) \models_x maintenance\,(L_1, \ldots, L_n)] \wedge \\ \{L^1_1, \ldots, L^1_i, \ldots, L^m_1, \ldots, L^m_j, L_1, \ldots, L_n\} \nvdash \bot\end{array}}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi \cup \{(\pi_{achieve}, \pi_{clean}, goal\,(L_1, \ldots, L_n))\}\rangle}$$

Notice that the condition of consistency of the agent's intentions may not yet be the best option to avoid irrational actions. Winikoff et al. suggest, in [19], that it is necessary also to analyze the *plans* of the agent, as well as the *resources* available to achieve the intentions. However, this is out of the scope of this paper.

We have just introduced a rule to adopt new intentions. Considering that intentions are committed goals, if the goal that the intention represents is no longer pursued by the agent, it would make sense to drop it. Therefore, we introduce into our agent framework the *Intention Dropping Rule*. Informally, the semantics of this rule is to stop the execution of the plan used to achieve the goal ($\pi_{achieve}$), if the goal is no longer supported by the goal base of an agent, and start to execute the plan used to perform the cleaning up actions (π_{clean}). The next definition formalizes this idea.

Definition 4 (Intention Dropping Rule). *Let $\langle \sigma, \gamma, \Pi \rangle$ be an agent configuration, $x \in \{\cap, \cup, \Omega\}$, where $\{(\pi_{achieve}, \pi_{clean}, \psi)\} \subseteq \Pi$, $\psi = goal\,(L_1, \ldots, L_n)$, and $\pi_{achieve} \neq \emptyset$. Then:*

$$\frac{(\gamma, \sigma) \nvDash_x goal\,(L_1, \ldots, L_n) \wedge (\gamma, \sigma) \nvDash_x maintenance\,(L_1, \ldots, L_n)}{\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma, \Pi \setminus \{(\pi_{achieve}, \pi_{clean}, \psi)\} \cup \{(\emptyset, \pi_{clean}, \psi)\}\rangle}$$

Since an agent's goal base is represented by a Dynamic Logic Program, an agent can easily update its goal base with a GP using the DLP semantics. As we will investigate in the next Section, updating a goal base with a GP will enable the agent to have dynamic goals, e.g. by goal adoption or goal dropping. For this purpose, we introduce a new type of reasoning rule to the system, namely the *Goal Update Rules*.

Definition 5 (Goal Update Rule). *The Goal Update Rule is a tuple* $\langle \Sigma_B, \Sigma_G, P\rangle$ *where* P *is a Goal Program, and* $\Sigma_B \subseteq \mathcal{L}^{\neg,not}$, *and* $\Sigma_G \subseteq \mathcal{L}_{\mathcal{G}}^{not}$. *We will call* Σ_B *and* Σ_G *the precondition of the goal update rule.*

Informally, the *semantics* of the goal update rule $\langle \Sigma_B, \Sigma_G, P\rangle$, is that when the precondition, Σ_B, Σ_G, is satisfied, respectively, by the agent's belief base and by its goal base, the goal base of an agent is updated by the goal program P. For example, consider the rule:

$$\langle \{tough\_competition\}, \{goal(go\_to\_school)\}, \{goal(good\_in\_math) \leftarrow\}\rangle$$

according to which the agent will only update its goal base with the goal of being good in math, if the agent believes that the competition will be tough ($tough\_competition$), and if it has the goal of going to school ($goal(go\_to\_school)$).

Definition 6 (Semantics of Goal Update Rules). *Let* $\langle \sigma, \gamma, \Pi\rangle$ *be an agent configuration, and* $x \in \{\cap, \cup, \Omega\}$. *The semantics of a Goal Update Rule,* $\langle \Sigma_B, \Sigma_G, P\rangle$ *is given by the transition rule:*

$$\frac{\sigma \models \Sigma_B \wedge (\gamma, \sigma) \models_x \Sigma_G}{\langle \sigma, \gamma, \Pi\rangle \longrightarrow \langle \sigma, (\gamma, P), \Pi\rangle}$$

In this framework, we will use the special symbols, $goal()$ and $maintenance()$ to be able to differentiate between *maintenance* and *achievement goals*. A maintenance goal represents a state of affairs that the agent wants to hold in all states. For example, a person doesn't want to get hurt. An achievement goal represents a state of affairs that, once *achieved*, is no longer pursued. For example, an agent that has as goal to write a paper for a congress, after it believes it has written the paper, it should no longer consider this as a goal.

We are going to use a *goal update operator* to drop the achievement goals that have been achieved. The idea is to apply the goal update operator whenever the belief base of the agent is changed (this could be done by a deliberation cycle).

Definition 7 (Goal Update Operator - Γ). *Let* $\langle \sigma, \gamma, \Pi\rangle \longrightarrow \langle \sigma', \gamma', \Pi'\rangle$ *be a transition in the transition system,* $x \in \{\cap, \cup, \Omega\}$, *where* $\langle \sigma, \gamma, \Pi\rangle$ *and* $\langle \sigma', \gamma', \Pi'\rangle$ *are agent configurations. We define the goal update operator,* Γ, *as follows:*

$$\Gamma(\gamma, \sigma') = \gamma' = (\gamma, \mu(\sigma', \gamma))$$

where:

$$\mu(\sigma', \gamma) = \{not\, goal(L_1, \ldots, L_m) \leftarrow \mid (\gamma, \sigma') \models_x goal(L_1, \ldots, L_m), \sigma' \models \{L_1, \ldots, L_m\}\}$$

Notice that the agent will still consider maintenance goals as goals even if the goal is currently achieved.

As previously mentioned, the semantics of an individual agent is defined by the reasoning rules we just introduced. More specifically the meaning of individual agents consist of a set of so called computation runs.

Definition 8 (Computation Runs). *Given a transition system, a computation run, $CR(s_0)$, is a finite or infinite sequence, $s_0, \ldots, s_n, \ldots$, such that for all $i \geq 0$, s_i is an agent configuration, and $s_i \rightarrow s_{i+1}$ is a transition in the transition system.*

From the above definitions it is easy to see that a maintenance goal will remain entailed by the agent unless dropped by means of a goal update rule, or no longer supported due to some change in the agent's beliefs. Similar reasoning is applicable to achievement goals: if an achievement goal is not dropped using a goal update rule, and the beliefs of the agent do not change so as to no longer support it, it will remain entailed by an agent until it believes that the goal is achieved. We will investigate more about goal update rules in the next Section, when we discuss goal adoption and goal dropping.

4 Adopting and Dropping Goals

In this section we are going to investigate how to represent, in our system, situations where an agent has to adopt or drop goals. We begin, in Subsection 4.1, by discussing some possible motivations of why an agent should adopt a goal and also investigate how to represent these motivations in our agent framework. Later, in Subsection 4.2, we investigate how to represent failure conditions for goals and discuss some other situations to drop a goal. Finally in Subsection 4.3, we identify some further properties of our framework.

4.1 Goal Adoption

Agents often have to adopt new goals. The reasons for adopting new goals can be varied, the simplest one, when dealing with pro-active agents, could be because the agent doesn't have any goals and it is in an *idle* state.

We follow [17], and distinguish two motivations behind the adoption of a goal: *internal* and *external*. Goals that derive from the desires of an agent, represented by *abstract goals*, have an internal motivation to be adopted. External motivations, such as *norms*, *obligations*, and *impositions* from other agents, can also be a reason for the agent to adopt new goals. An example of a norm, in the daily life, is that a person should obey the law. Obligations could derive from a *negotiation* where an agent commits to give a service to another agent e.g. your internet provider should (is obliged to) provide the internet connection at your home. Agents usually have a *social point of view* e.g. a son usually respects his father more than a stranger, and it may be the case that an agent imposes another agent some specific goals e.g. a father telling the son to study.

To be able to commit to obligations, changes in norms, or changes in desires, an agent needs to be able to update its goal base during execution. For example, if a new deal is agreed to provide a service to another agent, the agent must entail this new obligation. By using the Goal Update Rule, an agent will be able to update its goal base and adopt new goals, as states the following proposition.

Proposition 1 (Goal Adoption Property). *Let* $Goal \in \mathcal{L}_{\mathcal{G}}$, $\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma', \Pi \rangle$ *be the transition rule of the goal update rule* $\langle \Sigma_B, \Sigma_G, P \rangle$, *where* $r : Goal \leftarrow \in P$ *is not conflicting in* P, *and* $x \in \{\cap, \cup, \Omega\}$. *Then:*

$$(\gamma', \sigma) \models_x Goal$$

Proof: Since $Goal \leftarrow \in P$ *is not conflicting in* P. *For all interpretations,* r *will not be rejected by any other rule in the goal base. Therefore, we have that* $(\gamma', \sigma) \models_x Goal$.

Now, we discuss some situations where an agent has to adopt new goals.

Adopt New Concrete Goals - Dignum and Conte discuss in [6], that an agent may have some desires that can be represented by abstract goal κ that is usually not really achievable, but the agent believes that it can be approximated by some concrete goals $(\kappa_1, \ldots, \kappa_n)$. Consider that the agent learns that there is another concrete goal κ_l that, if achieved, can better approximate the abstract goal, κ. The agent can update its goal base using the following Goal Update Rule, $\langle \{concrete\_goal(\kappa_l, \kappa)\}, \{\}, \{goal(\kappa_l) \leftarrow goal(\kappa)\} \rangle$, as κ is a goal of the agent, it will activate the new rule, hence the new concrete goal, κ_l, will also be a goal of the agent. In example 1, the girl agent considers initially that a more concrete goal to have a boyfriend is of being pretty;

Norm Changes - Consider that the agent belongs to a *society* with some norms that have to be obeyed $(norm_1, \ldots, norm_n)$ and furthermore that there is a change in the norms. Specifically, the $norm_i$ is changed to $norm'_i$, hence the agent's goal base must change. We do this change straightforwardly, using the goal update rule, $\langle \{change(norm_i, norm'_i)\}, \{\}, \{not\ goal(norm_i) \leftarrow ; goal(norm'_i) \leftarrow\} \rangle$. This update will force all the rules, r, with $Head(r) = goal(norm_i)$ to be rejected and $norm_i$ will no longer be a goal of the agent. Notice that there must be some coherence with the change in the norms. For example, the agent shouldn't believe that on $change(norm_i, norm_j)$ and at the same time on $change(norm_j, norm_i)$;

New Obligations - Agents are usually immersed with other agents in an environment and, to achieve certain goals, it might be necessary to negotiate with them. After a negotiation round, it is normal for agents to have an agreement that stipulates some conditions and obligations (e.g. in *Service Level Agreements* [8]). The agent can again easily use the goal update rules to incorporate new obligations, $\langle \{obligation(\phi)\}, \{\}, \{goal(\phi) \leftarrow\} \rangle$, as well as *dismiss* an obligation when an agreement is over, $\langle \{\neg obligation(\phi)\}, \{\}, \{not\ goal(\phi) \leftarrow\} \rangle$;

Impositions - Agents not only negotiate, but sometimes have to *cooperate* with or *obey* other superior agents. This sense of superiority is quite subjective and can be, for example, the obedience of an employee to his boss, or a provider towards his client. It will depend on the beliefs of the agent to decide if it should adopt a new goal or not, but this can be modeled using the goal update rule, $\langle\{received(achieve, \phi, agent_i), obey(agent_i)\}, \{\}, \{\ goal(\phi) \leftarrow \}\rangle$. Meaning that if it received a message from $agent_i$ to adopt a new goal ϕ, and the receiving agent believes it should obey $agent_i$, it will update its goal base. Notice that more complex hierarchy could be achieved by means of *preferences* between the agents. However, it would be necessary to elaborate a mechanism to solve possible *conflicts* (e.g by using Multi-Dimensional Dynamic Logic Programming [11]).

4.2 Goal Dropping

In this Subsection, we are going to investigate some situations where the agent must *drop a goal* and discuss how this could be done with our agent framework.

The next proposition, states that goal update rules can be used to drop achievement goals, as well as maintenance goals.

Proposition 2 (Goal Drop Property). *Let* $\langle\sigma, \gamma, \Pi\rangle \longrightarrow \langle\sigma, \gamma', \Pi\rangle$ *be the transition rule of the goal update rule* $\langle\Sigma_B, \Sigma_G, P\rangle$*, such that* $r : not\, Goal \leftarrow \in P$*, and* $x \in \{\cap, \cup, \Omega\}$*, where* $Goal \in \mathcal{L}_{\mathcal{G}}$*. Then:*

$$(\gamma', \sigma) \nvDash_x Goal$$

Proof: Since $r \in P$ *and that the goal update rule semantics adds the program* P *to the end of the goal base.* r *will reject all the rules,* r'*, in the goal base* γ*, with* $Head(r') = Goal$*. Therefore,* $(\gamma', \sigma) \nvDash_x Goal$*.*

We already have discussed in the previous Subsection, some situations where the agent must drop a goal, for instance, when obligations with other agents are ended, or when there is change in the norms that the agent should obey. Another situation that could force an agent to drop a goal, is suggested by Winikoff et al. in [19], by defining *failure conditions*. The idea is that when the failure condition is true the goal should be dropped. We can easily define failure conditions for goals using Goal Update Rules. Consider the following example:

Example 3. Consider an agent that has to write a paper until a deadline of a conference. We could represent this situation using the following Goal Update Rule, $\langle\{deadline\_over\}, \emptyset, \{not\, goal(write\_paper) \leftarrow\rangle$. The agent will drop the goal of writing a paper if the deadline is over.

Agents should also drop achievement goals, whenever this goal is achieved. The agent framework will perform this by using the goal update operator whenever there is a change in the agent's beliefs. As the following proposition shows, this operator updates the agent's goal base in such a way that the agent will no longer consider as goals previous achievement goals that have been achieved.

Proposition 3 (Goal Update Operator Property). *Let $\mathcal{A} = \langle \sigma, \gamma, \Pi \rangle$ be an agent configuration such that $\sigma \models \{L_1, \ldots, L_n\}$, and $(\gamma, \sigma) \models_x goal(L_1, \ldots, L_n)$, and let $\gamma' = \Gamma(\gamma, \sigma)$, and $x \in \{\cap, \cup, \Omega\}$. Then for any belief base σ_i:*

$$(\gamma', \sigma_i) \not\models_x goal(L_1, \ldots, L_n)$$

Proof: Since $\sigma \models \{L_1, \ldots, L_n\}$, and $(\gamma, \sigma) \models_x goal(L_1, \ldots, L_n)$, the goal update operator will update the goal base γ with a program P containing the rule not $goal(L_1, \ldots, L_n) \leftarrow$, that will reject all the rules in the goal base with head $goal(L_1, \ldots, L_n)$. Therefore, for any σ_i and $x \in \{\cup, \cap, \Omega\}$, we have that $(\gamma', \sigma_i) \not\models_x goal(L_1, \ldots, L_n)$.

4.3 Further Properties

We still can identify some more properties that could be elegantly achieved by using the goal update rule:

Defining Maintenance and Achievement Goals - We can define a goal as a maintenance goal if a certain condition is satisfied. For example, an initially single male agent finds the woman agent of its life and marries it. After this is achieved, it might like to be married with this agent until the end of its life. This can be represented by the goal update rule $\langle \{married(girl)\}$, $\{\}$, $\{$ $maintenance(married(girl)) \leftarrow\} \rangle$. The opposite can also be easily achieved, using the goal update rule. A goal that initially was a maintenance goal can be dropped or switched to an achievement goal. For example, consider that the previous agent had a fight with its agent wife and, after the divorce, it doesn't want to marry again. This can be represented by the goal update rule, $\langle \{divorce(girl)\}$, $\{\}$, $\{$ $not\ goal(married(girl)) \leftarrow$; $not\ maintenance(married(girl)) \leftarrow\} \rangle$. We define a new achievement or modify a maintenance goal to an achievement by using the following goal update rule $\langle \{achieve(L)\}$, $\{\}$, $\{$ $goal(L) \leftarrow$; $not\ maintenance(L) \leftarrow\} \rangle$;

The next corollary guarantees the effectiveness of the change of one achievement goal to a maintenance goal. A similar result could be used to change one maintenance goal to an achievement goal.

Corollary 1 (Achievement to Maintenance Goal). *Let $\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma', \Pi \rangle$ be the transition rule of the goal update rule $\langle \Sigma_B, \Sigma_G, P \rangle$, where $P = \{maintenance(L_1, \ldots, L_n) \leftarrow; not\ goal(L_1, \ldots, L_n) \leftarrow\}$, and $x \in \{\cap, \cup, \Omega\}$. Then:*

$$(\gamma', \sigma) \not\models_x goal(L_1, \ldots, L_n) \wedge (\gamma', \sigma) \models_x maintenance(L_1, \ldots, L_n)$$

Proof: Follows from propositions, 1 and 2.

Corollary 2 (Maintenance to Achievement Goal). *Let $\langle \sigma, \gamma, \Pi \rangle \longrightarrow \langle \sigma, \gamma', \Pi \rangle$ be the transition rule of the goal update rule $\langle \Sigma_B, \Sigma_G, P \rangle$, where*

$P = \{not\, maintenance(L_1, \ldots, L_n) \leftarrow; goal(L_1, \ldots, L_n) \leftarrow\}$, *and* $x \in \{\cap, \cup, \Omega\}$. *Then:*

$$(\gamma', \sigma) \models_x goal(L_1, \ldots, L_n) \wedge (\gamma', \sigma) \not\models_x maintenance(L_1, \ldots, L_n)$$

Proof: Follows from propositions, 1 and 2.

Representing Defeasible Goals - We can use the special symbol $def(.)$ to represent *Defeasible Goals*, i.e. goals that with the current knowledge are considered as goals (or not), but if new knowledge is acquired, the goals are dropped (or adopted). We take the surgery example from Bacchus and Grove [3]. A person may prefer not having surgery over having surgery, but this preference might be reversed in the circumstances where surgery improves one's long term health. We can defeasibly infer that the person prefers no surgery only as long as it is known that surgery improves his or her long term health. This could be modeled by the following program:

$$\begin{array}{l} maintenance(long\_life) \leftarrow \\ goal(\neg surgery) \leftarrow not\, goal(surgery), not\, def(\neg surgery) \\ goal(surgery) \leftarrow not\, goal(\neg surgery), not\, def(surgery) \\ def(\neg surgery) \leftarrow needs\_surgery, maintenance(long\_life) \\ def(surgery) \leftarrow not\, needs\_surgery \end{array}$$

the agent will only have surgery as a goal if it needs surgery ($needs\_surgery$) and has the goal of living long.

5 Example

Consider the following situation. The *wife* agent of a recently married couple, invites her *mother-in-law* for dinner at her house. Since, the couple has recently been married, the *wife* is still very concerned of her relations with her mother-in-law (mother-in-law are famous for not being very fond of daughter-in-law). And as the daughter-in-law loves her husband, she doesn't want any problems with his mother. We can represent its initial goal base as $\gamma = (P_1)$, where P_1 is as follows:

$$\begin{array}{l} P_1 : maintenance(husband's\_love) \leftarrow \\ \quad goal(please\_motherInLaw) \leftarrow maintenance(husband's\_love) \end{array}$$

P_1 states that she has as maintenance goal to have the love of her husband and hence, she has to please her mother-in-law, represented by its unique stable model, $\{maintenance(husband's\_love), goal(please\_motherInLaw)\}$. To please her mother-in-law is not a very easy task (probably, there is no plan to please a person, but there are plans to achieve more concrete goals). However, she knows that by making a good dinner, she will give her mother-in-law a very good impression. But not being a real master cook, the wife agent searches in

the internet how to make a good dinner, and discovers that she should use *white wine* if serving *fish*, and *red wine* if serving *lamb*. Promptly, she updates her goals using the following goal update rule:

$$\langle \{norm(lamb, red\_wine), norm(fish, white\_wine)\}, \{goal(please\_motherInLaw)\}, P_2 \rangle$$

where:

$$P_2 : goal(lamb, red\_wine) \leftarrow not\ goal(fish, white\_wine)$$
$$goal(fish, white\_wine) \leftarrow not\ goal(lamb, red\_wine)$$

The wife's goal base, (P_1, P_2) has two stable models, namely one where she has as goal to prepare fish with white wine ($\{maintenance(husband's\_love)$, $goal(please\_motherIn\ Law)$, $goal(fish, white\_wine)\}$) and another where she instead, would like to cook lamb with red wine ($\{maintenance(husband's\_love)$, $goal(please\_mother\ InLaw)$, $goal(lamb, red\_wine)\}$). She decides for some reason, that the lamb would be a better option. Notice that the agent in this example, is using the Casuistic approach to handle the multiple stable models (where the agent chooses one of the DLP's stable models to determine its semantics). However, she finds out that the red wine she reserved for a special occasion is mysteriously gone. Therefore, she cannot make lamb with red wine anymore (failure condition), updating its goal base with the following goal update rule, $\langle \{not\ red\_wine\}, \{\}, P_3 \rangle$, where:

$$P_3 : not\ goal(lamb, red\_wine) \leftarrow$$

After this update, the wife's goals will change, and she will have to prepare the fish with white wine. since the rule in P_3 will reject the rule with head $goal(lamb, red\_wine)$ in P_2. Hence, the DLP (P_1, P_2, P_3) will have one stable model, namely:

$$\{maintenance(husband's\_love), goal(please\_motherInLaw), goal(fish, white\_wine)\}.$$

After preparing the fish and collecting the white wine, the wife updates its goal base with the following program, P_4, obtained from the goal update operator:

$$P_4 : not\ goal(fish, white\_wine) \leftarrow$$

Since the rule $not\ goal(fish, white\_wine) \leftarrow$ in P_4 will reject the rule with head $goal(fish, white\_wine)$ in P_2, the goals of the agent will be again:

$$\{maintenance(husband's\_love), goal(please\_motherInLaw)\}$$

However, the wife agent still puzzled how the red wine mysteriously disappeared, tries to find it. Until a point that she looks inside the husband's closet, and finds a shirt stained with the wine and inside its pocket a paper with a love letter and

a telephone. Immediately, she considers that her husband is cheating her with another women and updates her goals with the following goal update rule:

$$\langle \{cheating\_husband\}, \{\}, \{not\, maintenance(husband's\_love) \leftarrow\} \rangle$$

The rule in this new update will reject the rule $maintenance(husband's\_love) \leftarrow$ in P_1 and she won't consider as a goal to have the husband's love. Furthermore, the cheated wife will no longer consider as a goal to please her mother-in-law.

In this example, we illustrate several aspects of how an agent framework with a DLP representing its goal base, can be used. First, we can represent more concrete goals using logic rules, e.g., when the wife agent had the maintenance goal of having her husband's love, she had the more concrete goal of pleasing his mother. Second, representing the norms of society, e.g., when the agent investigated in the internet how the dinner should be, in this case, red wine with lamb and white wine with fish. Third, dropping goals, when the agent realized that the goal of preparing lamb with red wine is not achievable (since there is no red wine) the agent drops this goal, and when the agent prepared the fish and arranged the white wine the goal of making dinner was dropped. Fourth, knowledge updates, when the agent finds out that her husband is cheating her with another girl, she updates negatively the goal of having the love of her husband, and consequently, the goal of pleasing her mother-in-law is abandoned.

6 Conclusions

In this paper, we introduced a simple agent framework with the purpose of introducing the agent's goal base as a Dynamic Logic Program. We investigated some properties of this framework. We were able to express, in a simple manner, maintenance and achievement goals, as well as identify some situations where the agent would need to adopt and drop goals, and how this could be done in this framework.

Since the objective of this paper was to investigate the use of DLP as the goal base of an agent, we didn't investigate any additional properties we could have by also using the belief base as a DLP. We also didn't give an adequate solution for conflicting intentions, since it would probably be also necessary to analyze the plans of the agent as well as its resources [19] to be able to conclude which goals to commit to.

Further investigation could also be done to solve possible conflicts in the social point of view of the agent. For example, if the agent considers the opinion of his mother and father equally, it would be necessary to have a mechanism to solve the conflicts since the agent doesn't prefer any one of them more than the other. [11] introduces the concept of *Multi Dimensional Dynamic Logic Programming* (MDLP) that could represent an agent's social point of view. Further investigation could be made in trying to incorporate the social point of view of an agent as a MDLP in our agent framework.

References

1. J. J. Alferes, F. Banti, A. Brogi, and J. A. Leite. The refined extension principle for semantics of dynamic logic programming. *Studia Logica*, 79(1), 2005.
2. J. J. Alferes, J. Leite, L. M. Pereira, H. Przymusinska, and T. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *Journal of Logic Programming*, 45(1-3):43–70, 2000.
3. Fahiem Bacchus and Adam J. Grove. Utility independence in a qualitative decision theory. In *KR*, pages 542–552, 1996.
4. M. Bratman. *Intentions, Plans and Practical Reason*. Harvard University Press, 1987.
5. M. Dastani, M. B. van Riemsdijk, and J.-J. Ch. Meyer. Programming multi-agent systems in 3APL. In *Multi-Agent Programming: Languages, Platforms and Applications*, chapter 2. Springer, 2005.
6. F. Dignum and R. Conte. Intentional agents and goal formation. In *Intelligent Agents IV*, volume 1365 of *LNAI*, pages 231–243, 1998.
7. K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming with declarative goals. In *Intelligent Agents VII*, volume 1986 of *LNAI*, pages 228–243. Springer, 2000.
8. N. R. Jennings, T. J. Norman, P. Faratin, P. O'Brien, and B. Odgers. Autonomous agents for business process management. *Applied Artificial Intelligence*, 14(2):145–189, 2000.
9. J. Leite. *Evolving Knowledge Bases*. IOS press, 2003.
10. J. Leite. On some differences between semantics of logic program updates. In *IBERAMIA'04*, volume 3315 of *LNAI*, pages 375–385. Springer, 2004.
11. J. Leite, J. J. Alferes, and L. M. Pereira. On the use of multi-dimensional dynamic logic programming to represent societal agents' viewpoints. In *EPIA'01*, volume 2258 of *LNAI*, pages 276–289. Springer, 2001.
12. J. Leite, J. J. Alferes, and L. M. Pereira. Minerva - a dynamic logic programming agent architecture. In *Intelligent Agents VIII*, volume 2333 of *LNAI*. Springer, 2002.
13. J. Leite and L. M. Pereira. Generalizing updates: From models to programs. In *LPKR'97*, volume 1471 of *LNAI*, pages 224–246. Springer, 1998.
14. Á. 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 *DALT'03*, volume 2990 of *LNAI*, pages 135–154. Springer, 2004.
15. V. Nigam and J. Leite. Incorporating knowledge updates in 3apl. In *PROMAS'06*, 2006.
16. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting & avoiding interference between goals in intelligent agents. In *IJCAI'03*, pages 721–726. Morgan Kaufmann, 2003.
17. B. van Riemsdijk, M. Dastani, F. Dignum, and J.-J. Ch. Meyer. Dynamics of declarative goals in agent programming. In *DALT'04*, volume 3476 of *LNAI*, pages 1–18, 2004.
18. M. B. van Riemsdijk, M. Dastani, and J.-J. Ch. Meyer. Semantics of declarative goals in agent programming. In *AAMAS'05*. ACM Press, 2005.
19. M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *KR'02*. Morgan Kaufmann, 2002.
20. M. Wooldridge. *Multi-agent systems : an introduction*. Wiley, 2001.