Agent-Oriented Programming with Underlying Ontological Reasoning

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

[1] Universidade Federal do Rio Grande do Sul, Brazil
afmoreira@inf.ufrgs.br
[2] Universidade do Vale do Rio dos Sinos, Brazil
renata@exatas.unisinos.br
[3] University of Durham, UK
R.Bordini@durham.ac.uk
[4] Universidade Regional de Blumenau, Brazil
jomi@inf.furb.br

Abstract. Developing applications that make effective 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-based and ubiquitous computing. In this paper, we formally define a version of the BDI agent-oriented programming language AgentSpeak based on description logic rather than predicate logic. In this approach, the belief base of an agent contains the definition of complex concepts, besides specific factual knowledge. We illustrate the approach using examples based on the well-known smart meeting-room scenario. The advantages of combining AgentSpeak with description logics 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 given that (ontological) consistency of a belief addition can be checked; (iii) retrieving 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 OWL. Extending agent programming languages with description logics can have a significant impact on the development of multi-agent systems for the semantic web.

1 Introduction

Developing applications that make effective 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 technologies are also being used as the basis for other important trends in computer science such as grid computing [13] and ubiquitous computing [9].

Among the key components of semantic web technologies are *domain ontologies* [24], responsible for the specification of application-specific knowledge. As they can be expressed logically, they can be the basis for sound reasoning in a specific domain. Several ontologies are being developed for specific applications [10, 12, 18, 25]. Another key component of semantic web technologies are *intelligent agents*, which

M. Baldoni et al. (Eds.): DALT 2005, LNAI 3904, pp. 155–170, 2006.

should make use of the available knowledge, and interact with other agents autonomously, so as to act on the user's best interests.

In this work, we bring together these two key semantic web components 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, 2, 6]. The AgentSpeak variant proposed here is based on Description Logic (DL) [3] 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 information — 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 and its interpreter with underlying automated 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 the context of the semantic web. 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. Further, ontological reasoning combined with an agent programming language itself facilitates certain tasks involved in programming in such languages.

The remainder of this paper is organised as follows. In Section 2, we describe the syntax of the AgentSpeak language based on DL and we also explain briefly the main characteristics of the particular DL 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 its intuition and illustrating the practical benefits of the proposed modification. The examples used here are related to the well-known scenario of smart meeting-room applications. 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 [8], 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, besides factual knowledge.

In order to keep the formal treatment and examples simple, in this paper we assume $\mathcal{ALC}$ as the underlying description logic [4] for AgentSpeak-DL. An agent program *ag* is thus given by an ontology *Ont* and a set *ps* of plans, as defined by the grammar in Figure 1.

$$
\begin{array}{lll}
ag & ::= & Ont \quad ps \\
 & & \\
Ont & ::= & TBox \quad ABox \\
TBox & ::= & TBoxAx_1 \ldots TBoxAc_n \qquad (n \geq 0) \\
ABox & ::= & at_1 \ldots at_n \qquad (n \geq 0) \\
TBoxAx & ::= & D_1 \equiv D_2 \mid D_1 \sqsubseteq D_1 \\
D & ::= & \bot \mid A \mid \neg D \mid D_1 \sqcap D_2 \mid D_1 \sqcup D_2 \mid \forall R.D \mid \exists R.D \\
at & ::= & A(t) \mid R(t_1, t_2) \\
 & & \\
ps & ::= & p_1 \ldots p_n \qquad (n \geq 1) \\
p & ::= & te : ct \leftarrow h \\
te & ::= & +at \mid -at \mid +g \mid -g \\
ct & ::= & at \mid \neg at \mid ct \ \wedge \ ct \mid \mathsf{T} \\
h & ::= & a \mid g \mid u \mid h; h \mid \mathsf{T} \\
g & ::= & !at \mid ?at \\
u & ::= & +at \mid -at
\end{array}
$$

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. An ABox describes the state of an application domain by asserting that certain individuals are instances of certain classes or that certain individuals are related by a property.

In the grammar, metavariable D represent classes; $D_1 \equiv D_2$ asserts that both classes are equivalent, and $D_1 \sqsubseteq D_2$ asserts that class D_1 is subsumed by class D_2. The definition of classes assumes the existence of identifiers for primitive classes and properties. The metavariable A stands for names of primitive classes (i.e., predefined classes) as well as atomic class names chosen to identify constructed classes defined in the TBox; the metavariable R stands for primitive properties. New classes can be defined by using certain constructors, such as $\sqcup$ and $\sqcap$, for instance ($\sqcap$ and $\sqcup$ represent the intersection and the union of two concepts, respectively). Metavariable t is used for (first-order) terms.

An AgentSpeak 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* for handling an event. The remainder of the plan, the plan *body*, is a sequence of basic actions to be executed, (sub)goals that the agent has to achieve (or test), or (internal) belief updates. The *basic actions* (represented by the metavariable a in the grammar above) that can appear in a plan body denote the pre-defined ways in which an agent is able to change its environment.

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 (i.e., beliefs or goals).

In every reasoning cycle, one of the existing events that have not been dealt with yet is selected for being handled next. If there is an applicable plan for it (i.e., a relevant plan whose context is satisfied), an instance of that plan becomes an "intended means". The agent is then committed to execute that plan, as part of one of its intentions. Also within a reasoning cycle, one of the (possibly various) intentions of the agent is selected for being further executed.

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 [9]. An example of TBox components for such scenario is as follows:

$$\begin{aligned} \mathit{presenter} &\equiv \mathit{invitedSpeaker} \sqcup \mathit{paperPresenter} \\ \mathit{attendee} &\equiv \mathit{person} \sqcap \mathit{registered} \sqcap \neg \mathit{presenter} \ldots \end{aligned}$$

This TBox asserts that the concept *presenter* is equivalent to *invited speaker or paper presenter*, and the concept *attendee* is equivalent to the concept of a *registered person who is not a presenter*. Examples of elements of an ABox defined with respect to the TBox above are:

$$\begin{aligned} &\mathit{invitedSpeaker}(\mathit{john}) \\ &\mathit{paperPresenter}(\mathit{mary}) \end{aligned}$$

We assume that, as usual, the schedule of an academic event has slots for paper presenters and for invited speakers. At the end of a presentation slot, an event is generated indicating the next presenter, according to the schedule, is now required for the continuation of the session. For example, at the time Mary is required to start her presentation, a meeting-room agent would acquire (say, by communication with the schedule agent which has sensors for detecting when a speaker leaves the

```
+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

stage) the belief `paperPresenter(mary)`, which generates an event for handling `+paperPresenter(mary)`, a belief addition event. In Figure 2, we give examples of AgentSpeak-DL plans to handle such "next presenter" events.

The first plan in Figure 2 says that if a presenter of a paper is late s/he is rescheduled to the end of the session (and the session continues with the next scheduled speaker). If an invited speaker is late, apologies are given to the audience and the speaker is announced (this event only happens when the invited speaker actually arrives, assuming the paper sessions must not begin before the invited talk). The third is a general plan that announces any presenter (`paperPresenter` or `invitedSpeaker`) if s/he is not late.

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

3 Semantics of AgentSpeak-DL

The reasoning cycle of an AgentSpeak agent follows a sequence 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 {ProcMsg, SelEv, RelPl, ApplPl, SelAppl, AddIM, SelInt, ExecInt, ClrInt}; they stand for, respectively: processing communication messages, 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 selected 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. However, we concentrate here on the formalisation of the steps that required changes to accommodate the DL extensions. Operational semantics [21] is a widely used method for giving semantics to programming languages and studying their properties.

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:

– *plan retrieval and selection*: 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);

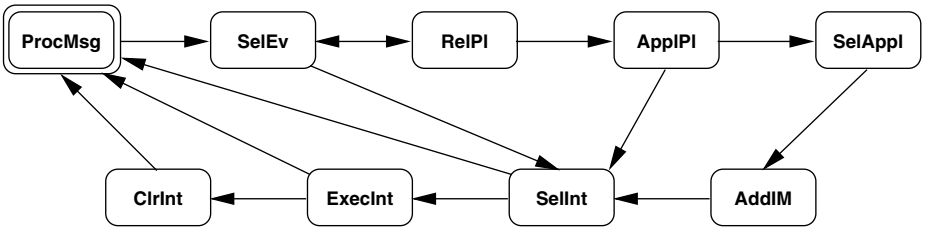

Fig. 3. Transitions between Reasoning Cycle Steps

– *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 semantics for AgentSpeak is given in [8]. 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 (the particular intention that generated an internal event, or the empty intention $\top$ in case of an external event).
  • A is a set of *actions* that the agent has chosen for execution; the agent's effector will change the environment accordingly.
– It helps in giving the semantics to use a structure T which keeps track of temporary information that is required in subsequent steps of the reasoning cycle; however, such information is only used within a single reasoning cycle. T is used to denote a tuple $\langle R, Ap, \iota, \varepsilon, \rho\rangle$ with the required 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 subset of 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 a reasoning cycle.
– The current step s within an agent's reasoning cycle is annotated by labels $s \in \{\mathsf{SelEv}, \mathsf{RelPl}, \mathsf{ApplPl}, \mathsf{SelAppl}, \mathsf{AddIM}, \mathsf{SelInt}, \mathsf{ExecInt}, \mathsf{ClrInt}\}$ (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 notation:

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

3.2 Retrieving and Selecting Plans 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 of predicate-logic AgentSpeak, this is formalised by the following rules.

Rule $\mathbf{Rel}_1$ below initialises the R component of T with the set of relevant plans determined by the auxiliary function RelPlans (which is formally defined below), and sets the reasoning cycle to the step (ApplPl) that determines the applicable plans among those in the R component.

$$\frac{T_\varepsilon = \langle te, i\rangle \qquad \mathsf{RelPlans}(ag_{ps}, te) \neq \{\}}{\langle ag, C, T, \mathsf{RelPl}\rangle \longrightarrow \langle ag, C, T', \mathsf{ApplPl}\rangle} \quad (\mathbf{Rel_1})$$

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

$$\frac{T_\varepsilon = \langle te, i\rangle \qquad \mathsf{RelPlans}(ag_{ps}, te) = \{\}}{\langle ag, C, T, \mathsf{RelPl}\rangle \longrightarrow \langle ag, C, T, \mathsf{SelEv}\rangle} \quad (\mathbf{Rel_2})$$

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 another event from the set of events (rule $\mathbf{Rel}_2$). If there are no pending events to handle, the cycle skips to the intention execution step.

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

The auxiliary function 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 selected event $\langle te, i\rangle$, the set $\mathit{RelPlans}(ps, te)$ of relevant plans for that event is defined as follows:*

$$\mathit{RelPlans}(ps, te) = \{(p, \theta) \mid p \in ps \textit{ and}$$

$$\theta \textit{ is a mgu s.t. } te\theta = \mathit{TrEv}(p)\theta\}.$$

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 specific plans 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 added (instead of using 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 the plan's triggering event has a more general relevance, in the sense that it subsumes the actual event. In practice, this is checked by: (i) finding a plan whose triggering event predicate is related (in the ontology) by subsumption or equivalence relations to the predicate in the event 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 $\mathbf{Rel}_1$ and $\mathbf{Rel}_2$ still apply, but the auxiliary function RelPlans for AgentSpeak-DL has to be redefined as follows (recall that D is a metavariable for classes of an ontology):

Definition 2. *Given plans ps and ontology Ont of an agent, and an event* $\langle *A_1(t), i\rangle$ *where* $* \in \{+,-,+!,+?,-!,-?\}$*, the set* $\mathsf{RelPlans}(Ont, ps, *A_1(t))$ *is a set of pairs* (p,θ) *such that* $p \in ps$*, with* $\mathsf{TrEv}(p) = *' A_2(t')$*, such that*

- $* = *'$
- $\theta = \mathsf{mgu}(t, t')$
- *Ont* $\models A_1 \sqsubseteq A_2$ *or Ont* $\models A_1 \equiv A_2$

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. Suppose that the application detects that the next slot is allocated to the invited speaker *john*. This causes the addition of the external event $\langle +\texttt{invitedSpeaker}(\texttt{john}), \top\rangle$ to the set of events. Recall that *invitedSpeaker* $\sqsubseteq$ *presenter* can be inferred from the ontology. With this, and given Definition 2, the plan with triggering event $+\texttt{presenter}(\texttt{X})$ is also considered relevant for dealing with that event (see Figure 2). Observe that using subsumption instead of unification alone as the mechanism for selecting relevant plans results in a potentially 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 $\mathbf{Appl}_1$ and $\mathbf{Appl}_2$ formalise the step of the reasoning cycle that determines the applicable plans from the set of relevant plans.

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

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

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

Rule $\mathbf{Appl}_1$ initialises the T_{Ap} component with the set of applicable plans; $\mathbf{Appl}_2$ 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 depend on the auxiliary function ApplPlans, which for predicate-logic AgentSpeak has been defined as follows (note that *bs* was originally the agent's set of beliefs, which now has been replaced by an ontology).

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

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

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

Observe that the context of a plan is a conjunction of literals and, as the belief base is formed by a set of *ground atomic formulæ* only, the problem of checking if the plan's context is a logical consequence of the belief base reduces to the problem of checking membership (or otherwise, for default negation) of context literals to the set of beliefs while finding an appropriate unifier.

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; a literal l is either $A(t)$ or $\neg A(t)$. 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 ones are applicable is then formalised below. Again, because the belief base is structured and that reasoning is based on ontological knowledge rather than just straightforward variable instantiation, the resulting set of applicable plans might be larger than in predicate-logic AgentSpeak.

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

$$\mathsf{AppPlans}(Ont, R) = \{(p, \theta' \circ \theta) \mid (p, \theta) \in R \text{ and}$$

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

More than one plan can be considered applicable for handling an event at a given moment in time. 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 selected plan is then assigned to the T_ρ component of the configuration indicating, for the next steps in the reasoning cycle, that an instance of this plan has to be added to the agent's intentions.

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

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

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 $+\texttt{presenter(P)}$ and $+\texttt{invitedSpeaker(P)}$ are 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 (see rule **SelApplOnt** below). In the example we are using, the selected plan should be the one

with triggering event +`invitedSpeaker` as probably this plan has been written to deal more specifically with the case of invited speakers, 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 speakers that we have in the example is not applicable (because s/he is not late), instead of the agent not acting at all for lack of applicable plans, the more general plan for presenters can then be used.

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

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 being handled in a particular reasoning cycle is external, a new intention is created and its single plan is the plan p assigned to the ρ component in the previous steps of the reasoning cycle. If the event is internal, rule **IntEv** (omitted here) states that the plan in ρ should be pushed on top of the intention associated with the given event. Rule **ExtEv** creates a new intention (i.e., a new *focus of attention*) for an external event.

3.3 Intention Execution: Querying the Belief Base

The next step, after updating the set of intentions as explained above, uses an agent-specific function (S_I) that selects the intention to be executed next (recall that an intention is a stack of plans). 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 (by adding or removing "internal beliefs", as opposed to those resulting from perception of the environment).

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, from the point of view of the AgentSpeak interpreter, reduces to instructing other architectural components to perform the respective action so as to change the environment. The execution of achievement goals adds a new internal event to the set of events. That event will be selected at a later reasoning cycle, and handled as explained above.

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 Test auxiliary function defined below returns a set of most general unifiers, all of which make 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* $\mathit{Test}(bs, at)$ *produced by testing at against bs is defined as follows:*

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

This auxiliary function is then used in the formal semantics by 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, in recent extensions of AgentSpeak it may be the case that the test goal is used as a triggering event of a plan, which is used by programmers to formulate more sophisticated queries[1]. 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).

$$\frac{T_\iota = i[head \leftarrow ?at\,;h] \qquad \mathsf{Test}(ag_{bs}, at) \neq \{\}}{\langle ag, C, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag, C, T, ClrInt\rangle} \qquad (\mathbf{Test_1})$$

$$\begin{array}{l} where:\ C'_I = (C_I \setminus \{T_\iota\}) \cup \{(i[head \leftarrow h])\theta\} \\ \qquad\qquad \theta \in \mathsf{Test}(ag_{bs}, at) \end{array}$$

$$\frac{T_\iota = i[head \leftarrow ?at\,;h] \qquad \mathsf{Test}(ag_{bs}, at) = \{\}}{\langle ag, C, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag, C, T, \mathsf{ClrInt}\rangle} \qquad (\mathbf{Test_2})$$

$$\begin{array}{l} where:\ C'_E = C_E \cup \{\langle +?at, i[head \leftarrow h]\rangle\} \\ \qquad\qquad C'_I = C_I \setminus \{T_\iota\} \end{array}$$

In AgentSpeak-DL, the semantic rules for the evaluation of a test goal $?A(t)$ are exactly as the rules $\mathbf{Test}_1$ and $\mathbf{Test}_2$ above. However, the function Test used to check whether the formula $A(t)$ is a logical consequence of the agent's belief base, which is now based on an ontology, needs to be changed. 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*

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

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 description logic allows agents to infer knowledge that is implicit in the ontology. As an example, suppose that the agent's 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 to *mary*.

3.4 Belief Updating

In predicate-logic AgentSpeak, the addition or deletion of internal beliefs has no further implications apart from a possible new event to be included in the set of events E (as for

[1] 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).

belief update from perception of the environment, events are generated whenever beliefs are added or removed from the belief base). 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 accordingly. In practice, this mechanism for adding and removing internal beliefs allow agents to have "mental notes", which can be useful at times (for practical programming tasks). These beliefs should not normally be confused with beliefs acquired from perception of the environment.

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

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

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

In predicate-logic AgentSpeak, the belief base consists solely of ground atomic formulæ so ensuring consistency is not a major task. In AgentSpeak-DL, however, belief update is more complicated than in predicate-logic AgentSpeak. The agent's ABox contains class assertions $A(t)$ and property assertions $R(t_1, t_2)$. The representation of such information should, of course, be consistent with its TBox.

In AgentSpeak-DL, the addition of assertions to the agent's ABox is only allowed if the ABox resulting of such addition is consistent with the TBox (i.e., if adding the given belief to the ontology's ABox[2] maitains consistency). Approaches for checking consistency of an ABox with respect to a TBox are discussed in detail in [4]. The semantic rule **AddBel** has to be modified accordingly:

$$\frac{T_\iota = i[head \leftarrow +b\,;h] \qquad ag_{Ont} \cup \{b\} \textit{ is consistent}}{\langle ag, C, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag', C', T, \mathsf{ClrInt}\rangle} \qquad \textbf{(AddBelOnt1)}$$

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

There is a similar rule for belief deletions (i.e., a formula such as $-at$ in a plan body), but it is trivially defined based on the one above, so we do not include it explicitly here.

Expanding on the smart meeting-room example, assume that the TBox is such that the concepts *chair* and *bestPaperWinner* are disjoint. Clearly, if the ABox asserts that $chair(mary)$, the assertion $bestPaperWinner(mary)$ cannot simply be added to it, as the resulting belief base would become inconsistent. The rule bellow formalises the semantics of AgentSpeak-DL for such cases.

[2] In the **AddBelOnt** rules below, we assume that the union operation applied to an ontology $\langle TBox_1, ABox_1\rangle$ and an ABox $ABox_2$ would result in an ontology $\langle TBox_1, ABox_1 \cup ABox_2\rangle$, as expected.

$$\frac{\begin{array}{c} T_\iota = i[head \leftarrow +b\,;h] \\ ag_{Ont} \cup \{b\} \textit{ is not consistent} \\ (adds, dels) = \mathsf{BRF}(ag_{Ont}, b) \end{array}}{\langle ag, C, T, \mathsf{ExecInt}\rangle \longrightarrow \langle ag', C', T, \mathsf{ClrInt}\rangle} \quad \textbf{(AddBelOnt2)}$$

$$\begin{array}{rl} \textit{where: } ag'_{Ont} & = (ag_{Ont} \cup adds) \setminus dels \\ C'_E & = C_E \cup \{\langle +b', \mathsf{T}\rangle \mid b' \in adds\} \\ & \quad \cup \{\langle -b'', \mathsf{T}\rangle \mid b'' \in dels\} \\ C'_I & = (C_I \setminus \{T_\iota\}) \cup \{i[head \leftarrow h]\} \end{array}$$

According to this rule, a belief revision function (BRF) is responsible for determining the necessary modifications in the belief base as a consequence of attempting to add a belief that would cause ontological inconsistency. Given an ontology and a belief atom, this function returns a pair of sets of atomic formulæ that were, respectively, added (*adds*) and deleted (*dels*) to/from the belief base. Belief revision is a complex subject and clearly outside the scope of this paper; see [1] for an interesting (tractable) approach to belief revision.

The rules above are specific to the 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. However, the same BRF function is used when beliefs need to be added as a consequence of perception of the environment. In the general case, whenever a belief needs to be added to the belief base, a belief revision function should be able to determine whether the requested belief addition will take place and whether any belief deletions are necessary so that the addition can be carried out and consistency maintained. Note that the precise information on resulting additions and deletions of beliefs are needed by the AgentSpeak interpreter so that the necessary (external) events are generated, hence the signature of the BRF function as defined above, and the relevant events added to C_E in rule **AddBelOnt2**.

The reasoning cycle finishes by removing from the set of intentions an intended means or a whole intention that has been executed to completion. There are no changes in those rules of the original semantics as a consequence of the extensions presented in this paper.

4 Conclusions and Future Work

This paper has formalised the changes in the semantics of AgentSpeak that were required for combining agent-oriented programming with ontological reasoning. The main improvements to AgentSpeak resulting from the variant based on a description logic are: (i) queries to the belief base are more expressive as their results do not depend only on explicit knowledge but can be inferred from the ontology; (ii) the notion of belief update is refined so that a property about an individual can only be added if the resulting ontology-based belief base would preserve consistency (i.e., if the ABox assertion is consistent with the concept descriptions in the TBox); (iii) retrieving a plan (from 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 rela-

tion between concepts; and (iv) agents may share knowledge by using web ontology languages such as OWL.

With this paper, we hope to have contributed towards showing that extending an agent programming language with the descriptive and reasoning power of description logics can have a significant impact in 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 of major interest such as grid and ubiquitous computing. It also creates perspectives for elaborate forms of agent migration, where plans carefully written to use (local) ontological descriptions can ease the process of agent adaptation to different societies [11].

Clearly, the advantages of the more sophisticated reasoning that is possible using ontologies also represent an increase in the computational cost of running agent programs. The trade-off between increased expressive power and possible decrease in computational efficiency in the context of this work has not been considered as yet, and remains future work.

Also as future work, we aim at incorporating other ongoing activities related to agent-oriented programming and semantic web technologies into AgentSpeak-DL. 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 underlying OWL Lite and OWL DL [16]. 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 called ***Jason*** [7] is available *open source* (`http://jason.sourceforge.net`). An AgentSpeak-DL interpreter is currently being implemented on top of ***Jason***, 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 [8], 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 be done by existing software such as those described in [14, 20, 15]. We are currently considering the use of RACER [14] in particular for extending ***Jason*** so that belief bases can refer to OWL ontologies.

Another of our planned future work is the integration of AgentSpeak-DL with the AgentSpeak extension presented in [19], which 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. Another relevant feature 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 belief refers. For example, the fact that an agent `ag1` has informed another agent `ag2` about a belief `p(t)` as defined in a given ontology `ont` can be expressed in `ag2`'s belief base as `p(t)[source(ag1),ontology("http://.../ont")]`.

An interesting issue associated with ontologies is that of how different ontologies for the same domain can be integrated. Recent work reported in [5] 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.

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

Acknowledgements

Rafael Bordini gratefully acknowledges the support of The Nuffield Foundation (grant number NAL/01065/G).

References

1. N. Alechina, M. Jago, and B. Logan. Resource-bounded Belief Revision and Contraction. In *this volume*.
2. 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.
3. F. Baader, D. Calvanese, D. N. D. McGuinness, and P. Patel-Schneider, editors. *Handbook of Description Logics*. Cambridge University Press, Cambridge, 2003.
4. 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.
5. 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.
6. 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.
7. 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/`.
8. 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.
9. 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.

10. 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.
11. 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.
12. 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.
13. I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, second edition, 2003.
14. 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.
15. 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.
16. 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.
17. D. L. McGuinness and F. van Harmelen, editors. *OWL Web Ontology Language overview. W3C Recommendation*. Avalilable 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.