Agent-Oriented Programming with Underlying Ontological Reasoning

Álvaro F. Moreira[1], Renata Vieira[2], Rafael H. Bordini[3], and Jomi Hübner[4]

[1] Universidade Federal do Rio Grande do Sul
afmoreira@inf.ufrgs.br

[2] Universidade do Vale do Rio dos Sinos
renata@exatas.unisinos.br

[3] University of Durham,
R.Bordini@durham.ac.uk

[4] Universidade Regional de Blumenau,
jomi@inf.furb.br

Abstract. Developing applications that make use of machine-readable knowledge sources as promised by the Semantic Web vision is attracting much of current research interest; this vision is also affecting important trends in Computer Science such as Grid and Mobile computing. In this paper we propose a version of the BDI agent-oriented programming language AgentSpeak that is based on Description Logic (DL). In this approach, the belief base of an agent contains the definition of complex concepts, besides specific factual knowledge. The advantages of combining AgentSpeak with DL are: (i) queries to the belief base are more expressive as their results do not rely only on explicit knowledge but can be inferred from the ontology; (ii) the notion of belief update is refined since the (ontological) consistency of a belief addition can be checked; (iii) the search for a plan for handling an event is more flexible as it is not based solely on unification but on the subsumption relation between concepts; and (iv) agents may share knowledge by using ontology languages such as the Ontology Web Language (OWL). Extending agent programming languages with DL can have a significant impact on the development of multi-agent systems for the Semantic Web.

1 Introduction

Developing applications that make full use of machine-readable knowledge sources as promised by the Semantic Web vision is attracting much of current research interest. More than that, Semantic Web technology is also being used as the basis for other important trends in Computer Science such as Grid Computing [12] and Ubiquitous Computing [8].

Among the key components of the Semantic Web are *domain ontologies* [24]. They are responsible for the specification of the domain knowledge, and as they can be expressed logically, they can be the basis for sound reasoning in the specified domain. Several ontologies are being proposed for the development of specific applications [9,

18, 11, 25]. Another key component of the Semantic Web technology is the work on intelligent agents which are responsible for making use of the available knowledge, autonomously interacting with other agents, so as to act on the user's best interest.

In this work we bring these two key Semantic Web components together by proposing an extension to the BDI agent programming language AgentSpeak [22]; there has been much work on extending this language so that it becomes a fully-fledged programming language for multi-agent systems [19, 1, 5]. The AgentSpeak extension proposed here is based on Description Logic (DL) [2] rather than classical (predicate) logic; we shall call them AgentSpeak-DL and predicate-logic AgentSpeak (for emphasis), respectively. With DL, the belief base of an AgentSpeak agent consists of the definition of complex concepts and relationships among them, as well as specific factual knowledge (or beliefs, in this case) — in DL terminology, these are called TBox and ABox respectively. To the best of our knowledge, this is the first work to address ontological reasoning as an underlying mechanism within an agent-oriented programming language.

Description logics are at the core of widely known ontology languages, such as the Ontology Web Language (OWL) [17]. An extension of AgentSpeak with underlying automatic reasoning over ontologies expressed in such languages can have a major impact on the development of agents and multi-agent systems that can operate in a Semantic Web context. Although applications for the Semantic Web are already being developed, often based on the Agents paradigm, most such development is being done on a completely *ad hoc* fashion as far as agent-oriented programming is concerned. This work contributes to the development of multi-agent systems for Semantic Web applications in a principled way; also, the ontological reasoning being embedded in the agent programming language itself facilitates the tasks involved in using such languages.

On the other hand, agent-oriented programming languages, granting that considerable improvement has been achieved since the paradigm was first thought out [23], are still in early stages of development and have clear shortfalls as far as their use in the software industry is concerned, as pointed out by Kinny in [16]. One such shortfalls has precisely to do with the unrealistic simplicity of the way in which belief bases often are implemented in these languages. Industrial-strength applications often require massive use of data, thus a belief base that is simply an unstructured collection of ground predicates is just not good enough. This work contributes to extending agent programming languages with belief bases that have a more sophisticated underlying structure. It is also promising that the particular structure proposed here follows the work on ontologies which is having a huge impact in many areas of computer science, and of course this extension is particularly desired in these days of Semantic Web fervour.

The remainder of this paper is organised as follows. In Section 2, we describe the syntax of the AgentSpeak language based on Description Logic and we also explain briefly the main characteristics of the subset of Description logic used for defining ontologies in the context of this paper. Section 3 presents the modifications that are necessary, in the formal semantics of predicate-logic AgentSpeak, as a consequence of introducing ontological description and reasoning. Each modification in the semantics is followed by a small example giving the intuition supporting it and illustrating the prac-

tical benefits of incorporating ontological description and reasoning into AgentSpeak. In the final section we draw some conclusions and discuss future work.

2 The AgentSpeak-DL Language

The syntax of AgentSpeak-DL is essentially the same as the syntax of predicate-logic AgentSpeak [7], the only difference being that in predicate-logic AgentSpeak the belief base of an agent consists solely of ground atoms, whereas in AgentSpeak-DL the belief base contains the definition of complex concepts and relationships, besides factual knowledge. An AgentSpeak-DL agent specification *ag* is thus given by an ontology *Ont* and a set *ps* of plans, as defined by the grammar in Figure 1.

$$
\begin{aligned}
ag \quad &::= Ont \quad ps \\[4pt]
Ont \quad &::= TBox \; ABox \\
TBox &::= C_1 \equiv D_1 \ldots C_n \equiv D_n \qquad (n \geq 0) \\
&\quad | \quad C_1 \sqsubseteq D_1 \ldots C_n \sqsubseteq D_n \qquad (n \geq 0) \\
&\quad | \quad R_1 \equiv S_1 \ldots R_n \equiv S_n \qquad (n \geq 0) \\
&\quad | \quad R_1 \sqsubseteq S_1 \ldots R_n \sqsubseteq R_n \qquad (n \geq 0) \\
C, D \quad &::= A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C \\
R, S \quad &::= P \mid R \sqcap S \mid R \sqcup S \\
ABox &::= at_1 \ldots at_n \qquad (n \geq 0) \\
at \quad &::= C(t) \mid R(t_1, t_2) \\[4pt]
ps \quad &::= p_1 \ldots p_n \qquad (n \geq 1) \\
p \quad &::= te : ct \leftarrow h \\
te \quad &::= +at \mid -at \mid +g \mid -g \\
ct \quad &::= at \mid \neg at \mid ct \wedge ct \mid \mathsf{T} \\
h \quad &::= h_1; \mathsf{T} \mid \mathsf{T} \\
h_1 \quad &::= a \mid g \mid u \mid h_1; h_1 \\
g \quad &::= !at \mid ?at \\
u \quad &::= +at \mid -at
\end{aligned}
$$

Fig. 1. AgentSpeak-DL Syntax.

As seen in the grammar above, an ontology consists of a TBox and an ABox. A TBox is a set of class and property descriptions, and axioms establishing equivalence and subsumption relationships between classes (unary predicates) and properties (binary predicates). In the grammar, the metavariables C and D represent classes; $C \equiv D$ asserts that both classes are equivalent; and $C \sqsubseteq D$ asserts that class C is subsumed by class D. Similarly for when '$\equiv$' and '$\sqsubseteq$' are applied to properties R and S.

The definition of classes and properties assumes the existence of identifiers for primitive (i.e., not defined) classes and properties (A and P respectively). The metavariable

A stands for names of primitive classes (i.e., predefined classes) as well as names chosen to identify constructed classes defined in the TBox; the metavariable *P* stands for primitive properties and for names given to constructed properties. New classes and properties can be constructed by using certain constructors, such as $\sqcup$ and $\sqcap$, for instance ($\sqcap$ and $\sqcup$ represent the intersection and the union of two concepts, respectively).

An ABox describes the state of an application domain by asserting that certain individuals are instances of certain classes and that certain individuals are related by a property. In this paper, we assume that the class and property constructors are those available for the $\mathcal{ALC}$ description logic [3].

A complete definition of all the class and property constructors of Description Logic [2] is out of the scope of this paper. Therefore, in order to keep the formal treatment and examples simple, from now on we will consider TBoxes with classes only, and ABoxes with instances of those classes; that is, we assume a simplified language with no properties.

A plan is formed by a *triggering event* — denoting the events for which that plan should be considered *relevant* — followed by a conjunction of belief literals representing a *context*. The context must be a logical consequence of that agent's current beliefs for the plan to be *applicable*. The remainder of the plan is a sequence of basic actions or (sub)goals that the agent has to achieve (or test) when the plan, if applicable, is chosen for execution.

AgentSpeak distinguishes two types of goals: *achievement goals* and *test goals*. Achievement and test goals are predicates (as for beliefs) prefixed with operators '!' and '?' respectively. Achievement goals state that the agent wants to achieve a state of the world where the associated predicate is true. (In practice, these initiate the execution of *subplans*.) A *test goal* corresponds to a query to the agent's belief base.

Events, which initiate the execution of plans, can be internal, when a subgoal needs to be achieved, or external, when generated from belief updates as a result of perceiving the environment. There are two types of triggering events that can be used in plans: those related to the *addition* ('+') and *deletion* ('-') of mental attitudes (beliefs or goals). Plans also refer to the *basic actions* (represented by the metavariable *a* in the grammar above) that an agent is able to perform in its environment.

Throughout the paper, we illustrate the practical impact of ontological reasoning with simple examples related to the well-known scenario of smart meeting-room applications [8]. In Figure 2, we give examples of AgentSpeak-DL plans that were written to deal with the event resulting from the arrival of a presenter/speaker in the room.

The first plan in Figure 2 says that if a presenter of a paper is late he is rescheduled to the end of the session (and the session goes on). If an invited speaker is late, apologies are given to the audience and the speaker is announced. The third plan just announces any presenter (presenter being a concept that is the union of paperPresenter and invitedSpeaker) if he is not late.

An example of TBox components using the language above is as follows:

$$presenter \equiv invitedSpeaker \sqcup paperPresenter$$
$$attendee \equiv person \sqcap registered \sqcap \neg presenter \dots$$

This TBox asserts that the concept *presenter* is equivalent to the concept *invited speaker or paper presenter* and the concept *attendee* is equivalent to the concept *regis-*

```
+paperPresenter(P)
   : late(P)
       ← !reschedule(P).

+invitedSpeaker(P)
   : late(P)
       ← !apologise;
          !announce(P).

+presenter(P)
   : ¬late(P)
       ← !announce(P).
```

Fig. 2. Examples of AgentSpeak plans.

tered person which is not a presenter. Examples of elements of an ABox defined with respect to the TBox above are:

$$invitedSpeaker(john)$$
$$paperPresenter(mary)$$

We now proceed to discuss the implications of incorporating ontologies in AgentSpeak to its formal semantics.

3 Semantics of AgentSpeak-DL

The reasoning cycle of an AgentSpeak agent follows a set of steps. The graph in Figure 3 shows all possible transitions between the various steps in an agent's reasoning cycle (the labels in the nodes name each step in the cycle). The set of labels used is {SelEv, RelPl, ApplPl, SelAppl, AddIM, SelInt, ExecInt, ClrInt}; they stand for: selecting an event from the set of events, retrieving all relevant plans, checking which of those are applicable, selecting one particular applicable plan (the intended means), adding the new intended means to the set of intentions, selecting an intention, executing the select intention, and clearing an intention or intended means that may have finished in the previous step.

In this section we present an operational semantics for AgentSpeak-DL that formalises some of the possible transitions depicted in the reasoning cycle of Figure 3. Operational semantics is a widely used method for giving semantics to programming languages and studying their properties [21].

The semantic rules for the steps in the reasoning cycle are essentially the same in AgentSpeak-DL as for predicate-logic AgentSpeak, with the exception of the following aspects that are affected by the introduction of ontological reasoning:
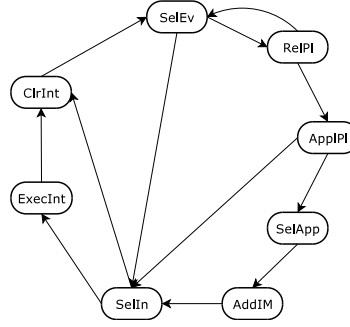
Fig. 3. Transitions between Reasoning Cycle Steps.

- *plan search*: performed in the steps responsible for collecting relevant and applicable plans, and selecting one plan among the set of applicable plans (steps RelPl, ApplPl, and SelAppl of Figure 3, respectively);
- *querying the belief base*: performed in step ExecInt of Figure 3); and
- *belief updating*: also performed in step ExecInt of Figure 3).

A complete version of an operational semantic for AgentSpeak is given in [7]. For this reason, in this work, we give only the semantic rules of AgentSpeak-DL that are different from their counterparts in the operational semantics of AgentSpeak.

3.1 Configuration of the Transition System

The operational semantics is given by a set of rules that define a transition relation between configurations $\langle ag, C, T, s \rangle$ where:

- An agent program *ag* is, as defined above, a set of beliefs and a set of plans. Note that in predicate-logic AgentSpeak, the set of beliefs is simply a collection of ground atoms. In AgentSpeak-DL the belief base is an ontology.
- An agent's circumstance *C* is a tuple $\langle I, E, A \rangle$ where:
  - *I* is a set of *intentions* $\{i, i', \ldots\}$. Each intention *i* is a stack of partially instantiated plans.
  - *E* is a set of *events* $\{(te, i), (te', i'), \ldots\}$. Each event is a pair (te, i), where *te* is a triggering event and *i* is an intention (a stack of plans in case of an internal event or $\top$ representing an external event).
  - *A* is a set of *actions* to be performed in the environment.
- It helps in giving the semantics to use a structure *T* which keeps track of temporary information that is required in subsequent stages but only within a single reasoning cycle. *T* is the tuple $\langle R, Ap, \iota, \varepsilon, \rho \rangle$ with such temporary information; it has as components (i) *R* for the set of *relevant plans* (for the event being handled); (ii) *Ap* for the set of *applicable plans* (the relevant plans whose context are true), and (iii) ι, ε, and ρ keep record of a particular intention, event and applicable plan (respectively) being considered along the execution of an agent.

137

– The current step *s* within an agent's reasoning cycle is symbolically annotated by $s \in \{\mathsf{SelEv}, \mathsf{RelPl}, \mathsf{ApplPl}, \mathsf{SelAppl}, \mathsf{AddIM}, \mathsf{SelInt}, \mathsf{ExecInt}, \mathsf{ClrInt}\}$ (as seen in Figure 3).

In the general case, an agent's initial configuration is $\langle ag, C, T, \mathsf{SelEv}\rangle$, where *ag* is as given by the agent program, and all components of *C* and *T* are empty.

In order to keep the semantic rules elegant, we adopt the following notations:

– If *C* is an AgentSpeak agent circumstance, we write C_E to make reference to the component *E* of *C*. Similarly for all the other components of a configuration.
– We write $i[p]$ to denote an intention *i* that has plan *p* on its top.

3.2 Plan Search in AgentSpeak-DL

The reasoning cycle of an agent can be better understood by assuming that it starts with the selection of an event from the set of events (this step assumes the existence of a selection function S_E). The next step in the reasoning cycle is the search for relevant plans for dealing with the selected event. In the semantics this is formalised by the following rules.

Rule **Rel$_1$** below initialises the *R* component of *T* with the set of relevant plans determined by the auxiliary function $\mathsf{RelPlans}$, and sets the reasoning cycle to the step responsible for determining the applicable plans among those in the *R* component.

$$\frac{T_\varepsilon = \langle te, i\rangle \qquad \mathsf{RelPlans}(ag_{ps}, te) \neq \{\}}{\langle ag, C, T, \mathsf{RelPl}\rangle \longrightarrow \langle ag, C, T', \mathsf{ApplPl}\rangle} \quad (\textbf{Rel}_1)$$

$$where: \ T'_R = \mathsf{RelPlans}(ag_{ps}, te)$$

$$\frac{T_\varepsilon = \langle te, i\rangle \qquad \mathsf{RelPlans}(ag_{ps}, te) = \{\}}{\langle ag, C, T, \mathsf{RelPl}\rangle \longrightarrow \langle ag, C, T, \mathsf{SelEv}\rangle} \quad (\textbf{Rel}_2)$$

If there are no relevant plans for an event, it is simply discarded and, with it, the associated intention. In this case the cycle starts again with the selection of other event from the set of events (rule **Rel$_2$**). If there is no event to handle, the cycle skips to the intention execution.

In predicate-logic AgentSpeak, a plan is considered relevant in relation to a triggering event if it has been written to deal specifically with that event. In practice, that is checked in predicate-logic AgentSpeak by trying to unify the triggering event part of the plan with the triggering event that has been selected from the set *E* of events for begin handled during this reasoning cycle.

The auxiliary function $\mathsf{RelPlans}$ for predicate-logic AgentSpeak is then defined as follows (below, if *p* is a plan of the form $te : ct \leftarrow h$, we define $\mathsf{TrEv}(p) = te$).

Definition 1 *Given the plans ps of an agent and a triggering event e, the set $\mathsf{RelPlans}(ps, e)$ of relevant plans is given as follows:*

$$\mathsf{RelPlans}(ps, e) = \{(p, \theta) \mid p \in ps \ and$$

$$\theta \ is \ a \ mgu \ s.t. \ e\theta = \mathsf{TrEv}(p)\theta\}.$$

It is important to remark that, in predicate-logic AgentSpeak, the key mechanism used for searching relevant plans in the agent's plan library is unification. This means that the programmer has to write plans specifically for each possible type of event. The only degree of generality is obtained by the use of variables in the triggering event of plans.

When ontological reasoning is used (instead of unification only), a plan is considered relevant in relation to an event not only if it has been written specifically to deal with that event, but also if its triggering event has a more general relevance, in the sense that it subsumes the actual event. In practice, that is checked by: (i) finding a plan whose triggering event name is related (in the ontology) by the subsumption relation to the event name that has been selected for handling; and (ii) unifying the terms that are arguments for the event and the plan's triggering event.

In the formal semantics of AgentSpeak-DL, rules **Rel**$_1$ and **Rel**$_2$ still apply, but the auxiliary function RelPlans for AgentSpeak-DL has to be redefined as follows (recall that C is a metavariable for classes of an ontology):

Definition 2 *Given the plans ps and the ontology Ont of an agent, and a triggering event op $C(t)$ where op $\in \{+, -, +!, +?, -!, -?\}$, the set RelPlans(Ont, ps, op $C(t)$) is a set of pairs (p, θ) such that $p \in ps$, with TrEv$(p) = op'\ D(t')$, such that*

- *op $= op'$,*
- *$\theta =$ mgu(t, t'), and*
- *Ont $\models C \sqsubseteq D$*

As an example let us consider the case of checking for plans that are relevant for a particular event in the smart meeting-room scenario again. Suppose that a sensor in a smart meeting-room has somehow detected in the environment the arrival of the invited speaker *john*. This causes the addition of the external event $\langle +\texttt{invitedSpeaker(john)}, \top \rangle$ to the set of events. Suppose also that *invitedSpeaker $\sqsubseteq$ presenter* can be inferred from the ontology. Under these circumstances, a plan with triggering event $+\texttt{presenter(X)}$ is also considered relevant for dealing with the event. Observe that using subsumption instead of unification alone as the mechanism for selecting relevant plans potentially results in a larger set of plans than in predicate-logic AgentSpeak.

A plan is applicable if it is relevant and its context is a logical consequence of the agent's beliefs. Rules **Appl**$_1$ and **Appl**$_2$ formalise the step of the reasoning cycle that determines the applicable plans from the set of relevant plans.

$$\frac{\text{AppPlans}(ag_{bs}, T_R) \neq \{\}}{\langle ag, C, T, \text{ApplPl} \rangle \longrightarrow \langle ag, C, T', \text{SelAppl} \rangle} \quad (\textbf{Appl}_1)$$

$$\text{where: } T'_{Ap} = \text{AppPlans}(ag_{bs}, T_R)$$

$$\frac{\text{AppPlans}(ag_{bs}, T_R) = \{\}}{\langle ag, C, T, \text{ApplPl} \rangle \longrightarrow \langle ag, C, T, \text{SelInt} \rangle} \quad (\textbf{Appl}_2)$$

The rule **Appl**$_1$ initialises the T_{Ap} component with the set of applicable plans; **Appl**$_2$ is the rule for the case where there are no applicable plans (to avoid discussing details of a possible plan failure handling mechanism, we assume the event is simply discarded). Both rules depends on the auxiliary function ApplPlans, which is defined as follows in predicate-logic AgentSpeak:

Definition 3 *Given a set of relevant plans R and the beliefs bs of an agent, the set of applicable plans AppPlans(bs, R) is defined as follows:*

$$AppPlans(bs, R) = \{(p, \theta' \circ \theta) \mid (p, \theta) \in R \ and$$

$$\theta' \ is \ s.t. \ bs \models Ctxt(p)\theta\theta'\}.$$

Observe that the context of a plan is a conjunction of literals and, as the belief base is made only of a set of ground atomic predicates, the problem of checking if the plan's context is a logical consequence of the belief base reduces to the problem of checking membership of context literals to the set of beliefs.

In AgentSpeak-DL, a plan is applicable if it is relevant and its context can be inferred from the whole ontology forming the belief base. A plan's context is a conjunction of literals[1] (l is either $C(t)$ or $\neg C(t)$). We can say that $Ont \models l_1 \wedge \ldots \wedge l_n$ if, and only if, $Ont \models l_i$ for $i = 1 \ldots n$. The auxiliary function for checking, from a set of relevant plans, which are the applicable ones is then formalised as follows.

Definition 4 *Given a set of relevant plans R and ontology Ont of an agent, the set of applicable plans AppPlans(Ont, R) is defined as follows:*

$$AppPlans(bs, R) = \{(p, \theta' \circ \theta) \mid (p, \theta) \in R \ and$$

$$\theta' \ is \ s.t. \ Ont \models Ctxt(p)\theta\theta'\}.$$

Again, due to the fact that the belief base is structured and that reasoning is based on subsumption as well as instantiation, the resulting set of applicable plans might be larger than in predicate-logic AgentSpeak.

More than one plan can be considered applicable for dealing with an event at a given moment in time. The rule SelAppl in the formal semantics of predicate-logic AgentSpeak assumes the existence of a (given, application-specific) selection function S_{Ap} that selects a plan from a set of applicable plans T_{Ap}. The plan selected is then assigned to the T_ρ component of the configuration indicating, for the next steps in the reasoning cycle, that this plan has to be added to the agent's intentions.

$$\frac{S_{Ap}(T_{Ap}) = (p, \theta)}{\langle ag, C, T, \text{SelAppl} \rangle \longrightarrow \langle ag, C, T', \text{AddIM} \rangle} \qquad \textbf{(SelAppl)}$$

$$\textit{where: } T'_\rho = (p, \theta)$$

---

[1] Note that in the context of the Semantic Web, open world is often assumed, so negation here, in contrast to as originally defined for predicate-logic AgentSpeak, is "strong negation", in the usual sense in logic programming.

In predicate-logic AgentSpeak, users define the applicable plan selection function (S_{Ap}) in a way that suits that particular application. For example, if in a certain domain there are known probabilities of the chance of success or resulting quality of achieved tasks associated with various plans, this can be easily used to specify such function. However, note that, in predicate-logic AgentSpeak, the predicate in the triggering event of all the plans in the set of applicable plans are exactly the same.

On the contrary, in AgentSpeak-DL, because of the way the relevant and applicable plans are determined it is possible that plans with triggering events $+\texttt{presenter(X)}$ and $+\texttt{invitedSpeaker(X)}$ were both considered relevant and applicable for handling an event $\langle +\texttt{invitedSpeaker(john)}, \top \rangle$. The function S_{Ap} in rule **SelAppl**, could be used to select, for example, the *the least general* plan among those in the set of applicable plans. To allow this to happen, the semantic rule has to be slightly modified so as to include as argument to S_{Ap} the event that has triggered the search for a plan. Here, the selected plan should be the one with triggering event $+\texttt{invitedSpeaker}$ as probably this plan has been written to deal more particularly with the case of invited speakers arriving in the room, rather than the more general plan which can be used for other types of presenters as well. On the other hand, if the particular plan for invited speaker is not applicable (e.g., because it involves alerting the session chair of the arrival of the celebrity speaker but the chair is not present), instead of the agent not acting at all for lack of applicable plans, the more general plan for speakers can then be tried, the relevance being determined by the underlying ontology instead.

$$\frac{T_\varepsilon = \langle te, i \rangle \qquad S_{Ap}(T_{Ap}, te) = (p, \theta)}{\langle ag, C, T, \mathsf{SelAppl} \rangle \longrightarrow \langle ag, C, T', \mathsf{AddIM} \rangle} \qquad \textbf{(SelApplOnt)}$$

$$where: \ T'_\rho = (p, \theta)$$

3.3 Querying the Belief Base

Events can be classified as external or internal (depending on whether they were generated from the agent's perception of the environment, or whether they were generated by goal additions during the execution of other plans, respectively). If the event is external, a new intention is created and its single plan is the plan p annotated in the ρ component in the previous step of the reasoning cycle. If the event is internal, rule **IntEv** says that the plan in ρ should be put on the top of the intention associated with the event.

This step uses an agent-specific function (S_I) that selects the intention (i.e., a stack of plans) to be executed next. When the set of intentions is empty the reasoning cycle is simply restarted. The plan to be executed is always the one at the top of the intention that has been selected. Agents can execute actions, achievement goals, test goals, or belief base updates.

Both the execution of actions and the execution of achievement goals are not affected by the introduction of ontological reasoning, so their semantics are exactly the same as their counterparts in predicate-logic AgentSpeak: the execution of actions means that the AgentSpeak interpreter tells other architectural components to perform the respective action on the environment, hence changing it; and the execution

of achievement goals puts a new internal event in the set of events. This event will be then eventually selected at a later reasoning cycle.

The evaluation of a test goal $?at$, however, is more expressive in AgentSpeak-DL than in predicate-logic AgentSpeak. In predicate-logic AgentSpeak, the execution of a test goal consists in testing if at is a logical consequence of the agent's beliefs. The function defined below returns a set of most general unifiers all of which make the the formula at a logical consequence of a set of formulæ bs:

Definition 5 *Given a set of formulæ bs and a formula at, the set of substitutions* $\mathsf{Test}(bs, at)$ *produced by testing at against bs is defined as follows:*

$$\mathsf{Test}(bs, at) = \{\theta \mid bs \models at\theta\}.$$

This auxiliary function is then used in the formal semantics by the rules $\mathbf{Test}_1$ and $\mathbf{Test}_2$ below. If the test goal succeeds (rule $\mathbf{Test}_1$), the substitution is applied to the whole intended means, and the reasoning cycle can carry on. If that is not the case, it may be that the test goal is used as a triggering event of a plan, which is used by programmers to formulate more sophisticated queries[2]. Rule $\mathbf{Test}_2$ is used in such case: it generates an internal event, which may eventually trigger the execution of a plan (explicitly created to carry out a complex query).

$$\frac{T_\iota = i[head \leftarrow ?at\,;h] \qquad \mathsf{Test}(ag_{bs}, at) \neq \{\}}{\langle ag, C, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag, C, T, \mathsf{ClrInt}\rangle} \tag{$\mathbf{Test}_1$}$$

$$\text{where:} \quad C_I' = (C_I \setminus \{T_\iota\}) \cup \{(i[head \leftarrow h])\theta\}$$
$$\theta \in \mathsf{Test}(ag_{bs}, at)$$

$$\frac{T_\iota = i[head \leftarrow ?at\,;h] \qquad \mathsf{Test}(ag_{bs}, at) = \{\}}{\langle ag, C, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag, C, T, \mathsf{ClrInt}\rangle} \tag{$\mathbf{Test}_2$}$$

$$\text{where:} \quad C_E' = C_E \cup \{\langle +?at, i[head \leftarrow h]\rangle\}$$
$$C_I' = C_I \setminus \{T_\iota\}$$

In AgentSpeak-DL the semantic rules for the evaluation of a test goal $?C(t)$ are exactly as the rules $\mathbf{Test}_1$ and $\mathbf{Test}_2$ above. However, the function Test checks whether the formula $C(t)$ is a logical consequence of the (more structured) agent's belief base, now based on an ontology. The auxiliary function Test is redefined as follows:

Definition 6 *Given a set of formulæ Ont and a formula ?at, the set of substitutions* $\mathsf{Test}(Ont, at)$ *is given by*

$$\mathsf{Test}(Ont, at) = \{\theta \mid Ont \models at\theta\}.$$

---

[2] Note that this was not clear in the original definition of AgentSpeak(L). In our work on extensions of AgentSpeak we have given its semantics in this way as it allows a complex plan to be used for determining the values to be part of the substitution resulting from a test goal (rather than just retrieving specific values previously stored in the belief base).

Observe that this definition is similar to the definition of the auxiliary function Test given for predicate-logic AgentSpeak. The crucial difference is that now the reasoning capabilities of DL allows agents to infer knowledge that is implicit in the ontology. As an example, suppose that the agent belief base does not refer to instances of *attendee*, but has instead the facts *invitedSpeaker*(*john*) and *paperPresenter*(*mary*). A test goal such as ?*attendee*(*A*) succeeds in this case producing substitutions that map *A* to *john* and *mary*.

3.4 Belief Updating

In predicate-logic AgentSpeak, the addition of a belief to the belief base has no further implications (apart from a possible new event that is added the set of events E to deal with any change in the belief base). The rule **AddBel** below formalises belief addition in predicate-logic AgentSpeak: the formula $+b$ is removed from the body of the plan and the set of intentions is updated properly. There is a similar rule for deletion of beliefs ($-b$). In practice, this mechanism for adding and removing beliefs is useful for the agent to have "mental notes", which can be quite useful at times (in practical programming). These beliefs should not normally be confused with beliefs acquired from perception of the environment.

In the rules below, $bs' = bs + b$ means that bs' is as bs except that $bs' \models b$.

$$\frac{T_\iota = i[head \leftarrow +b\,;h]}{\langle ag, C, T, \mathsf{ExecInt} \rangle \longrightarrow \langle ag', C', T, \mathsf{ClrInt} \rangle} \quad \textbf{(AddBel)}$$

$$\begin{aligned} where: \ ag'_{bs} &= ag_{bs} + b \\ C'_E &= C_E \cup \{\langle +b, \mathsf{T} \rangle\} \\ C'_I &= (C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\} \end{aligned}$$

In AgentSpeak-DL, an ABox contains class assertions $C(t)$ and property assertions $R(t_1, \ldots, t_n)$. The representation of such information should, of course, be consistent with the TBox. Suppose for instance that the TBox is such that can be inferred from it that the concepts *chair* and *bestPaperWinner* are disjoint. Clearly, if the ABox asserts that *chair*(*mary*), the assertion *bestPaperWinner*(*mary*) is not to be added to it, otherwise the belief base would become inconsistent.

Observe that in predicate-logic AgentSpeak the belief base consists solely of ground atomic formulas so consistency is not a major problem. In AgentSpeak-DL, the addition of assertions to the agent ABox is only allowed if the result is consistent with the ontology. Approaches for checking consistency of an ABox with respect to a TBox are discussed in detail in [3]. The semantic rule **AddBel** has to be modified accordingly:

$$\frac{T_\iota = i[head \leftarrow +b\,;h] \qquad ag_{Ont} \cup \{b\} \ is \ consistent}{\langle ag, C, T, \mathsf{ExecInt} \rangle \longrightarrow \langle ag', C', T, \mathsf{ClrInt} \rangle} \quad \textbf{(AddBelOnt)}$$

$$\begin{aligned} where: \ ag'_{Ont} &= ag_{Ont} \cup \{b\} \\ C'_E &= C_E \cup \{\langle +b, \mathsf{T} \rangle\} \\ C'_I &= (C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\} \end{aligned}$$

There is a similar rule for belief deletions, but it is trivially defined based on the one above, so we do not include it explicitly here.

Clearly, if a belief addition fails due to inconsistency with the underlying ontology, we cannot simply ignore that event. In practice, a plan failure mechanism must be activated in that case. However, we avoid formalising this here as such mechanism is quite involved and would not help us in describing the main issues associated with ontologies, which is the focus of this paper.

The rule above is specific to addition of beliefs that arise from the execution of plans. Recall that these are used as mental notes that the agent uses for its own processing. What is formalised above in rule **AddBelOnt** should also apply for belief revision: when the ABox is changed as a consequence of perception of the environment, similar care must be taken to avoid the ABox becoming inconsistent with the TBox. As this is assumed to be part of the general agent architecture rather than part of the AgentSpeak interpreter itself, no formalisation is provided for this here, but it is important to emphasise that ontological consistency must be ensured during that belief revision process as much as during belief additions (and deletions) as formalised above.

The reasoning cycle finishes by removing from the set of intentions an intended means or a whole intention that have been fully executed.

4 Conclusions and Future Work

This paper has formalised the changes in the semantics of AgentSpeak that were required for agent-oriented programming with underlying ontological reasoning. The main improvements to AgentSpeak resulting from the extension based on a description logic are: (i) the search for a plan (in the agent's plan library) that is relevant for dealing with a particular event is more flexible as this is not based solely on unification, but also on the subsumption relation between concepts; (ii) queries to the belief base are more expressive as their result do not depend only on explicit knowledge but can be inferred from the ontology; (iii) the notion of belief update is refined so that a property about an individual can only be added if the resulting belief base is consistent with the concept description; and (iv) agents may share knowledge by using web ontology languages such as OWL. The advantages of the more specialised reasoning that is possible using ontologies also represent an increase in computational cost of agent programs. However, this trade-off between increased expressivity and possible decrease in computational efficiency in the context of this work has not been considered as yet.

With this paper, we expect to contribute towards showing that extending an agent programming language with the descriptive and reasoning power of description logics can have a significant impact on the way agent-oriented programming works in general, and in particular for the development of Semantic Web applications using the agent-oriented paradigm. In fact, this extension makes agent-oriented programming more directly suitable for other application areas that are currently very important such as Grid and Ubiquitous computing. It also creates perspectives for elaborate forms of agent migration, where plans carefully written to use ontological descriptions can ease the process of agent adaptation to different societies [10].

In future work, we aim at in integrating what is proposed here with other ongoing activities related to agent-oriented programming and semantic web technology. To start with, we plan to improve the semantics of AgentSpeak-DL, to move from the simple $\mathcal{ALC}$ used here to more expressive DLs such as those of OWL Lite and OWL DL [15]. The idea is to allow AgentSpeak-DL agents to use ontologies written in OWL, so that applications written in AgentSpeak-DL can be deployed on the Web and interoperate with other semantic web applications based on the OWL W3C standard (`http://www.w3.org/2004/OWL/`).

An interpreter for predicate-logic AgentSpeak has been implemented and is available *open source*; it is called ***Jason*** ([6], `http://jason.sourceforge.net`). An AgentSpeak-DL interpreter is currently being implemented, based on the formalisation presented here. ***Jason*** is in fact an implementation of a much extended version of AgentSpeak. It has various available features which are relevant for developing an AgentSpeak-DL interpreter; particularly, it implements the operational semantics of AgentSpeak as defined in [7], thus the semantic changes formalised in Section 3 can be directly transferred to the ***Jason*** code. However, ***Jason***'s inference engine needs to be extended to incorporate ontological reasoning, which can already be done by existing software such as [13, 20, 14]. We are currently considering the use of RACER [13] in particular for extending ***Jason*** so that belief bases can be written in OWL.

One of our planned future work is the integration of this work with the AgentSpeak extension presented in [19] that gives semantics to speech-act-based communication between AgentSpeak agents. In such integrated approach, agents in a society can refer to specific TBox components when exchanging messages. One of the features provided by ***Jason*** is that predicates have a (generic) list of "annotations"; in the context of this work, we can use that mechanism to specify the particular ontology to which each predicate belongs to. For example, the fact that an agent `a` has informed another agent about a fact `f` as defined in a given ontology `ont` can be expressed in that agent's belief base as `f[source(a),ontology("http://.../ont")]`.

An interesting issue associated with ontologies is that of how different ontologies for the same domain can be integrated. Recent work [4] has proposed the use of type theory in order to both detect discrepancies among ontologies as well as align them. We plan to investigate the use of type-theoretic approaches to provide our ontology-based agent-oriented programming framework with techniques for coping with ontological mismatch. The trade-off between added expressivity and computational costs of the extension of AgentSpeak proposed here should also be investigated.

Although the usefulness of combining ontological reasoning within an agent-oriented programming language seems quite clear (e.g., from the discussions presented in this paper), the implementation of practical applications are essential to fully support such claims. This also is planned as part of our future work.

References

1. D. Ancona, V. Mascardi, J. F. Hübner, and R. H. Bordini. Coo-AgentSpeak: Cooperation in AgentSpeak through plan exchange. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2004), New York, NY, 19–23 July*, pages 698–705, New York, NY, 2004. ACM Press.

2. F. Baader, D. Calvanese, D. N. D. McGuinness, and P. Patel-Schneider, editors. *Handbook of Description Logics*. Cambridge University Press, Cambridge, 2003.

3. F. Baader and W. Nutt. Basic description logics. In F. Baader, D. Calvanese, D. N. D. McGuinness, and P. Patel-Schneider, editors, *Handbook of Description Logics*, pages 43–95. Cambridge University Press, Cambridge, 2003.

4. R.-J. Beun, R. M. van Eijk, and H. Prüst. Ontological feedback in multiagent systems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2004)*, New York, NY, 19–23 July, 2004.

5. R. H. Bordini, A. L. C. Bazzan, R. O. Jannone, D. M. Basso, R. M. Vicari, and V. R. Lesser. AgentSpeak(XL): Efficient intention selection in BDI agents via decision-theoretic task scheduling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2002), 15–19 July, Bologna, Italy*, pages 1294–1302, New York, NY, 2002. ACM Press.

6. R. H. Bordini, J. F. Hübner, et al. *Jason: A Java-based AgentSpeak interpreter used with Saci for multi-agent distribution over the net*, manual, version 0.6 edition, Feb 2005. `http://jason.sourceforge.net/`.

7. R. H. Bordini and Á. F. Moreira. Proving BDI properties of agent-oriented programming languages: The asymmetry thesis principles in AgentSpeak(L). *Annals of Mathematics and Artificial Intelligence*, 42(1–3):197–226, Sept. 2004. Special Issue on Computational Logic in Multi-Agent Systems.

8. H. Chen, T. Finin, A. Joshi, F. Perich, D. Chakraborty, , and L. Kagal. Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing*, 19(5):69–79, November/December 2004.

9. H. Chen, F. Perich, T. Finin, and A. Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications. In *International Conference on Mobile and Ubiquitous Systems: Networking and Services*, Boston, MA, August 2004.

10. A. C. da Rocha Costa, J. F. Hübner, and R. H. Bordini. On entering an open society. In *XI Brazilian Symposium on Artificial Intelligence*, pages 535–546, Fortaleza, Oct. 1994. Brazilian Computing Society.

11. Y. Ding, D. Fensel, M. C. A. Klein, B. Omelayenko, and E. Schulten. The role of ontologies in ecommerce. In Staab and Studer [24], pages 593–616.

12. I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, second edition, 2003.

13. V. Haarslev and R. Moller. Description of the RACER system and its applications. In C. A. Goble, D. L. McGuinness, R. Möller, and P. F. Patel-Schneider, editors, *Proceedings of the International Workshop in Description Logics 2001 (DL'01)*, 2001.

14. I. Horrocks. FaCT and iFaCT. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. Patel-Schneider, editors, *Proceedings of the International Workshop on Description Logics (DL'99)*, pages 133–135, 1999.

15. I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in LNCS, pages 17–29. Springer, 2003.

16. D. Kinny. Agents – the challenge of relevance to the IT mainstream. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, editors, *Programming Multi-Agent Systems: Second International Workshop (ProMAS 2004), held with AAMAS-2004, 20th of July, New York City, NY*, number 3346 in LNAI. Springer-Verlag, Berlin, 2004.

17. D. L. McGuinness and F. van Harmelen, editors. *OWL Web Ontology Language overview. W3C Recommendation*. Available at http://www.w3.org/TR/owl-features/, February 2004.

18. S. E. Middleton, D. D. Roure, and N. R. Shadbolt. Ontology-based recommender systems. In Staab and Studer [24], pages 577–498.

19. 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 *Declarative Agent Languages and Technologies, Proceedings of the First International Workshop (DALT-03), held with AAMAS-03, 15 July, 2003, Melbourne, Australia*, number 2990 in LNAI, pages 135–154, Berlin, 2004. Springer-Verlag.

20. P. F. Patel-Schneider. DLP system description. In E. Franconi, G. D. Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors, *Proceedings of the International Workshop in Description Logics 1998 (DL'98)*, pages 133–135, 1998.

21. G. Plotkin. A structural approach to operational semantics, 1981. Technical Report, Department of Computer Science, Aarhus University.

22. A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. Van de Velde and J. Perram, editors, *Proceedings of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96), 22–25 January, Eindhoven, The Netherlands*, number 1038 in LNAI, pages 42–55, London, 1996. Springer-Verlag.

23. Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60:51–92, 1993.

24. S. Staab and R. Studer, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, 2004.

25. R. Stevens, C. Wroe, P. W. Lord, and C. A. Goble. Ontologies in bioinformatics. In Staab and Studer [24], pages 635–658.