CooL-AgentSpeak: Enhancing AgentSpeak-DL Agents with Plan Exchange and Ontology Services

Viviana Mascardi*, Davide Ancona*, Rafael H. Bordini† and Alessandro Ricci‡

*DISI - University of Genova
Via Dodecaneso 35, 16146, Genova, Italy
Email: Viviana.Mascardi@unige.it, Davide.Ancona@unige.it

† INF - Federal Univ. of Rio Grande do Sul
PO Box 15064, 91501-970 Porto Alegre RS, Brazil
Email: R.Bordini@inf.ufrgs.br

‡ DEIS - University of Bologna
Via Venezia 52, 47023, Cesena (FC), Italy
Email: a.ricci@unibo.it

***Abstract*—In this paper we present CooL-AgentSpeak, an extension of AgentSpeak-DL with plan exchange and ontology services. In CooL-AgentSpeak, the search for a plan is no longer limited to the agent's local plan library but is carried out in the other agents' libraries too, according to a cooperation strategy, and it is not based solely on unification and on the subsumption relation between concepts, but also on ontology matching. Belief querying and updating take advantage of ontological reasoning and matching as well.**

***Keywords*-AgentSpeak, cooperation, plan exchange, ontology matching**

I. INTRODUCTION

Cooperation is important in the context of multi-agent systems to allow agents to help others achieve their goals. In our past research [1] we devised some scenarios where cooperation obtained by allowing BDI agents to exchange their plans would have turned out to be extremely useful. We named that extension to the standard BDI approach Coo-BDI. One scenario where Coo-BDI features could demonstrate their potential is that of digital butlers. Quoting [1],

A "digital butler" is an agent that assists the user in some task such as managing her/his agenda, filtering incoming mail, retrieving interesting information from the web. A typical feature of a digital butler is its ability to dynamically adapt its behaviour to the user needs. This ability is achieved by cooperating both with more experienced digital butlers and with the assisted user.

Let us consider now the above scenario, where a digital butler a needs to manage the event *+invitee(john, tomorrow)* that its human user generated by means of the user interface. Let us suppose that a does not know how to deal with the presence of an invitee (namely, it has no relevant plans for that event) and asks the more experienced digital butler b. Agent b has a nice plan triggered by event *+visitor(Who, When)* that states how to make a guest feel as comfortable as possible by offering them all the hospitality that they deserve. Unfortunately, *+invitee(john, tomorrow)* and *+visitor(Who, When)* do not unify, and b will not send its nice plan to a for not realising it is relevant.

Now, let us suppose that the belief base of agents is not just a set of atoms, but it consists of the definition of complex concepts and relationships among them, as well as specific factual knowledge (or beliefs, in this case), namely, in Description Logic terminology, in a TBox and an ABox. With this assumption applied to the AgentSpeak language we would obtain the AgentSpeak-DL language as reported in [2].

If both a and b referred to the same ontology $\mathtt{ont}(oid)$, and if we could demonstrate that $\mathtt{ont}(oid) \models invitee \sqsubseteq visitor$, we could solve a's problem: $+invitee(john, tomorrow)$ and $+visitor(Who, When)$ do not unify, but a plan that works for a visitor should work for an invitee as well, since the latter is a subconcept of the former according to ontology $\mathtt{ont}(oid)$.

However, let us consider an even more complex scenario, where a and b do not refer to the same ontology (they refer to $\mathtt{ont}(a)$ and $\mathtt{ont}(b)$ respectively), and where b's plan is triggered by *+guest(Who, When)*. Even if we combined the features of AgentSpeak-DL and of Coo-BDI, we could not manage this situation properly. In fact, what a and b would need here, is some "cross ontological" unification of concept allowing b to realise that $guest \in \mathtt{ont}(b)$ is equivalent (at least up to a certain degree) to $invitee \in \mathtt{ont}(a)$. In this case, b could send its plan for dealing with guests to a, and a could use it for dealing with the invitee.

In this paper we discuss the integration of Coo-BDI and AgentSpeak-DL, and the enhancement of the resulting language with *ontology matching capabilities* to deal with situations such as the one above. In the CooL-AgentSpeak language we introduce in this paper, the search for a plan takes place not only in the agent's local plan library but also in the other agents' libraries, according to the cooperation strategy as in Coo-BDI. However, handling an event is more flexible as it is not based solely on unification and on the subsumption relation between concepts as in AgentSpeak-DL, but also on ontology matching. Belief querying and updating take advantage of ontological matching as well.

The paper is organised in the following way: Section II provides background knowledge on AgentSpeak-DL, Coo-BDI, ontology services in MAS, and ontology matching techniques; Section III introduces the CooL-AgentSpeak language and Section IV outlines its semantics in an informal way. Section V sketches the implementation of CooL-AgentSpeak in *Jason*. Finally, Section VI provides an overview of the related work and concludes by discussing future research directions.

II. Background

A. AgentSpeak-DL and its JASDL Implementation

Many extensions of AgentSpeak have appeared over the years. In [3], for example, Bordini and colleagues defined the formal semantics of AgentSpeak agents to be able to process speech-act based messages, which is fundamental to allow social behaviour in BDI agents. In agent communication, the assumption that ontologies are used to ensure interoperability has been made since the very beginning of the work on ontologies, before they made the basis for the Semantic Web effort. However, what was not considered before the work in [2] is that ontological reasoning can facilitate the development of agent programs written in agent-oriented programming languages.

That paper introduced AgentSpeak-DL, a variant of the AgentSpeak logic-based BDI-inspired agent-oriented programming language. The paper proposed a formal semantics for AgentSpeak-DL, a variant of AgentSpeak based on description logic. In that theoretical proposal, the belief base contained a TBox and an ABox, so all predicates used in an agent program were effectively part of an ontology. With this, queries to the belief base could use ontological reasoning in order to answer the query; belief update was able to ensure ontological consistency of the belief base; triggering plan execution could also be based on subsumption of the event and the plan's trigger; and of course this pointed to future practical work where agents could share knowledge represented in OWL ontologies, for example.

Exactly to allow the practical use of these ideas, extensive work was carried out by Klapiscak and Bordini [4]. That paper introduced JASDL, an extension of the *Jason* AgentSpeak interpreter making available all features of AgentSpeak-DL and others, including belief revision. Most importantly, the development of JASDL used *Jason* extensibility mechanism rather than altering the hardwired implementation of the operational semantics. In JASDL, belief annotations are used to point out which predicates are defined externally in OWL ontologies available on the web; this means that traditional AgentSpeak code can be used together with AgentSpeak-DL code. OWL-API was used to allow the integration with ontological reasoners which would make the semantics available elsewhere (in OWL ontologies on the web) usable in an agent program, so as to allow, for example, more compact programs that can handle various subsumed events by a single, more general, plan (when appropriate).

B. Coo-BDI and its Coo-AgentSpeak Implementation

Coo-BDI (Cooperative BDI [1]) extends traditional BDI agent-oriented programming languages in many respects. As in the traditional BDI setting, Coo-BDI agents are characterised by an event queue, a mailbox, a plan library, a belief base, and a set of intentions. The main extensions of Coo-BDI involve the introduction of *cooperation* among agents for the retrieval of external plans for a given triggering event; the extension of *plans* with "access specifiers"; the extension of *intentions* to take into account the external plan retrieval mechanism; and the modification of the *Coo-BDI engine* (i.e., the interpreter) to cope with all these issues.

The cooperation strategy of an agent includes the set of agents with which it is expected to cooperate, the plan retrieval policy and the plan acquisition policy. The cooperation strategy may evolve over time, allowing maximum flexibility and autonomy for the agents. Four predicates specify an agent's current *cooperation strategy*:

- `trustedAgents`(*TrustedAgents*) specifying the set of identifiers of the agents currently trusted by the agent;
- `retrievalPolicy`(*Retrieval*) specifying the current retrieval policy (*always* if external relevant plans should be always looked for, *noLocal* if they should be looked for only when no local relevant plans can be found);
- `acquisitionPolicy`(*Acquisition*) specifying the current plan acquisition policy (*discard* when the retrieved plan must be used and then discarded, *add* when it must be added to the local plan library, *replace* when it must replace existing relevant local plans);
- `timeout`(*Nat*), where Nat is a natural number, stating the number of milliseconds the agent will wait for a cooperative plan exchange request to be answered.

A plan *access specifier* determines the set of agents that the plan can be shared with, and the source of that plan. It may assume three values: *private* (the plan cannot be shared), *public* (the plan can be shared with any agent) and *only(TrustedAgents)* (the plan can be shared only with the agents contained in the *TrustedAgents* set).

Coo-BDI has been applied to (predicate logic) AgentSpeak (i.e., AgentSpeak without ontological reasonig), and made practical using the *Jason* interpreter [5].

C. Ontology Services in MAS

The problem of semantic mediation at the vocabulary and domain of discourse levels was tackled by the "Ontology Service Specification" issued by FIPA in 2001, http://www.fipa.org/specs/fipa00086/. According to that specification, an "Ontology Agent" (OA, for short) should be integrated into a MAS in order to provide services such as translating expressions between different ontologies and/or different content languages and answering queries about relationships between terms or between ontologies. Although the FIPA Ontology Service Specification represents an important attempt to analyse in a systematic way the services that an OA should provide for ensuring semantic interoperability in an open MAS, it has many limitation including the model to which ontologies should adhere (OKBC, http://www.ai.sri.com/~okbc/, when the most widely used language for representing ontologies today is OWL [6]) and the fact that agents are allowed to specify only one ontology as reference vocabulary for any given message.

Perhaps due to these limitations, there have been very few attempts to design and implement OAs. The only two attempts of integrating a FIPA-compliant OA into JADE, that we are aware of, are [7] and [8]. Both follow the FIPA specification but adapt it to ontologies represented in OWL.

An extension of the OA, described in [8], with services for ontology access, navigation, querying, modification, and versioning of modified ontologies has been recently carried out by F. Mulattieri [9]. Its integration in the *Jason* platform, carried out by exploiting the *Jason*–JADE interface, has been implemented by M. Barbieri and is briefly described in [9].

D. Ontology Matching

According to [10], a correspondence between an entity e belonging to ontology o and an entity e' belonging to ontology o' is a 5-tuple $< id, e, e', R, conf >$ where:

- id is a unique identifier of the correspondence;
- e and e' are the entities (e.g., properties, classes, individuals) of o and o' respectively;
- R is a relation, such as "equivalence", "more general", "disjointness", "overlapping", holding between the entities e and e';
- $conf$ is a confidence measure (typically in the $[0, 1]$ range) holding for the correspondence between the entities e and e'.

An alignment of ontologies o and o' is a set of correspondences between entities of o and o'.

Finally, a matching process can be seen as a function f which takes two ontologies o and o', a set of parameters p and a set of oracles and resources r, and returns an alignment A between o and o'.

III. The CooL-AgentSpeak Language

CooL-AgentSpeak stands for "Cooperative Description-Logic AgentSpeak". The syntax of the language is summarised in Figure 1. With respect to previous work on AgentSpeak-DL, the definition of a matching strategy ms is a completely new feature of CooL-AgentSpeak, as well as the annotation of ontological knowledge with sources and ontology identifiers.

ag	::=	Ont ps cs ms
Ont	::=	ABox TBox
ABox	::=	at_1 ... at_n $(n \geq 0)$
TBox	::=	$C_1 \equiv D_1$ ... $C_n \equiv D_n$ $(n \geq 0)$ $\mid$
		$C_1 \sqsubseteq D_1$ ... $C_n \sqsubseteq D_n$ $(n \geq 0)$ $\mid$
		$R_1 \equiv S_1$ ... $R_n \equiv S_n$ $(n \geq 0)$ $\mid$
		$R_1 \sqsubseteq S_1$ ... $R_n \sqsubseteq S_n$ $(n \geq 0)$
C, D	::=	$A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \forall R.C \mid \exists R.C$
R, S	::=	$P \mid R \sqcap S \mid R \sqcup S$
at	::=	C(t)[`ont`(oid), `src`(bsrc)] $\mid$
		R(t_1, t_2)[`ont`(oid), `src`(bsrc)]
bsrc	::=	`self` $\mid$ aid_1, ..., aid_n $\mid$ `percept`
oid	::=	*a string identifying an ontology* $\mid$ `self`
aid	::=	*a string identifying an agent*
ps	::=	p_1 ... p_n $(n \geq 1)$
p	::=	as te : ct $\leftarrow$ h [`src`(psrc)]
psrc	::=	`self` $\mid$ aid_1, ..., aid_n
as	::=	`private` $\mid$ `public` $\mid$ `only`(aid_1, ..., aid_n)
te	::=	+at $\mid$ -at $\mid$ +g $\mid$ -g
ct	::=	at $\mid$ $\neg$ at $\mid$ ct $\wedge$ ct $\mid$ `true`
h	::=	h_1; `true` $\mid$ `true`
h_1	::=	a $\mid$ g $\mid$ u $\mid$ h_1; h_1
g	::=	!at $\mid$?at
u	::=	+at $\mid$ -at
cs	::=	`trustedAgents`(aid_1, ..., aid_n)
		`retrievalPolicy`(rp)
		`acquisitionPolicy`(ap)
		`timeout`(Nat) $(Nat \in \mathbb{N})$
rp	::=	`always` $\mid$ `noLocal`
ap	::=	`discard` $\mid$ `add` $\mid$ `replace`
ms	::=	`match`(F) `parameters`(Par) `resources`(Res)
		`threshold`(Th) $(Th \in [0, 1] \bigcup \{\infty\})$

Figure 1. CooL-AgentSpeak: Syntax

Ontological knowledge. Following [2], we assume $\mathcal{ALC}$ as the underlying description logic [11] for representing the cognitive structures of CooL-AgentSpeak agents. The definition of classes and properties belonging to the ABox of the ontology assumes the existence of identifiers for primitive (i.e., not defined) classes and properties (metavariables A and P respectively). New classes and properties can be defined by using certain constructs, such as $\sqcap$ and $\sqcup$ that represent the intersection and the union of two entities, respectively. The TBox is a set of axioms establishing equivalence and subsumption relationships between classes and between properties. With respect to [11] and [2], we extended the syntax of the language used for representing the ontology by introducing annotations of concepts and properties, in order to make CooL-AgentSpeak practical as discussed below. Annotations are ignored during ontological reasoning and matching, hence they do not change the $\mathcal{ALC}$ semantics.

An agent belief is an atom belonging to the ABox annotated with `ont`(oid), where *oid* is the identifier of the ontology (*oid*=*self* if the belief is a naive one [4], i.e., a normal AgentSpeak belief that does not relate to an ontology). Along the lines of [3], beliefs are also annotated with sources `src`(*bsrc*), where *bsrc* can be either an agent identifier *aid* specifying the agent which previously communicated that information, or *self* to denote beliefs created by the agent itself, or *percept* to indicate that the belief was acquired through perception of the environment.

Matching functions. *F* is a metavariable representing a matching function name. We assume that matching functions can be unequivocally identified by means of their name. *Par* and *Res* are metavariables representing the parameters and resources needed by matching function *F*.

Agent. An agent *ag* is characterised by an ontology *Ont*, a plan library *ps*, a cooperation strategy *cs*, and an ontology matching strategy *ms*.

Plan library. The plan library consists of a set of CooL-AgentSpeak plans. Like predicate-logic AgentSpeak plans, CooL-AgentSpeak plans consist of an access modifier *as* (defining the accessibility level for the plan, as introduced in Section II-B), a trigger *te*, a context *ct*, a body *h*, and an optional list of sources `src`(*psrc*) which specify the agents from which the plan has been obtained (*self* if the plan belongs to the agent itself).

Cooperation strategy. The cooperation strategy *cs* follows the description given in Section II-B and is defined through the predicates `trustedAgents`, `retrievalPolicy`, `acquisitionPolicy`, and `timeout`.

Ontology matching strategy. The ontology matching strategy *ms* is a characteristic feature of CooL-AgentSpeak. It follows the description given in Section II-D and consists of:

- `match`(*F*) specifying which matching function *F* the agent uses to perform the match;
- the arguments `parameters`(*P*) and `resources`(*Res*) that will be passed on to the matching function, besides the two ontologies to match;
- `threshold`(*Th*) with $Th \in [0,1] \cup \{\infty\}$ stating the confidence threshold below which correspondences returned by the matching function will be discarded; if the threshold is set to ∞, all the correspondences will be discarded.

The ontology matching strategy may evolve as well, thus allowing an agent to use different matching functions and different parameters throughout its execution.

IV. CooL-AgentSpeak Informal Semantics

In this section we discuss the main extensions and changes we made to the language semantics in order to take plan exchange and ontology matching services into account.

The most relevant steps of an AgentSpeak reasoning cycle are the following: processing received messages (`ProcMsg`); selecting an event from the set of events (`SelEv`); retrieving all relevant plans (`RelPl`); checking which of those are applicable (`ApplPl`); selecting one particular applicable plan (the intended means) (`SelAppl`); adding the new intended means to the set of intentions (`AddIM`); selecting an intention (`SelInt`); executing the selected intention (`ExecInt`), and clearing an intention or intended means that may have finished in the previous step (`ClrInt`).

The semantic rules for these steps are essentially the same in CooL-AgentSpeak as in predicate-logic AgentSpeak [12] and in AgentSpeak-DL [2], with the exception of the following aspects that are affected by the introduction of ontology matching and plan exchange:

- *plan search*: performed in the steps responsible for collecting relevant and applicable plans, and selecting one plan among the set of applicable plans (`RelPl`, `ApplPl`, and `SelAppl`);
- *querying the belief base*: performed in the step devoted to executing the selected intention, `ExecInt`; and
- *belief updating*: performed in steps `ExecInt` and `ProcMsg` (e.g., in processing messages with performative tell from other agents). It also happens in perception of the environment that takes place before `ProcMsg`. Percepts can be annotated with a reference to an ontology as well.

Remember that, in CooL-AgentSpeak, both atoms (and hence beliefs, goals, triggering events, etc.) and plans are annotated with their *sources*.

In the setting we are going to present, plan (resp. belief) retrieval and update do not depend on plan (resp. belief) sources so we drop them for readability[1]. In order to take them into consideration properly, we would need to introduce some more sophisticated policies depending on sources too.

A scenario where plan sources would make the difference in the plan search stage might be, for example, the one where external plans are accepted only if they come from trusted sources. However, if we assume that, at the MAS initialisation, agents only possess their own plans, and that trust is transitive, this criterion is satisfied for free. In

[1]Instead of considering atoms of the form *C(t)[*`ont`*(oid),* `src`*(bsrc)]* and $R(t_1,\ t_2)$*[*`ont`*(oid),* `src`*(bsrc)]* as stated in the BNF in Figure 1, we consider the simpler form *C(t)[oid]*: having dropped sources, we do not need to explicitly state that *oid* is an ontology identifier any longer. The considerations we make for atoms of type *C(t)[oid]* apply for atoms having the form $R(t_1,\ t_2)$*[oid]* as well. For the sake of clarity, we do not deal with the latter form explicitly.

fact, when agents look for external plans for the first time, they ask their trusted agents an hence obtain plans whose sources are trusted. In successive plan exchanges, they may obtain plans whose sources are trusted from their trusted agents, and so on, hence meeting the constraint of "trust propagation". Belief sources might impact on querying the belief base and updating it in a similar way.

A. Plan Search in CooL-AgentSpeak

As far as plan search is concerned, introducing the "CooL" features to the AgentSpeak-DL language only changes the way relevant plans are retrieved.

A plan p with triggering event $\mathsf{TrEv}(p) = op'D(t')[oid']$ is relevant for the event $opC(t)[oid]$ ($op, op' \in \{+,-,+!,+?,-!,-?\}$) if the operator op is the same as op', t and t' unify, and:

1) if the two events refer to the same ontology ($oid = oid'$)
   a) either D is identical to C (as in AgentSpeak),
   b) or C can be inferred from D in ontology oid by means of ontological reasoning (as in AgentSpeak-DL);
2) otherwise, if the two events refer to different ontologies ($oid \neq oid'$), C in oid can be matched with D in oid' using the matching strategy and threshold adopted by the agent that is looking for the plan p (CooL-AgentSpeak new feature).

Besides all the above, relevant plans can be both local and external ones (as in Coo-AgentSpeak). Below, we formalise the these intuitions.

We use the notation $\mathtt{ont}(oid)$, where oid is an ontology identifier, to indicate the ontology (i.e., an ABox and a TBox) identified by oid. Given an agent's strategy s (either cooperation or matching) we use the dot notation "$s.f$" to refer to the value assigned to field f of the strategy. For example, if the cooperation strategy cs of agent Tom contains the field $trustedAgents(Alice, Bob)$, then for Tom we have $cs.trustedAgent = \{Alice, Bob\}$.

Given plans ps, cooperation strategy cs, and matching strategy ms of an agent, and a triggering event $te = opC(t)[oid]$, we define the set of local relevant plans (LRP) and the set of external relevant plans (ERP) as follows.

Local Relevant Plans. LRP = $\mathsf{LocalRelPlans}(ps, ms, opC(t)[oid])$ is the set of pairs (p, θ) such that

- $p \in ps$,
- $\mathsf{TrEv}(p) = op'D(t')[oid']$,
- $op = op'$,
- $\theta = mgu(t, t')$, and
- if $oid = oid'$
  then $\mathtt{ont}(oid) \models C \sqsubseteq D$
  else $\langle C, D, \equiv, Conf\rangle \in ms.match(\mathtt{ont}(oid), \mathtt{ont}(oid'), ms.parameters, ms.resources)$ and $Conf \geq ms.threshold$

The function RetrieveExtRelPlans returns the set of local relevant plans possessed by each agent a in a given set ags for the given triggering event te. If ontology matching techniques are used, the confidence in the matching between te and the triggering event te_{local} of plans local to a must be greater than the threshold thr set by the agent looking for external plans. The $m.g.u.$ between the arguments of te and te_{local} is also returned.

Formally, we define $ms[thr'/thr]$ the ontology matching strategy ms where the threshold thr has been replaced by thr'.

$$\mathsf{RetrieveExtRelPlans}(te, thr, ags) = \bigcup_{a \in \{ags\}} \mathsf{LocalRelPlans}(ps_a, ms_a[thr/ms_a.threshold], te)$$

Note that we use $ms_a[thr/ms_a.threshold]$ as the second argument of LocalRelPlans, to ensure that the threshold used by agent a when applying its matching strategy ms_a is thr, namely the one of the agent that is looking for external plans, which might be more or less restrictive than a's own threshold.

Having defined this auxiliary function, we are now ready to define the set of external plans relevant for a given event.

External Relevant Plans. $ERP =$ $\mathsf{ExternalRelPlans}(cs, ms, te)$ is defined as

- $\emptyset$ if $cs.retrievalPolicy = noLocal$ and $LRP \neq \emptyset$,
- $\mathsf{RetrieveExtRelPlans}(te, ms.threshold, cs.trustedAgents)$ otherwise.

Relevant Plans. The set of relevant plans is the union of local and external relevant plans: $RP = \mathsf{RelPlans}(ps, cs, ms, te) = LRP \cup ERP$.

As far as the rules for applicable plans are concerned, they are the same reported in [2]. If we applied ontology matching techniques to the verification of context satisfiability as well, they would have required changes accordingly.

B. Querying the Belief Base

The execution of actions and the execution of achievement goals are not affected by the introduction of the CooL features and their semantics are the same as in AgentSpeak-DL. The evaluation of a test goal $?C(t)[oid]$, however, requires taking advantage of ontological reasoning and ontology matching. The function that tests if an atom is a logical consequence of the agent's beliefs and returns the set of substitutions that satisfy the test goal, used in the rules of the original semantics, is the only thing that changes with respect to predicate logic AgentSpeak and AgentSpeak-DL. In CooL-AgentSpeak, it is redefined as follows:

Testing whether a test goal derives from the agent's beliefs. $\mathsf{Test}(bs, C(t)[oid]) = \{\theta \mid \mathtt{ont}(oid) \models C(t)\theta\} \cup \{\theta \mid \exists D(t')[oid'] \in bs, \langle C, D, \equiv, Conf\rangle \in ms.match(\mathtt{ont}(oid), \mathtt{ont}(oid'), ms.parameters, ms.resources),\ Conf \geq ms.threshold,\ \theta = mgu(t, t')\}$.

To give an example, let us consider the goal $?paper(Id, iat, 2011)[o1]$ to be tested by agent a. If a possesses the belief $article(coolAgentSpeak, iat, Edition)[o2]$ and — according to a's matching strategy — $paper \in o1$ is equivalent to $article \in o2$ with confidence greater than the threshold, then $\theta = \{Id \leftarrow coolAgentSpeak, Edition \leftarrow 2011\}$ should be returned by the Test function.

C. Belief Updating

In *Jason*, beliefs are changed through perception (sensing the environment), through agent communication, and also through plan execution; in the latter case, beliefs are called "mental notes" and used by an agent to remind itself of things that have happened, or things it has done, for example. There is no automatic check for consistency, which means that, unless programmers are very careful, there is considerable chance that the belief base will become contradictory.

One advantage of having references to ontologies annotating individual beliefs is that, at least for those beliefs, logical consistency can be checked automatically using the underlying ontological reasoner, whenever a change in the belief base is to take place. In [4], a mechanism was created that rolled back the ontology to its previous state in case an attempt to update the part of the belief base that resided in a particular ontology would make that ontology inconsistent.

In CooL-AgentSpeak, the addition of ontology matching makes things even more complicated than in AgentSpeak-DL, regarding revision. In principle, all previous matching of concepts in different ontologies should be taken into consideration when checking for consistency. However, if intra-ontology consistency is already rather heavy for a practical interpreter such as JASDL, consistency across different ontologies, particularly for agents that make reference to large numbers of ontologies, is unlikely to be possible in practice. We aim to do further work to empirically assess the feasibility of such consistency checks in practice.

V. Implementing CooL-AgentSpeak in *Jason*

In this section we outline how the RetrieveExtRelPlans and ontology match functions can be implemented in *Jason*.

A. Retrieving External Plans

In order to implement the RetrieveExtRelPlans function, we use a multicast synchronous ask with timeout that has been added to *Jason* specifically for this purpose. This action sends a multicast message with `askHow` performative, used to ask the receiver if it has plans relevant for the triggering event passed as argument [3], and combines the answers from multiple agents until the deadline is reached. The intention that requested the plans is suspended until all agents answer or the deadline is expired. For example, `.send([ag1,ag2,ag3], askHow, +article(Title, Conf, Y)[ont(bibtex)], Ans, 100)` either returns after 100 ms with `Ans` unified with the list of plans relevant for the *+article(Title, Conf, Y)[*`ont`*(bibtex)]* event[2] that arrived before 100 ms (if at least one agent among `ag1`, `ag2`, and `ag3` did not reply in time), or resumes before 100 ms with `Ans` containing all the expected answers, if `ag1`, `ag2`, `ag3` all reply in time.

Since we also want to take the acceptable matching threshold *thr* into account, we annotate the triggering event with it.

The relationship between the abstract definitions and the concrete implementation of the function for retrieving external relevant plans is the following (given that the timeout to be used is `T` and that `[Ags]` represents the list of agents Ags in the correct *Jason* format):
$\mathsf{RetrieveExtRelPlans}(Te[\mathtt{ont}(O), src(S)], Thr, Ags) = Answ$ *iff* `.send([Ags], askHow, Te[ont(O), src(S), thr(Thr)], Answ, T)`.

B. Matching Ontologies

If each agent in the MAS were able to retrieve ontologies given their identifiers and to match them using its $ms.match$ function, ontology matching activities should be fairly distributed among agents, each being able to perform this task. Although this situation may occur in some cases, it seems nevertheless unrealistic to simply assume that *all* agents in a MAS are able to perform ontology matching activities: in the experience of the authors involved in industrial collaborations aimed at designing and developing multi-agent systems for solving real problems ([13], [14], just to cite two recent ones), no agent had this capability. In the spirit of the "ontology service" FIPA specification discussed in Section II-C, we think that the execution of the match function should be delegated to one or more Ontology Agents always available in the MAS. When agent a must perform an action that involves matching two ontologies, it just sends the ontology identifiers, the match function to be used, its parameters, and the atom $C(t)$ involved in the matching to the Ontology Agent(s). It is up to it performing all the work by offering the services foreseen and standardised by the FIPA specification.

With the help of M. Barbieri, a student of the Artificial Intelligent course at the Computer Science Department of Genova University, we started experimenting with the feasibility of this approach by integrating a "quasi" FIPA-compliant[3] Ontology Agent developed for Jade by another student, F. Mulattieri [9], into *Jason*. That Ontology Agent was developed independently from our research on CooL-AgentSpeak, and for other and more general purposes. We believe that the agent-based approach may contribute to solve the problem of semantic interoperability due to the standardisation effort made by FIPA in the last years: a fully-fledged FIPA-compliant framework provides Ontology

[2]This event can be expressed according to the existing Bibtex ontology that can be downloaded from http://zeitkunst.org/bibtex/0.1/bibtex.owl.

[3]"Quasi" is only due to the adoption of OWL instead of OKBC as the ontology representation language.

Agents offering the semantic-related services that other agents need, instead of leaving to the developer the entire burden of addressing semantic issues. Querying Ontology Agents may become a standard step operated during the agent life cycle for addressing interoperability issues at the level of the agent oriented programming language, rather than done in an *ad hoc* manner for each application.

A simple example of interaction requiring ontology matching capabilities is shown in Figure 2. DomesticRobot and SuperMarket are *Jason* agents, whereas OA is the Jade Ontology Agent:

- DomesticRobot wants to buy *Chips* from the SuperMarket;
- SuperMarket does not know *Chips*, and asks OA to match the concept *Chips* with some concept in its own ontology;
- OA tells SuperMarket that *Crisps* in SuperMarket's ontology and *Chips* in DomesticRobot ontology correspond with confidence 1;
- SuperMarket tells DomesticRobot that it can sell *Crisp*, that might correspond to what DomesticRobot is looking for.

The use of an Ontology Agent for matching beliefs and goals within the BDI reasoning cycle would be trickier than this simple example, but we are confident that this is the right and most practical approach to follow for implementing ontology matching services in *Jason*.

VI. Related and Future Work

Recent work by C. Fuzitaki, Á. Moreira, and R. Vieira [15] stems from [2] and proposes the core of a logic agent-oriented programming language based on DL-Lite [16]. With respect to [2], the work by Fuzitaki *et al.* addresses ontological reasoning providing efficient algorithms for belief base querying, plan selection, and belief update and removal that were not defined there. However, no implementation in an agent programming framework is proposed, whereas in this paper we provide an outline of how to implementing CooL-AgentSpeak in *Jason* by extending the implementation discussed in [4].

In [17] and [18], K.L. Clark and F.G. McCabe explore the use of a formal ontology as a constraining framework for the belief store of a rational agent and show the implementation of their proposal in the `Go!` multi-threaded logic programming language [18]. A `Go!` agent typically comprises several threads that implement different aspects of the agent's behaviour and which share a set of updatable objects. These are used to represent the agent's changing beliefs, desires, and intentions. In the `Go!` "ontology oriented programming" extension, the static beliefs of the agent are the axioms of the ontology whereas the dynamic beliefs are the descriptions of the individuals that are instances of the ontology classes. Belief updates not conforming to the axioms lead to either rejection of the update or some other revision of the dynamic belief store to maintain consistency. Our work and that by Clark and McCabe both share the aim of integrating ontologies in a language suitable for programming BDI agents. However, whereas their work mainly aims at defining a mapping between OWL-Lite constructs and labelled theories in the `Go!` language, losing references to the external ontologies which define the agents' vocabulary, our work implicitly assumes that ontologies exist outside the agents' "mind" and makes explicit the references to external ontologies so as to realise semantic integration among agents as envisioned by the work on the semantic web, through ontologies made available on the web.

Neither [15] nor [17], [18] take ontology matching into account as a means for inferring "cross-ontological knowledge" and none of them consider "cross-ontological reasoning" for exchanging behavioural knowledge.

Of course, "cross-ontological" knowledge and reasoning may lead to unwanted behaviour. Even those correspondences of maximum confidence might be semantically wrong and this might cause a wrong match to be used Think for example of the "bank" word that has different meanings (the bank of a river, the bank where we save money, besides many other ones). Smart ontology matching algorithms using word sense disambiguation techniques are able to understand when the meaning of the "bank" concept in two ontologies is different according to the neighbouring concepts in the ontology, to the comments that label the concept itself, and to other contextual information [19]. Hence, these matching algorithms will not return the correspondence $\langle bank, bank, \equiv, 1\rangle$ if they realise that the meaning of the words is different in the two ontologies, despite the homonymy. However, this is not always the case, and simpler matching algorithms looking only at the string distance (but even smart matching algorithms that have not enough contextual knowledge available) would return such correspondence.

The consequence might be that an agent pursuing the goal of "putting money in a safe place" could retrieve an external plan saying "put them in the nearest bank", and, if the agent still does not know how to pursue this second goal, it might retrieve an external plan leading it to leave its money in the nearest river bank, because of ambiguity of the word "bank". However, similar misunderstandings might occur even among human beings, although contextual information in human communication is usually greater than that available to software agents.

If we use ontology matching techniques, we must be aware of their average precision and recall, which are definitely lower that 100% even for the best performing algorithms (see http://oaei.ontologymatching.org/2010/results/benchmarks/index.html). In order to cope with this intrinsic limitation of ontology matching techniques available today, we might extend the agents' strategies in order to tag some events as sensitive, and avoid using ontology matching

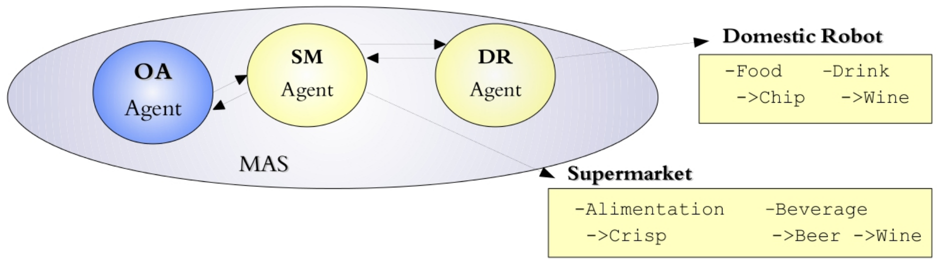

Figure 2. CooL-AgentSpeak OA at Work (courtesy of Matteo Barbieri).

techniques and/or to retrieve external plans for dealing with them.

It is part of our future work to fully implement the CooL-AgentSpeak language in *Jason* and to test it in realistic scenarios, by experimenting with different matching strategies. The safety issues arising from the risk of using wrong ontology matching will be investigated as well.

REFERENCES

[1] D. Ancona and V. Mascardi, "Coo-BDI: Extending the BDI model with cooperativity," in *Proc. of DALT*, ser. LNCS, J. A. Leite, A. Omicini, L. Sterling, and P. Torroni, Eds., vol. 2990. Springer, 2004, pp. 109–134.

[2] Á. F. Moreira, R. Vieira, R. H. Bordini, and J. F. Hübner, "Agent-oriented programming with underlying ontological reasoning," in *Proc. of DALT*, ser. LNCS, M. Baldoni, U. Endriss, A. Omicini, and P. Torroni, Eds., vol. 3904. Springer, 2006, pp. 155–170.

[3] R. Vieira, Á. F. Moreira, M. Wooldridge, and R. H. Bordini, "On the formal semantics of speech-act based communication in an agent-oriented programming language," *J. Artif. Intell. Res.*, vol. 29, pp. 221–267, 2007.

[4] T. Klapiscak and R. H. Bordini, "JASDL: A Practical Programming Approach Combining Agent and Semantic Web Technologies," in *Proc. of DALT*, ser. LNCS, M. Baldoni, T. C. Son, M. B. van Riemsdijk, and M. Winikoff, Eds., vol. 5397. Springer, 2009, pp. 91–110.

[5] D. Ancona, V. Mascardi, J. F. Hübner, and R. H. Bordini, "Coo-AgentSpeak: Cooperation in AgentSpeak through plan exchange," in *Proc. of AAMAS*. IEEE Computer Society, 2004, pp. 696–705.

[6] W3C, "OWL Web Ontology Language Overview – W3C Recommendation 10 February 2004." [Online]. Available: http://www.w3.org/TR/owl-features/

[7] M. Obitko and V. Snášel, "Ontology repository in multi-agent system," in *Proc. of AIA*, M. H. Hamza, Ed., 2004.

[8] D. Briola, A. Locoro, and V. Mascardi, "Ontology agents in FIPA-compliant platforms: a survey and a new proposal," in *Proc. of WOA*, M. Baldoni, M. Cossentino, F. De Paoli, and V. Seidita, Eds. Seneca Edizioni, 2008.

[9] F. Mulattieri, "Progettazione ed implementazione di un ontology agent," Master's thesis, DISI, CS Department, University of Genova, Italy, 2010, in Italian.

[10] J. Euzenat and P. Shvaiko, *Ontology Matching*. Springer, 2007.

[11] F. Baader and W. Nutt, "Basic description logics," in *Description Logic Handbook*, F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds. Cambridge University Press, 2003, pp. 43–95.

[12] R. H. Bordini and Á. F. Moreira, "Proving BDI properties of agent-oriented programming languages," *Ann. Math. Artif. Intell.*, vol. 42, no. 1-3, pp. 197–226, 2004.

[13] D. Briola, V. Mascardi, and M. Martelli, "Intelligent agents that monitor, diagnose and solve problems: Two success stories of industry-university collaboration," in *J. of Inf. Assurance and Security*, vol. 4, 2009, pp. 106–117.

[14] A. Locoro, V. Mascardi, F. Mortara, and R. Sanna, "Managing unavailabilities in a dynamic scenario following an agent-based approach," in *Proc. of WOA*, 2011, to appear.

[15] C. Fuzitaki, A. Moreira, and R. Vieira, "Ontology reasoning in agent-oriented programming," in *Advances in Artificial Intelligence SBIA 2010*, ser. LNCS, A. da Rocha Costa, R. Vicari, and F. Tonidandel, Eds. Springer Berlin / Heidelberg, 2011, vol. 6404, pp. 21–30.

[16] D. Calvanese, G. De Giacomo, D. Lemho, M. Lenzerini, and R. Rosati, "DL-Lite: tractable description logics for ontologies," in *Proc. of Nat. Conf. on Artificial intelligence - Vol. 2*. AAAI Press, 2005, pp. 602–607.

[17] K. L. Clark and F. G. McCabe, "Ontology schema for an agent belief store," *Int. J. Hum.-Comput. Stud.*, vol. 65, pp. 640–658, July 2007.

[18] ——, "Go! a multi-paradigm programming language for implementing multi-threaded agents," *Ann. Math. Artif. Intell.*, vol. 41, pp. 171–206, 2004.

[19] V. Mascardi, A. Locoro, and F. Larosa, "Exploiting Prolog and NLP techniques for matching ontologies and for repairing correspondences," in *Proc. of CILC*, 2009.