Web Intelligence and Agent Systems: An International Journal 5 (2007) 1–5
IOS Press

CooL-AgentSpeak: Endowing AgentSpeak-DL Agents with Plan Exchange and Ontology Services [1]

Viviana Mascardi [a,*], Davide Ancona [a], Matteo Barbieri [a], Rafael H. Bordini [b] and Alessandro Ricci [c]
[a] *DIBRIS – University of Genova*
Via Dodecaneso 35, 16146, Genova, Italy
Email: viviana.mascardi@unige.it, davide.ancona@unige.it, matteo.barbieri@oniriclabs.com
[b] *FACIN – PUCRS, Av. Ipiranga 6681, 90619-900, Porto Alegre - RS, Brazil*
Email: r.bordini@pucrs.br
[c] *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 an ontologically relevant 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 also take advantage of ontological reasoning and matching.

Keywords: AgentSpeak, cooperation, plan exchange, ontology matching

1. Introduction

Cooperation is an important feature in the context of Multi-Agent Systems (MASs) and is intrinsic in the definition of a MAS as "*a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver*" [30].

In our past research we discussed some scenarios where cooperation obtained by allowing BDI agents [50] to exchange their plans would have turned out to be extremely useful. We named that extension to the standard BDI approach Coo-BDI [1] and one of the scenarios we took under consideration involved unexperienced digital butlers [48] needing to cooperate with more experienced ones in order to better assist their user. In that scenario a digital butler a might need to manage the event *+invitee(john)* 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)* that states how to make guests feel as comfortable as possible by offering them all the hospitality that they deserve. Unfortunately, *+invitee(john)* and *+visitor(Who)* do not unify, and b will not send its nice plan to a for not realizing it is in fact relevant.

[1]This paper extends the work by Viviana Mascardi, Davide Ancona, Rafael H. Bordini and Alessandro Ricci "CooL-AgentSpeak: Enhancing AgentSpeak-DL Agents with Plan Exchange and Ontology Services" published in the Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, pp. 109-116, 2011.
[*]Corresponding author. E-mail: viviana.mascardi@unige.it

Now, let us suppose that agents' belief base 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 introduced in [46].

If a and b shared the same ontology $\mathsf{o}(oid)$, and if we could demonstrate that $\mathsf{o}(oid) \models invitee \sqsubseteq visitor$, we could solve a's problem: *+invitee(john)* and *+visitor(Who)* 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 $\mathsf{o}(oid)$.

Sharing a common ontology to boost the cooperation among agents was perceived as a pressing need since the dawning of MASs [23,29] and was implemented in many different domains including Microgrids [16], health care [59], trading [62], automated scheduling [25], cultural heritage [39]. Nonetheless, in many situations designing or eliciting such a common ontology may not be convenient, desirable or possible. For example, companies that are temporarily allied in a Virtual Enterprise might not want to disclose their local ontology to all the partners, but might be ready to share the minimal amount of information required to achieve the specific goal for which the Virtual Enterprise was built. A similar situation may take place among the forces of law and order, among different health care institutes, and among many other actors that on the one hand want to collaborate in order to be more effective, but on the other hand need to protect their own sensible knowledge. Matching their ontologies and exploiting only the resulting alignment may be a good compromise to trade-off between privacy and need to cooperate, and to cope with dynamic environments where matching must be necessarily performed on-the-fly [47] and with situations where agents do not interact often and creating a shared ontology is not worth [27].

With these considerations in mind, let us consider now a more involved scenario, where a and b do not share the same ontology (they refer to $\mathsf{o}(a)$ and $\mathsf{o}(b)$ respectively), and where b's plan is triggered by *+guest(Who)*. 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 concepts allowing b to realize that $guest \in \mathsf{o}(b)$ is equivalent (at least up to a certain degree) to $invitee \in \mathsf{o}(a)$. In this case, b could send its plan for dealing with guests to a, and a could use it for dealing with the invitee. If agents a and b could take advantage of some transparent and reliable mechanism performing the necessary cross ontological unification without requiring that they disclose their ontologies to each other, they could reach the goal of cooperating, still preserving their privacy.

The CooL-AgentSpeak language presented in this paper copes with the need of cooperating by exchanging procedural knowledge expressed according to ontologies local to the agents, without needing to build a common ontology or to share the local ones. CooL-AgentSpeak integrates Coo-BDI and AgentSpeak-DL and enhances the resulting language with *ontology matching capabilities* to deal with situations such as the one discussed above. In CooL-AgentSpeak 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 also take advantage of ontological matching.

The paper is organized in the following way. Section 2 provides background knowledge on the integration of speech-acts in AgentSpeak, AgentSpeak-DL, Coo-BDI, ontology services in MAS, and ontology matching techniques. Section 3 introduces the CooL-AgentSpeak language and Section 4 outlines its semantics in an informal way. Section 5 describes the implementation of CooL-AgentSpeak in *Jason*; a simple example of its use is shown in Section 6, whereas the experiments we carried out on three complex scenarios are discussed in Section 7. Section 8 discusses the related work and finally Section 9 provides final remarks and an outline of our future research directions.

2. Background

2.1. Speech-acts in AgentSpeak

Many extensions of AgentSpeak have appeared over the years. In [44], for example, Moreira and colleagues paved the way to the definition of the formal semantics of AgentSpeak agents able to process speech-act based messages, which is fundamental to allow social behavior in BDI agents. That preliminary work led to the full formalization of a large set of speech-acts in AgentSpeak presented in [57].

An agent's message $\langle sa, id, cnt\rangle$ consists of a speech-act *sa*, a unique sender identifier *id*, and a message content *cnt*; depending on the speech-act, *cnt* can be an atomic formula (at); a set of formulas (ATs); a ground atomic formula (b); a set of ground atomic formulas (Bs); a set of plans (PLs); or a triggering event (te). The informal semantics of each speech act is given below.

– $\langle$*Tell, id, Bs*$\rangle$ and $\langle$*Untell, id, ATs*$\rangle$: a *Tell* message might be sent to an agent either as a reply or as an "inform" action. When receiving a *Tell* message as an inform, the receiver will include the beliefs in the message content in its knowledge base and will annotate the sender as a source for them. When receiving an *Untell* message, the sender of the message is removed from the set of sources associated with the atomic formulas in the content of the message. In case the *Tell* or *Untell* message is sent as the reply to a previously issued message of type *Ask*, the suspended intention associated with that message is resumed.

– $\langle$*Achieve, id, at*$\rangle$ and $\langle$*Unachieve, id, at*$\rangle$: in an appropriate social context, the receiver of an *Achieve* message will try to execute a plan whose triggering event is *+!at*: the sender *delegates* the receiver to achieve that goal. The *Unachieve* speech-act is dealt with in a similar way, except that the deletion (rather than addition) of an achievement goal is included in the receiver's set of events.

– $\langle$*TellHow, id, PLs*$\rangle$ and $\langle$*UntellHow, id, PLs*$\rangle$: a *TellHow* message is used by the sender to inform the receiver of a plan that can be used for handling certain types of events as expressed in the plan's triggering event. This performative is fundamental for the implementation of plan exchange in Coo-AgentSpeak (Section 2.3). The management of *UntellHow* is similar, except that plans are removed from the receiver's plan library.

– $\langle$*AskIf, id, {b}*$\rangle$, $\langle$*AskAll, id, {at}*$\rangle$, and $\langle$*AskHow, id, te*$\rangle$: The receiver will respond to these requests for information if certain conditions imposed by the social settings hold between sender and receiver. The receiver processing an *AskIf* responds with the action of sending either a *Tell* (to reply positively) or *Untell* (to reply negatively) with the same content as the *AskIf* message. In case of an *AskAll*, the agent replies with all the atoms in the belief base that unify with the formula in the message content or with an *Untell*. Finally, the receiver of an *AskHow* responds with a *TellHow* message.

A further development of that research line is discussed in [45], where the authors revisit the motivations and the initial developments that led to their paper [44] and provide an overview of the state-of-the-art in the field.

2.2. AgentSpeak-DL and its JASDL Implementation

In agent communication, the importance of ontologies for ensuring interoperability has been recognized since their very beginning, even before they have been employed for the Semantic Web effort. Both KQML [42] and FIPA-ACL [22] allow agents to specify the ontology they are using, although none of them forces that. Agent communication languages were born with the Semantic Web in mind. However, what was not considered before the work in [46] 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 (operational) 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 assumed to be 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 the OWL language [58], for example.

Exactly to allow the practical use of these ideas, extensive work was carried out by Klapiscak and Bordini who implemented JASDL [35], an extension of the *Jason* AgentSpeak interpreter making available all features of AgentSpeak-DL and others, including preliminary work on 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; this means that traditional AgentSpeak code can be used together with AgentSpeak-DL code. The OWL API [28] was used to allow the integration with ontological reasoners which would make the knowledge available elsewhere (in ontologies on the web) usable within an agent program, so as to allow, for example, for more compact programs that can handle various subsumed events by a single, more general, plan (when appropriate).

2.3. 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 characterized 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 modifications in 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[1];

– `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 reasoning), and made practical using the *Jason* interpreter [2]. Its further developments are discussed in [38].

2.4. Ontology Services in MASs

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 [21]. According to that specification, an "Ontology Agent" (OA) 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 analyze in a systematic way the services that an OA should provide for ensuring semantic interoperability in an open MAS, it has many limitations including the model to which ontologies should adhere (OKBC[2], when the most widely used language for representing ontologies today is OWL) 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 FIPA-compliant OAs. The only two attempts of integrating a FIPA-compliant OA into JADE, that we are aware of, are described in [9] and [49]. Both follow the FIPA specification but adapt it to ontologies represented in OWL.

An extension of the OA, described in [9], with services for ontology access, navigation, querying, modification, and versioning of modified ontologies, has been exploited for supporting CooL-AgentSpeak features.

2.5. Ontology Matching

According to [20], a correspondence between an entity e belonging to ontology *ont* and an en-

[1]The *TrustedAgents* set is implemented as a Prolog list without repetitions.

[2]`http://www.ai.sri.com/~okbc/`, accessed on November 2013.

tity e' belonging to ontology *ont′* is a 5-tuple $\langle id, e, e', R, conf\rangle$ where:

– id is a unique identifier of the correspondence;

– e and e' are the entities (e.g., properties, classes, individuals) of *ont* and *ont′* 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) for the correspondence between the entities e and e'.

An alignment of ontologies *ont* and *ont′* is a set of correspondences between entities of *ont* and *ont′*.

Finally, a matching process can be seen as a function f which takes two ontologies *ont* and *ont′*, a set of parameters *Par*, and a set of oracles and resources *Res*, and returns an alignment A between *ont* and *ont′*.

Since in our work we use equivalence as relation, and we do not need the identifiers of correspondences, in the remainder of this paper we will represent correspondences as triples $\langle e, e', conf\rangle$.

3. The Language

CooL-AgentSpeak stands for "Cooperative description-Logic AgentSpeak". The syntax of the language is summarized in Figure 1. With respect to previous work on AgentSpeak-DL and JASDL, the definition of a matching strategy ms is a completely new feature of CooL-AgentSpeak.

Ontological knowledge. Following [46], we assume $\mathcal{ALC}$ as the underlying description logic [3] 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 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 relations between classes and between properties. With respect to [3] and [46], 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.

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

Fig. 1. CooL-AgentSpeak: Syntax.

An agent belief is an atom belonging to the ABox annotated with $\texttt{o}(oid)$, where *oid* is the identifier of the ontology. We use *oid=self* for "naive beliefs" [35], i.e., normal AgentSpeak beliefs that do not relate to any ontology. Along the lines of [57], beliefs are also annotated with sources $\texttt{src}(bsrc)$, 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. The metavariable f represents a matching-function name. We assume that matching functions can be unequivocally identified by means of their names (i.e., the functional symbols). *Par* and *Res* are metavariables representing the parameters and resources needed by matching function *f*.

Agent. An agent *a* is characterized 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 a plan label (preceded by `@`, and *t*, as elsewhere, stands for a simple term, in practice a lower-case identifier as in Prolog) and plan annotations which include an access specifier *as* (defining the accessibility level for the plan, as introduced in Section 2.3) and an optional list of sources `src`(*psrc*) which specifies the agents from which the plan has been obtained, a trigger *te*, a context *ct*, and a body *h*.

Cooperation strategy. The cooperation strategy *cs* follows the description given in Section 2.3 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 2.5 and consists of:

– `match`(*f*) specifying which matching function *f* the agent uses to perform the match;

– `parameters`(*Par*) and `resources`(*Res*) define the arguments 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 change dynamically as well, thus allowing an agent to use different matching functions and different parameters throughout its execution.

Note that above we only explained mainly the syntactic aspects that are specific to the CooL-AgentSpeak language being introduced in this paper and the CooBDI approach. The interested reader can find detailed descriptions of the other programming constructs inherited from AgentSpeak, AgentSpeak-DL, and JASDL in the relevant literature already cited.

4. 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 [6] and in AgentSpeak-DL [46], 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 local and external relevant plans (`RelPl`);

– *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` (belief update is normally considered part of the underlying agent architecture, so the formal semantics of an AgentSpeak interpreter usually just abstracts away from this aspect). Percepts can be annotated with a reference to an ontology as well.

Recall that, in CooL-AgentSpeak, both literals (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. 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 a difference in the plan search stage would 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 initialization, agents only possess their own plans, and that trust is transitive, this criterion is satisfied for free. In 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 by 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.

4.1. 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[3].

A plan p with triggering event

$$\mathsf{TrEv}(p) = op' D(t')[o(oid')]$$

is relevant for the event $op\ C(t)[o(oid)]$ ($op, op' \in \{+, -, +!, +?, -!, -?\}$) 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 $\mathsf{o}(oid)$[4] by means of ontological reasoning (as in AgentSpeak-DL);
2. otherwise, if the two events refer to different ontologies ($oid \neq oid'$), then it must be the case that C in $\mathsf{o}(oid)$ can be matched with D in $\mathsf{o}(oid')$ using the matching strategy and threshold adopted by the agent that is looking for the plan p (CooL-AgentSpeak new feature). Note that the match can be delegated to some component of the MAS that necessarily knows the ontologies to be matched, but that committed with the agents not to disclose them to others. This ensures that agents can communicate without sharing their own ontologies among themselves.

Besides all the above, relevant plans can be both local and external ones (as in Coo-AgentSpeak). Below, we formalize these intuitions. Given an agent's strategy s (either cooperation or matching) we use the dot notation "$s.fld$" to refer to the value assigned to field fld of the strategy. For example, if the cooperation strategy cs of agent Tom contains the field $trustedAgents([Alice, Bob])$, then for Tom we have $cs.trustedAgent = \{Alice, Bob\}$.

[3]For the sake of clarity, in the sequel we limit ourselves to deal with *C(t)[o(oid)]* literals, avoiding to deal with literals having the form *R(t_1, t_2)[o(oid)]* explicitly.

[4]We use the notation $\mathsf{o}(oid)$, where *oid* is an ontology identifier, to indicate the ontology (i.e., an ABox and a TBox) identified by *oid*.

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

Local Relevant Plans.
$LRP = \mathsf{LocalRelPlans}(ps, ms, op\ C(t)[o(oid)])$ is the set of pairs (p, θ) such that
– $p \in ps$,
– $\mathsf{TrEv}(p) = op' D(t')[o(oid')]$,
– $op = op'$,
– $\theta = mgu(t, t')$, and
– if $oid = oid'$ then $\mathsf{o}(oid) \models C \sqsubseteq D$ else $\langle C, D, conf\rangle \in ms.match(\mathsf{o}(oid), \mathsf{o}(oid'), ms.parameters, ms.resources)$ and $conf \geq ms.threshold$.

The function RetrieveExtRelPlans returns the set of all local relevant plans owned 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 th set by the agent looking for external plans. The $m.g.u.$ between the argument of te and te_{local} is also returned.

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

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

Note that we use $ms_a[th/ms_a.threshold]$ as the second argument of LocalRelPlans, to ensure that the threshold used by agent a when applying its matching strategy ms_a is th, namely the threshold 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 [46]. If we applied ontology matching techniques to the verification of context satisfiability as well, they would have required changes accordingly.

4.2. Querying the Belief Base

The execution of actions and achievement goals is 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 $?C(t)[o(oid)]$, however, requires ontological reasoning (as in AgentSpeak-DL) and ontology matching. Hence, the only component of the original semantics that needs to be modified is the function that tests whether an atom is a logical consequence of the agent's beliefs and returns the set of substitutions that satisfy the test goal. In CooL-AgentSpeak, it is redefined as follows:

$$\begin{gathered}\mathsf{Test}(bs, C(t)[o(oid)]) = \\ \{\theta \mid \mathsf{o}(oid) \models C(t)\theta\} \cup \\ \{\theta \mid \exists D(t')[o(oid')] \in bs, \\ \langle C, D, conf\rangle \in ms.match(\ \mathsf{o}(oid), \mathsf{o}(oid'), \\ ms.parameters, ms.resources), \\ conf \geq ms.threshold, \theta = mgu(t, t')\}.\end{gathered}$$

To give an example, let us consider the following goal, to be tested by agent a:

$$?paper(inst(Id, Title, Year))[o(o1)].$$

If a has the belief

$$article(inst(coolAS, \texttt{"}CooL\ldots\texttt{"}, 2013)[o(o2)]$$

and — according to a's matching strategy — the class $paper \in \mathsf{o}(o1)$ is equivalent to the class $article \in \mathsf{o}(o2)$ with confidence greater than the threshold, then $\theta = \{Id \leftarrow coolAS, Title \leftarrow \texttt{"}CooL\ldots\texttt{"}, Year \leftarrow 2013\}$ should be returned by the Test function.

4.3. 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 [35], a mechanism was created that rolled back the ontology to its previous state in case of inconsistent updates.

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.

5. Design and Implementation

The design of CooL-AgentSpeak is centered around an Ontology Artifact (OntArt in the sequel) defined according to the "Agents and Artifacts" (A&A) model [53] and the CArtAgO framework [52], offering the services foreseen by the FIPA Ontology Agent proposal.

Following A&A and CArtAgO, artifacts have been conceived to program and build a suitable agent working context or environment: a set of passive resources and tools encapsulating functionalities and services that agents can share and exploit to support their individual as well as social activities. A simple example is given by a blackboard artifact, that agents can use to communicate besides direct message passing.

In general, artifacts provide an effective way to design and program those components of a MAS that do not need to be autonomous or pro-active, but rather flexibly observable and usable by agents - without worrying about issues related to concurrency (that is, multiple agents using concurrently the same artifact) and distribution. Accordingly, they can be effectively used to model and implement those ontology services and functionalities described so far. In particular, we designed an ontology artifact functioning as an ontology repository tool - to store a (possibly dynamic) set of ontologies - and offering related ontology matching and alignment functionalities.

In order to be used by agents, an artifact provides a usage interface, composed by the set of actions that an agent can perform on it (called "operations" on the artifact side) and a set of observable properties, representing the observable state of the artifact that agents may need to perceive according to the artifact's functionality [51].

Currently, the usage interface of the Ontology Artifact includes the operations described below[5] working on ontologies expressed in OWL. The problems raised by the agent's willingness to keep their ontologies private can be easily managed by extending these operations with access control policies, hence allowing only the owner of an ontology to download, query, and modify it, but still allowing all the agents to match their own ontologies with other ones.

– `register` ⟨`oid`⟩ ⟨`uri`⟩ `[tags]`:
registers the ontology whose URI is `uri` to the Ontology Artifact, and identify it by `oid` (required to be unique in the MAS); `tags` is an optional list of keywords.

– `download` ⟨`oid`⟩:
downloads the ontology identified by `oid`.

– `look_for_ontology` ⟨`tags`⟩ ⟨`result`⟩:
looks for ontologies tagged with `tags`.

– `query` ⟨`oid`⟩ ⟨`RDQLq`⟩ ⟨`result`⟩:
performs query `RDQLq` on ontology `oid`.

– `add_property` ⟨`oid`⟩ ⟨`resource_uri`⟩ ⟨`property_uri`⟩ ⟨`property_value_uri`⟩:
adds a property to a resource.

– `add_class` ⟨`class_uri`⟩:
adds a new class.

– `add_disjointwith` ⟨`first_class_uri`⟩ ⟨`disj_class_uri`⟩:
adds a disjointness axiom to a class.

– `add_subclass` ⟨`first_class_uri`⟩ ⟨`subclass_uri`⟩:
adds a subclass to a given class.

– `add_equivalentclass` ⟨`first_class_uri`⟩ ⟨`equiv_class_uri`⟩:
adds an equivalence axiom to a class.

– `add_individual` ⟨`class_uri`⟩ ⟨`individual_uri`⟩:
adds an instance to a class.

– `add_comment` ⟨`resource_uri`⟩ ⟨`comment`⟩ ⟨`language`⟩:
adds a comment in a given language to a resource.

– `remove_* parameters`:
everything that can be added can also be removed by specifying the same parameters as the corresponding `add_*` statements.

– `align` ⟨`oid1`⟩ ⟨`oid2`⟩ `[method]` ⟨`result`⟩:
matches ontologies `oid1` and `oid2` using `method` (if `method` is not specified, then the default method is the WordNet-based one provided by the Align API).

– `concept_match` ⟨`oid`⟩ ⟨`resource`⟩ `[method] [threshold]` ⟨`result`⟩:
looks for the resource closest to `resource` ∈ `oid` belonging to the ontologies registered by the agent that calls the operation, using `method` as matching function and `threshold` as acceptable threshold to consider a match reliable (if `method` is not specified, then the default one is used; if `threshold` is not specified, then the default value `0.9` is used). Methods currently supported are those provided by the open source Align API [19], that include JWNL among the others, and AROMA [13]. JWNL, `http://alignapi.gforge.inria.fr/`, computes a substring distance between the entity names of the first ontology and the entity names of the second ontology expanded with WordNet [43] synsets. AROMA `http://aroma.gforge.inria.fr/` is an hybrid, extensional and asymmetric matching method relying on the implication intensity measure, a probabilistic model of deviation from independence.

Operating instructions are a description of how to use the artifact to get its functionality, whereas the function of an artifact is its intended purpose, i.e. the purpose established by the designer/programmer of the artifact. CooL-AgentSpeak agents use OntArt by executing internal actions (each operation offered by OntArt corresponds to an implemented internal action)[6]. Because of the simplicity of the interaction with OntArt, where the particular agent intention that required an artifact operation is suspended until feedback is received from the artifact operation, no further operating instructions and function descriptions are required.

[5]If the operation returns a result, it is stored in the ⟨result⟩ output parameter. Parameters in square brackets are optional.

[6]Jason internal actions are implemented in Java as a Boolean method and support is given for binding of logical variables; they can appear in a plan wherever a literal is expected.

Finally, the structure and behavior concern the internal aspects of the artifact, that is, how the artifact is implemented in order to provide its function. OntArt is implemented in Java using the OWL API for ontology management and the already cited Align API for ontology matching.

Among the many operations offered by OntArt, we heavily exploited `concept_match` to implement the CooL-AgentSpeak features. When an agent a executes a `concept_match` ⟨`oid`⟩ ⟨`resource`⟩ `[method]` `[threshold]` ⟨`result`⟩ operation, OntArt performs the following actions:

1. for each ontology oid' registered by the agent that is calling the concept match operation
   (a) if oid and oid' were never matched before using $method$, then match them and store the resulting alignment A; otherwise, retrieve the stored alignment A;
   (b) return those tuples $\langle e, e', th\rangle$ in the alignment A where $th > threshold$.

We chose a lazy approach to ontology matching: ontologies are matched only when required for the first time. Resulting alignments are cached for further use, so each matching is computed only once. Default values for parameters are used if actual values are not specified. Cached alignments expire after a timeout set by the MAS developer. The timeout, after whose expiration the matching must be re-computed, is used to cope with the (possible) evolution of ontologies.

As far as the retrieval of relevant plans is concerned, CooL-AgentSpeak agents are characterized by the following behavior that is implemented — in such an integrated way — neither in JASDL nor in Coo-AgentSpeak.

1. When agent a starts its execution, it first registers all its ontologies with OntArt by calling the `register` operation.
2. When a needs to retrieve plans relevant for a given triggering event *op* $C(t)$, with $C \in \mathtt{o}(oid)$, it first looks for them using the approach supported by AgentSpeak (no ontological reasoning) and AgentSpeak-DL (intra-agent ontological reasoning).
3. Then, a calls the concept match operation offered by OntArt for looking for a match between C and concepts in its own local ontologies (those it registered to OntArt), different from oid, `concept_match` ⟨`oid`⟩ ⟨`C`⟩ `ms.method ms.threshold` (where `ms` is a's matching strategy).
4. According to a's retrieval strategy and to the outcome of the search within local ontologies, different actions may take place afterward:
   (a) if acceptable mappings $\langle C, D_1, th_1\rangle$, $\langle C, D_2, th_2\rangle$, ..., $\langle C, D_m, th_m\rangle$ have been found locally, and if a has a *noLocal* strategy, and at least one relevant plan triggered by *op* $D_1(t)$ or *op* $D_2(t)$ or ... or *op* $D_m(t)$ is locally available to a, no cooperation is required; otherwise
   (b) a suspends the event related to *op* $C(t)$ and sends a plan request to each agent in its $cs.TrustedAgents$ set. Plan requests contain information on the ontology C belongs to and on a's matching strategy ms. To be more precise, in order to implement the RetrieveExtRelPlans function, we use a "multicast synchronous ask message with a timeout" that has been added to *Jason* as a demand from our work for this paper. 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 [57].
   (c) *op* $C(t)$ remains suspended until a deadline chosen by the MAS designer in the MAS setup expires. When the event is resumed, either some relevant plans have been received by a thanks to the cooperation with other agents, and thus the event can be managed, or no plan has been received, and the event fails.

When an agent r receives an `AskHow` request for dealing with *op* $C(t)$, where $C(t) \in oid$, using matching strategy ms, it performs the following actions.

1. First, it looks for plans whose triggering event op' $D(t')$ is annotated with $o(oid)$. If a plan is found where either D is equal to C, or it is a superconcept of C, $op = op'$, and t and t' unify, the plan is stored (in a list of plans to be sent as reply to the request) since it is relevant for *op* $C(t)$ according to intra-agent ontological reasoning.
2. Then, it calls `concept_match` ⟨`oid`⟩ ⟨`C`⟩ `ms.method ms.threshold`. Let us suppose that $\langle C, D_1, th_1\rangle$, $\langle C, D_2, th_2\rangle$, ..., $\langle C, D_k, th_k\rangle$ are returned by OntArt. Agent r substitutes D_i to C in $C(t)$ and looks for plans relevant for *op* $D_i(t)$ in its own plan base. If it finds such plans, it substitutes in them not only D_i with C, but also the other entities referring to r's ontologies found during this matching stage with their corresponding entities (if any) belonging to a's ontology oid. This conversion step is required because

a would not know that it can use a plan triggered by *op* $D_i(t)$ for coping with *op* $C(t)$, and would not know how to cope with other goals in that plan expressed according to unknown ontologies, thus necessarily originating other cooperation requests. Without this conversion, r's effort would be useless. "Backwards converted" plans are stored together with those found in step 1.
3. Stored relevant plans are sent back to a. Agent a will use them when event *op* $C(t)$ will be resumed.

This behavior is implemented within the CooL-AgentSpeak interpreter and is transparent to agents. 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 `Ag` represents the list of agents Ag in the appropriate *Jason* format):

$\mathsf{RetrieveExtRelPlans}(Te[\mathrm{o}(O), src(S)], Th, Ag) = \mathtt{R}$
iff
`.send(Ag, askHow, Te[o(O), src(S), thr(Th)], R, T).`

Answering an `askHow` message involves calls to OntArt operations, as discussed above.

6. CooL-AgentSpeak at Work

The scenario is inspired by the opening of [5], where T. Berners-Lee, J. Hendler and O. Lassila envision a world crowded by intelligent software agents living in all electronic devices, able to understand messages coming from both their human masters and other agents in the system, and that continuously face problems that require cooperation to be solved.

> *At the doctor's office, Lucy instructed her Semantic Web agent through her handheld Web browser. The agent promptly retrieved information about Mom's prescribed treatment from the doctor's agent, looked up several lists of providers, and checked for the ones in-plan for Mom's insurance within a 20-mile radius of her home and with a rating of excellent or very good on trusted rating services. It then began trying to find a match between available appointment times (supplied by the agents of individual providers through their Web sites) and Pete's and Lucy's busy schedules. In a few minutes the agent presented them with a plan.*

Our "HappyHousewives" scenario[7], is much less serious than the family healthcare problem discussed in [5], but relies on the same assumptions. It provides the reader with a simple and easy-to-follow example, meant to help her understanding how CooL-AgentSpeak can be used in practice. Housewives use their handheld devices to exchange "how-to" suggestions related to their main activities (cooking, housekeeping, kid care), and these suggestions can be directly executed by agents managing physical devices (e.g., ovens, radios, televisions) and by domestic robots that share the same set of actions, expressed according to a standard vocabulary agreed upon by companies selling those devices.

For example, we assume that kitchen robots are able to perform the most common activities required in a kitchen. They are equipped with image recognition capabilities that allow them to take food from the kitchen appliances given the food name (atomic action `take(+FoodName)`, that allows the robot to grasp the object and to add in its belief base the information that it is currently holding it) and to read information on the object they are holding (`read(+PropertyToRead, -ReadValue)`), such as the cooking time (expressed in minutes as usual), and have pre-defined programs for the basic cooking activities such as cooking pasta, given the amount of pasta (expressed in grams) to cook and the cooking time (`cook_pasta(P, T, Amount)`).

On the other hand, we make no assumptions on the higher level vocabulary used by housewives for encoding their "how-to" knowledge: for one of them, the cooking time property of a given course could be expressed by a `takesCookingTime` belief, for another by the slightly different `hasCookingTime` belief. One agent might know just that pasta exists, and another might know many different types of pasta.

Personal agents are equipped with one or more ontologies that formalize their "how-to" knowledge in given domains.

Let us now suppose that personal agent *barbara* uses ontologies `http://krono.act.uji.es/Links/ontologies/food.owl` shown in Figure 2 and `http://daisy.cti.gr/svn/ontologies/AtracoProject/Pasta/Pasta_new_1.owl`

[7]The name seems not very original, according to a Google search we made on November, 2013. The first ten sites returned by the search are very close to the spirit of our scenario. Maybe this means that the time is right for implementing a fully fledged "HappyHousewives" framework with CooL-AgentSpeak!

shown in Figure 3 to represent the food domain, whereas personal agent *alice* uses ontology `http://daisy.cti.gr/svn/ontologies/AtracoProject/Pasta/Pasta_4.owl` shown in Figure 4[8]. We will use `food`, `p1` and `p4` to identify them in the sequel.

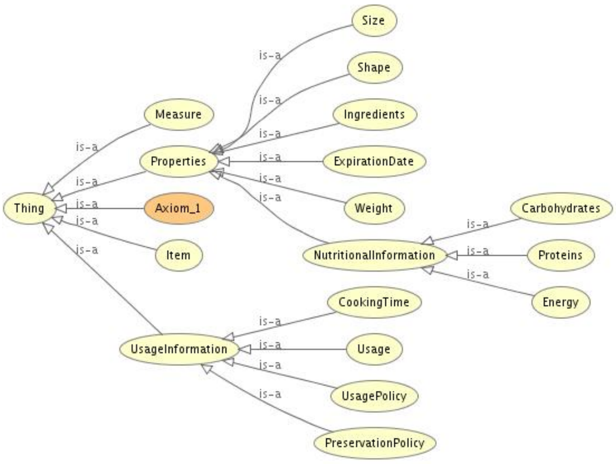

Fig. 3. Ontology `p1` used by *barbara*.

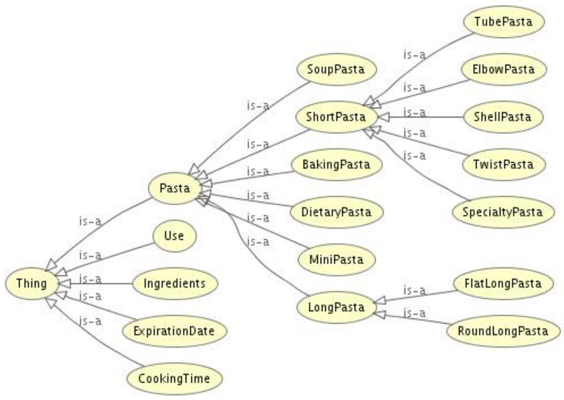

Fig. 4. Ontology `p4` used by *alice*.

Ontology `p4` includes a `takesCookingTime` property not shown in Figure 4 and ontology `p1` includes a `hasCookingTime` property.

alice has the following plan only:

```
+!takesCookingTime(P, T)[o("p4")] :
   holding(P) <-
      read(cooking_time, T).
```

[8] All the ontologies have been accessed on November 2013.

whereas *barbara* has the following one:

```
+!pasta(P)[o("food")] <-
      take(P);
      !hasCookingTime(P, T)[o("p1")];
      cook_pasta(P, T, 50).
```

Agent *alice* receives from her housewife the instruction to have `shortPasta` ready for dinner. `shortPasta` belongs to *alice*'s ontology but *alice* has no relevant plans for the `+!shortPasta(P)` event, nor plans relevant for triggering events involving one of `shortPasta`'s super-classes, such as `pasta`. As a result, she cannot deal with the request.

Since *barbara* is one of *alice*'s trusted agents, the cooperation among the two starts to look for plans in *barbara*'s plan base that might relate to the `+!shortPasta(P)` event.

The search for a plan in *barbara*'s plan library whose triggering event matches `+!shortPasta(P)` succeeds thanks to the Cool-AgentSpeak features. In fact, the correspondence $\langle$shortPasta $\in$ p4, pasta $\in$ food, 0.67$\rangle$ can be easily found by the JWNL ontology matching algorithm provided by the OntArt artifact, hence allowing the retrieval of a relevant plan for `+!pasta(P)` $\in$ food (and hence to `+!shortPasta` $\in$ p4) to succeed.

Nevertheless, sending the plan shown above (with triggering event `+!pasta(P)[o("food")]`) to *alice* would not help her for three reasons:

1. *alice* does not know what `pasta(P)[o("food")]` is because of the annotation that refers to an unknown ontology;
2. *alice* was looking for a plan triggered by `+!shortPasta(P)` and not by `+!pasta(P)`; and
3. she would not be able to execute the plan in that form because of the `!hasCookingTime(P, T)[o("p1")]` goal which raises problems due to both the ontology entity and the annotation.

Before sending the plan to *alice*, *barbara* changes all the concepts annotated with `food` or `p1` with the corresponding entity (if any) in `p4`, which is the ontology labeling the trigger that originated the plan search. This behavior is transparent to *barbara* as explained in Section 5. In this example, besides substituting

```
+!pasta(P)[o("food")]
```

with

```
+!shortPasta(P)[o("p4")]
```

in the trigger, *barbara* also substitutes

```
!hasCookingTime(P, T)[o("p1")]
```

Fig. 2. A portion of the ontology `food` used by $barbara$.

with

`!takesCookingTime(P, T)[o("p4")],`

in the body, obtained thanks to the mapping ⟨hasCookingTime ∈ p1, takesCookingTime ∈ p4, 0.8⟩ found by OntArt.

The plan sent back to $alice$ is then:

```
+!shortPasta(P)[o("p4")]  <-
   take(P);
   !takesCookingTime(P, T)[o("p4")];
   cook_pasta(P, T, 50).
```

that $alice$ can execute without needing further interactions.

Since `penne` is an instance of `shortPasta` in ontology "p4", $alice$ has the `shortPasta(penne)` [o("p4")] belief in her belief base. The trace we obtain by simulating external actions with a print action and using fixed values for the action parameters, is:

```
[alice] take(penne)
[alice] read(cooking_time, 8)
[alice] cook_pasta(penne, 8, 50)
```

7. Experiments

In the previous section we discussed a simple scenario, expressly designed to show the potential of CooL-AgentSpeak in a clear and easy way.

The empirical evaluation of CooL-AgentSpeak has been carried out on three complex scenarios in the biomedicine, enterprise document organization, and finance domains.

Scenario 1: Biomedicine

In this scenario, agents FMA and NCI both operate in the field of anatomy, but while FMA organizes its knowledge according to the Foundational Model of Anatomy[9], NCI reasons according to the National Cancer Institute Thesaurus[10].

The Foundational Model of Anatomy is a project of the Structural Informatics Group at the University of Washington. It has been under development since 1995 and the current version of the ontology includes 75,000 anatomical classes and 174 properties. The FMA ontology represents anatomical entities from a very fine granularity such as the biological molecules to cells, tissues, organs, organ systems, major body parts, up to the entire body.

The NCI Thesaurus is an ontology-like vocabulary that includes broad coverage of the cancer domain, including cancer related diseases, findings and abnor-

[9]http://sig.biostr.washington.edu/projects/fm/

[10]http://ncit.nci.nih.gov/

malities; anatomy; agents, drugs and chemicals; genes and gene products and so on. In certain areas, like cancer diseases and combination chemotherapies, it provides the most granular and consistent terminology available. The NCI Thesaurus currently contains over 34,000 concepts, structured into 20 taxonomic trees.

The two ontologies are semantically rich and contain tens of thousands of classes. For our experiments, we limited ourselves to a significant fragment of both[11] , whose quantitative descriptors (named classes, anonymous classes, properties, and dimension) are reported below in Table 1.

	Named classes	Anony. classes	Prop.	Dim (KB)
ont1 (fma agent)	3,696	30	0	2,000
ont2 (nci agent)	6,488	5,141	63	4,600

Table 1
Scenario 1: Quantitative descriptors of the ontologies.

The FMA agent operates on behalf of its human user by retrieving sources of information dealing with anatomical concepts represented according to the FMA ontology[12]. NCI is one of FMA's trusted agents and provides plans to retrieve sources of information on concepts represented according to the NCI ontology. FMA possesses only a subset of the plans that it would need to satisfy its user's requests: as shown in Section 7.1, without heavily exploiting the CooL-AgentSpeak features, FMA would not be able to achieve its goals.

Scenario 2: Enterprise Content Management

This scenario involves one real ontology developed during the "EC2M system (Enterprise Cloud Content Management)" Programma Operativo Regionale (POR) project funded by the Ligury region [8], and one artificial ontology obtained by modifying the original one for the purpose of running our experiments.

The EC2M project involved one of the authors from the Department of Informatics, Bioengineering, Robotics and System Engineering of the University of Genoa, Sempla[13], Nacon[14], and other partners from both academia and industry. It aimed at creating an improved Enterprise Content Management ECM system named "EC2M" exploiting ontologies to better classify, retrieve and share documentation among the different sites of Sempla. The ontology that has been created to model Sempla's business offers is actually used by Sempla and can be considered a good representative of ontologies for enterprise document classification.

In this scenario we moved a step further with respect to the EC2M project's goals by analyzing how ontology matching techniques could ease semantic interoperability among the different sites of Sempla, by allowing different sites to use slightly different ontologies.

We built an ontology starting from the real one introducing small variations in both the concepts' and properties' names, and we developed a scenario where agent Sempla1 uses the original ontology, and agent Sempla2 uses the modified one. As shown in Table 2, the descriptors of the ontologies are very similar since we only modified the names of some elements.

	Named classes	Anony. classes	Prop.	Dim (KB)
ont1 (sempla1 agent)	39	26	31	123.9
ont2 (sempla2 agent)	39	26	31	123.4

Table 2
Scenario 2: Quantitative descriptors of the ontologies.

As in the previous scenario, agent Sempla1 must retrieve content for the users of the site where it resides, but has not enough procedural knowledge to do that. It trusts Sempla2 agent and, thanks to the CooL-AgentSpeak features and to the knowledge possessed by Sempla2, its objectives can be achieved.

Scenario 3: Finance

The third scenario is inspired by the financial domain and exploits two ontologies belonging to the

[11] `http://www.cs.ox.ac.uk/isg/projects/SEALS/oaei/2012/LargeBioMed_dataset_oaei2012.zip`, files `oaei2012_NCI_small_overlapping_fma.owl` and `oaei2012_FMA_small_overlapping_nci.owl`.

[12] Modeling the information sources in a realistic way was out of the scope of this experiment, and we limited ourselves to represent them as strings.

[13] Sempla, `http://www.sempla.it/`, is an Italian company working in the areas of business services and IT consulting, program management, digital design, process and system design, package implementation and custom development, right/downsizing and outsourcing services. It mainly operates in the Financial Service, Industry, Public Administration, and Utilities and Energy markets.

[14] Nacon, `http://www.nacon.it/nacon/`, is a software house based in Genova, Italy, that just entered the Sempla group. Its main competencies are in the Financial and Bank markets.

Ontology Alignment Evaluation Initiative Benchmark (OAEI, `http://oaei.ontologymatching.org/`). OAEI is a coordinated international initiative to forge the consensus on evaluating ontology matching methods. Its first edition dates back to 2004, and it has been run at least yearly since then.

The two ontologies we used are the reference onto1 ontology for the finance data set (`http://oaei.ontologymatching.org/2012/benchmarks/tests-finance.zip`), and ontology 223 from the same data set, where numerous intermediate classes are introduced within the hierarchy w.r.t. the reference one. The updated Finance ontology is today a federation of ontologies where the main ontology is found at `http://fadyart.com/Finance.owl`; the two ontologies we used in our experiments, whose quantitative descriptors are given in Table 3, are simplified versions of the actual one.

	Named classes	**Anony. classes**	**Prop.**	**Dim (KB)**
ont1 (finance1 agent)	322	131	247	2,000
ont2 (finance2 agent)	644	131	247	2,100

Table 3
Scenario 3: Quantitative descriptors of the ontologies.

Also in this scenario, agents Finance1 and Finance2 model their knowledge according to ontologies onto1 and onto223 respectively, and again Finance1 lacks some procedural knowledge that would be necessary for completing its tasks, and that will be provided by Finance2 thanks to the CooL-AgentSpeak cooperation and ontology matching mechanisms.

Significance of the selected scenarios

We are aware of just a few works attempting to carry out a statistical study on existing ontologies [4,36,55,60,61]. One of the most recent ones [37] describes the results of analyzing the first 300 OWL ontologies returned by the query `filetype:owl` with Google. These results can be summarized in the following way:

- **Language.** 83.3% ontologies are expressed in English.
- **File size.** The average size for the OWL file is 204.26 KB. The standard deviation and variance are high, so the dispersion is high. The most repeated size in the sample is a file of 5 KB.
- **Number of entities.** The average number of classes is 384.73 and of properties is 67.98. The maximum number of classes is 23,141 and of properties is 1,507.

As far as the ontology domains are concerned, we are not aware of extensive studies on this subject. Anyway, by looking at existing libraries of ontologies[15], we can notice that the biological, medical and anatomical domains are the most widespread ones. The enterprise domain is covered by a couple of ontologies and other ontologies' domains range from business to finance, from territories to tourism, from cinema to food.

With respect to these considerations, the six ontologies that we selected for our experiments, although too few to represent a statistically relevant sample, are significant examples of different existing domains and of different features in terms of ontology properties (file size and number of entities). The choice of ontologies expressed in English is motivated by high percentage of real ontologies that are defined in this language.

7.1. Experiments

For each scenario and for each matching algorithm used within that scenario, we designed and implemented the two agents involved in the MAS following always the same schema. In this section we use the first scenario with AROMA as ontology matcher as our running example.

The first agent in the MAS (FMA, whose code is shown in Figure 5, in our running example) has an initial `+!start` goal consisting of achievement subgoals that involve concepts from ontology ont1 (the FMA ontology in our running example). Each subgoal is repeated twice, to allow us to verify the correct behavior when relevant plans are not available locally, and the *add* acquisition policy and *noLocal* retrieval policy are used. Among the subgoals,

– six (three different ones, repeated twice) involve concepts in ont1 for which the used ontology matching algorithm (AROMA, in the running example) can find corresponding concepts in ont2, and for which no local relevant plans exist, but plans in the trusted agent's

[15] `http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library#OWL_ontologies`; `http://en.wikipedia.org/wiki/Ontology_(information_science)#Examples_of_published_ontologies`; `swoogle.umbc.edu/`; all accessed on November 2013.

code can be found (we tag these subgoals with the label OK MATCH; NO LOCAL; OK TRUSTED); these subgoals could not be achieved without exploiting the CooL-AgentSpeak cooperation and matching features;

– four involve concepts in ont1 for which the used ontology matching algorithm can find corresponding concepts in ont2, and for which both local relevant plans and plans in the trusted agent's code exist (OK MATCH; OK LOCAL; OK TRUSTED);

– four involve concepts in ont1 for which the used ontology matching algorithm can find corresponding concepts in ont2, and for which local relevant plans exist, but no plan in the trusted agent's code can be found (OK MATCH; OK LOCAL; NO TRUSTED);

– six involve concepts in ont1 for which the used ontology matching algorithm cannot find any corresponding concept in ont2, and for which local relevant plans exist, but no plan in the trusted agent's code can be found (NO MATCH; OK LOCAL; NO TRUSTED).

The second agent (NCI, whose code is shown in Figure 6, in the running example) provides those plans that should be found in the trusted agent's code, characterized by a triggering event expressed using concepts from ontology ont2 (the NCI ontology).

Table 4 shows the correspondences that we exploited in our running example. By subgoal of type 1 (**sg** column in the table) we mean those tagged by "OK MATCH; NO LOCAL; OK TRUSTED" in Figure 5; subgoals of type 2 are the "OK MATCH; OK LOCAL; OK TRUSTED" ones; subgoals of type 3 are the "OK MATCH; OK LOCAL; NO TRUSTED" ones.

sg	**concept in ont1 (FMA)**	**concept in ont2 (NCI)**
1	endothelium_of_arteriole	arteriole_Endothelium
1	urethral_gland	urethra_Gland_MMHCC
1	tarsal_plate_of_eyelid	tarsal_Plate
2	root_of_tooth	radix_Dentis
2	epithelium_of_ciliary_body	ciliary_Epithelium
3	splenic_lymph_node	splenic_Hilar_Lymph_Node
3	internal_thoracic_vein	internal_Thoracic_Vein

Table 4

Correspondences exploited in our running example.

7.2. *Results*

As indicators for measuring how good an alignment is, we use precision, recall and F-measure adapted for ontology alignment evaluation [17].

Precision is defined as the number of correctly found correspondences with respect to a reference alignment (true positives) divided by the total number of found correspondences (true positives and false positives) and recall is defined as the number of correctly found correspondences (true positives) divided by the total number of expected correspondences (true positives and false negatives). To compute precision and recall, the alignment A returned by the algorithm is compared to a reference alignment R. Precision is given by the formula $P(A,R) = \frac{|R \cap A|}{|A|}$ whereas recall is defined as $R(A,R) = \frac{|R \cap A|}{|R|}$. We also use the harmonic mean of precision and recall, namely F-measure: $F(A,R) = 2 \cdot \frac{P(A,R) \cdot R(A,R)}{P(A,R)+R(A,R)}$.

Table 5 shows the results we obtained when aligning the first ontology (FMA; sempla-mod; onto1) and the second ontology (NCI; sempla; onto223) with JWNL and AROMA in our three scenarios. We used the reference alignments available in `http://www.cs.ox.ac.uk/isg/projects/SEALS/oaei/2012/LargeBioMed_dataset_oaei2012.zip` and `http://oaei.ontologymatching.org/2012/benchmarks/tests-finance.zip` for scenarios 1 and 3 respectively. We used a reference alignment built by ourselves for scenario 2.

	Precision	**Recall**	**F-measure**
Scen. 1, JWNL	0.88	0.75	0.81
Scen. 1, AROMA	0.50	0.61	0.55
Scen. 2, JWNL	0.25	0.79	0.39
Scen. 2, AROMA	0.78	0.78	0.78
Scen. 3, JWNL	0.36	1.00	0.53
Scen. 3, AROMA	1.00	1.00	1.00

Table 5

Quantitative measures of the matching algorithms performances.

CooL-AgentSpeak computes both the alignment between the first and the second ontology, and the one between the second and the first, because the ontology matchers we used in our experiments are asymmetric and looking for correspondences in both directions gives us more chances to find them. Computing the alignments between the two ontologies involved in the MAS is the most time-consuming activity, but once computed, the alignment can be stored for being used in successive runs. For this reason we computed the ontology matching execution time and the MAS execution time when a pre-computed alignment is used separately. The time required for running the MAS when the alignment must be computed from scratch, is the sum of these two values.

Table 6 reports the ontology matching execution time (**Total matching time**, amounting to the sum of the time for computing the alignments in both directions), the dimension of the resulting alignments (**Dim. ont1-ont2** and **Dim. ont2-ont1**), as well as their sum (**Total dim.**). **S***X* stands for "scenario *X*", **J** stands for "using JWNL matching method" and **A** stands for "using AROMA matching method".

	Total matching time	**Dim. ont1-ont2**	**Dim. ont2-ont1**	**Total dim.**
S1, J	18,272,462 (>5 h)	1.3 MB	2.4 MB	3.7 MB
S1, A	205,738 (>3 m)	998 KB	994 KB	~2 MB
S2, J	19,333 (~19 s)	75 KB	75 KB	150 KB
S2, A	2,675 (~2 s)	26 KB	26 KB	52 KB
S3, J	1,400,930 (>23 m)	793 KB	886 KB	~1.7 MB
S3, A	12,996 (~13 s)	259 KB	259 KB	518 KB

Table 6

Execution time in milliseconds of the ontology matching algorithms and dimension of the resulting alignments.

As an indicator of the efficiency of our system, we used the time required by FMA, Sempla1 and Finance1 to achieve their `+!start` goal.

We run our experiments by combining the two retrieval policies *always* and *noLocal* with the two acquisition policies *add* and *replace*. We verified that each combination behaved as expected by manually inspecting the messages printed on the Jason console.

For example, the *always+add* combination in scenario 1 leads to the following messages

```
[fma] Agent fma starting...
[cool] Searching relevant plans for event
+!endothelium_of_arteriole
Starting CooL-AgentSpeak strategy
[fma] Using nci plan for arteriole_Endothelium
[cool] Searching relevant plans for event
+!endothelium_of_arteriole
Starting CooL-AgentSpeak strategy
[fma] Using nci plan for arteriole_Endothelium
```

showing that the external relevant plan for dealing with `!endothelium_of_arteriole` is retrieved two times, even if it is added to the local plan library after the first time, because of the *always* policy.
The *noLocal+add* combination leads to

```
[fma] Agent fma starting...
[cool] Searching relevant plans for event
+!endothelium_of_arteriole
Starting CooL-AgentSpeak strategy
[fma] Using nci plan for arteriole_Endothelium
[fma] Using nci plan for arteriole_Endothelium
```

where the second call to subgoal `!endothelium_of_arteriole` does not generate any CooL-AgentSpeak interaction because the subgoal can be properly dealt with using the plan that NCI sent after the first call, and that FMA added to its plan library.

Finally, the *always+replace* combination for coping with subgoal `!root_of_tooth` leads to the following messages

```
[cool] Searching relevant plans for event
+!root_of_tooth
Starting CooL-AgentSpeak strategy
[fma] Using nci plan for radix_Dentis
```

showing that the plan sent by NCI for dealing with `+!radix_Dentis` has been used instead of the plan locally available to FMA for dealing with `+!root_of_tooth`, because of the *replace* policy.

In the sequel we indicate FMA, Sempla1 and Finance1 (in scenario 1, 2 and 3, respectively) with "first agent", and NCI, Sempla2 and Finance2 (in scenario 1, 2 and 3, respectively) with "second agent" or "trusted agent".

Table 7 shows the time required by the first agent to obtain a relevant plan for a triggering event *te1* from the second agent in the MAS, in case the second agent possesses a plan whose triggering event is *te2*, and *te1* and *te2* correspond according to the selected matching method. For each scenario and matching method we computed the average time on ten experiment runs using both the *add* and the *replace* acquisition policies, concluding that the policy does not impact on the execution time, but the dimension of the alignment does.

S1, J	**S1, A**	**S2, J**	**S2, A**	**S3, J**	**S3, A**
122 ms	86 ms	75 ms	48 ms	81 ms	57 ms

Table 7

Average execution time in milliseconds for obtaining and adding/replacing a relevant plan from a trusted agent.

Tables 8, 9, 10, report the execution time required to achieve the `+!start` goal of the first agent, under different configurations and using different matching methods. The threshold for considering a correspondence acceptable was set to 0.6 for all the experiments.

As far as the retrieval policies are concerned, **noLoc** stands for *noLocal* and **alw** stands for *always*. The acquisition policies may be **add** and **rep** (*replace*). When the noLocal retrieval policy is used, we set the timeout to 2000 milliseconds. When the always retrieval policy is used, we run the experiments using different timeouts: 50, 200, 2000, and 4000 milliseconds.

Our experiments have been designed in such a way that for those subgoals that cannot be handled locally, one relevant plan is available in the trusted agent's plan library. Hence, when the *noLocal* strategy is used, we are sure that a relevant plan coming from the trusted agent will be obtained, and there will be no need to wait for the timeout to expire. In this case, the execution time does not depend on the timeout.

On the other hand, when we use the *always* strategy, the first agent will ask for plans that the second agent could not possess. In this situation the second agent does not answer to the first one, which must wait for the timeout to expire before continuing the plan's execution looking for local plans. Changing the timeout, we clearly obtain different execution times. By running 20 experiments under different conditions and scenarios, we concluded that given *Timeout* the current timeout, the average time required for asking for a plan to the trusted agent, waiting for the timeout to expire, and using the local plan, amounts to *Timeout* + *T_external*, with *T_external* = 10 milliseconds. The average time required to satisfy one achievement goal using one local plan, without any cooperative interaction is *T_local* = 2 milliseconds.

Given

– *T_successful* the time required to retrieve a plan available in the trusted agent's plan library, shown in Table 7,

– *T_external* 10 milliseconds

– *T_local* 2 milliseconds

– *S_successful* the number of subgoals for which a retrieval of an external relevant plan would be needed, and the retrieval succeeds

– *S_failing* the number of subgoals for which a retrieval of an external relevant plan would be needed, and the retrieval fails

– *S_local* the number of subgoals for which no external retrieval is required

– *Timeout* the timeout of the current experiment

we conjectured that the total execution time should be given by the following formula:

(**Equation 1**)
Total_execution_time =
S_successful * *T_successful* +
S_failing * (*Timeout* + *T_external*) +
S_local * *T_local*

In the experiments with *noLocal* retrieval policy, *S_successful* = 3; *S_failing* = 0; *S_local* = 17. In the experiments with *always* retrieval policy, *S_successful* = 10; *S_failing* = 10; *S_local* = 0.

Tables 8, 9, 10 report the measured time in milliseconds (**Min**umum, **Med**ian and **Max**imum on 5 experiments for each configuration), the time expected according to Equation 1 (**Exp**), and the difference between the expected time and the median measured time (**Diff**). We indicate with "fail" those configurations where the timeout expired before the relevant plan was sent by the trusted agent, hence leading to a failure of the plan of the first agent, at least in one of the 5 experiments we carried out.

We run our experiments on an Acer TravelMate 6293 Notebook equipped with Intel Core Duo Processor P8400, 2GB of RAM, and Mandriva Linux as operating system.

7.3. Discussion

The results of our experiments allowed us to draw the following conclusions.

AROMA is more advisable than JWNL both for precision/recall and efficiency. Tables 5 and 6 do not require extensive comments and throw no surprise. As stated in the Alignment API home page, "The Alignment API [...] is not a matcher. A few examples of trivial matchers are provided with the Alignment API which will indeed match ontologies." We integrated the algorithms provided by the Alignment API into CooL-AgentSpeak because we preferred to give the opportunity to the MAS developer to make a choice among more possibilities, rather than imposing one. The case study discussed in Section 6 uses JWNL and it works in a satisfactory way, showing that JWNL can be used in practice, but the experiments we run on more complex ontologies demonstrate that in most cases AROMA is definitely preferable to JWNL.

The time required for obtaining a relevant plan from a trusted agent that possesses it, depends on the dimension of the alignment. Tables 6 and 7 show a close relationship between the average execution time for obtaining and adding/replacing a relevant plan from a trusted agent, and the dimension of

JWNL	Min	Med	Max	Exp	Diff	AROMA	Min	Med	Max	Exp	Diff
noLoc, add, 2000	351	360	371	400	40	**noLoc, add, 2000**	320	337	387	292	-45
noLoc, rep, 2000	332	362	397	400	38	**noLoc, rep, 2000**	282	307	374	292	-15
alw, add, 50	fail	fail	fail	1,790	–	**alw, add, 50**	fail	fail	fail	1,430	–
alw, rep, 50	fail	fail	fail	1,790	–	**alw, rep, 50**	fail	fail	fail	1,430	–
alw, add, 200	3,168	3,197	3,248	3,290	93	**alw, add, 200**	2,937	2,989	3,053	2,930	-59
alw, rep, 200	3,166	3,172	3,199	3,290	118	**alw, rep, 200**	2,859	2,930	3,071	2,930	0
alw, add, 2000	21,197	21,685	21,742	21,290	-395	**alw, add, 2000**	20,929	20,952	20,977	20,930	-22
alw, rep, 2000	21,184	21,233	21,453	21,290	57	**alw, rep, 2000**	20,913	20,978	21,021	20,930	-48
alw, add, 4000	41,134	41,222	41,327	41,290	68	**alw, add, 4000**	40,922	40,969	41,053	40,930	-39
alw, rep, 4000	41,198	41,225	41,439	41,290	65	**alw, rep, 4000**	40,995	41,006	41,071	40,930	-76

Table 8

Scenario 1 (biomedical domain, large ontologies): execution time.

JWNL	Min	Med	Max	Exp	Diff	AROMA	Min	Med	Max	Exp	Diff
noLoc, add, 2000	218	223	227	259	36	**noLoc, add, 2000**	172	184	192	178	-6
noLoc, rep, 2000	206	211	224	259	48	**noLoc, rep, 2000**	181	188	195	178	-10
alw, add, 50	1,080	1,126	1,215	1,320	194	**alw, add, 50**	1,030	1,047	1,068	1,050	3
alw, rep, 50	1,099	1,129	1,144	1,320	191	**alw, rep, 50**	1,031	1,064	1,103	1,050	-14
alw, add, 200	2,616	2,624	2,653	2,820	196	**alw, add, 200**	2,602	2,610	2,625	2,550	-60
alw, rep, 200	2,627	2,649	2,710	2,820	171	**alw, rep, 200**	2,593	2,604	2,616	2,550	-54
alw, add, 2000	20,698	20,721	20,832	20,820	99	**alw, add, 2000**	20,632	20,645	20,650	20,550	-95
alw, rep, 2000	20,680	20,708	20,794	20,820	112	**alw, rep, 2000**	20,630	20,645	20,662	20,550	-95
alw, add, 4000	40,606	40,642	40,718	40,820	178	**alw, add, 4000**	40,590	40,678	40,830	40,550	-128
alw, rep, 4000	40,555	40,612	40,720	40,820	208	**alw, rep, 4000**	40,618	40,644	40,679	40,550	-94

Table 9

Scenario 2 (enterprise content management domain, small ontologies): execution time.

JWNL	Min	Med	Max	Exp	Diff	AROMA	Min	Med	Max	Exp	Diff
noLoc, add, 2000	275	286	292	277	-9	**noLoc, add, 2000**	201	230	263	205	-25
noLoc, rep, 2000	269	272	283	277	5	**noLoc, rep, 2000**	209	224	235	205	-19
alw, add, 50	fail	fail	fail	1,380	–	**alw, add, 50**	1,133	1,149	1,177	1,140	-9
alw, rep, 50	1,296	1,333	1,351	1,380	–	**alw, rep, 50**	1,132	1,143	1,160	1,140	-3
alw, add, 200	2,855	2,874	2,904	2,880	6	**alw, add, 200**	2,603	2,620	2,651	2,640	20
alw, rep, 200	2,812	2,875	2,923	2,880	5	**alw, rep, 200**	2,707	2,715	2,736	2,640	-75
alw, add, 2000	20,776	20,871	20,913	20,880	9	**alw, add, 2000**	20,689	20,712	20,744	20,640	-72
alw, rep, 2000	20,901	20,924	20,954	20,880	-44	**alw, rep, 2000**	20,720	20,754	20,761	20,640	-114
alw, add, 4000	40,856	40,860	40,912	40,880	20	**alw, add, 4000**	40,670	40,765	40,900	40,640	-125
alw, rep, 4000	40,819	40,858	40,872	40,880	22	**alw, rep, 4000**	40,634	40,686	40,704	40,640	-46

Table 10

Scenario 3 (financial domain, medium ontologies): execution time.

the alignment between the ontologies used by the two agents. Since the second agent must inspect the alignment in order to search for a correspondence, the larger the alignment, the higher the time required for the search. The maximum values are obtained in scenario 1 with JWNL, where the total dimension of the alignment is 3.7 MB and the time is 122 ms, whereas the minimum values are obtained in scenario

2 with AROMA, where the total dimension of the alignment is 52 KB and the time is 48 ms.

There is no difference in execution time between adding and replacing a plan. Tables from 8 to 10 show that, at least in situations similar to those of our experiments, using the *add* or the *replace* acquisition policy has no impact on the execution time. This was a bit surprising since we expected that replacing a plan would be more time-consuming than just adding it, but we had no empirical validation of our expectation.

The noLocal retrieval policy is far more efficient than the always one. This observation was easily foreseeable: looking for external plans whenever a goal must be achieved is definitely more time consuming than looking for external plans only when no local plans can be used. This suggests that, unless required by the application, the *noLocal* policy should be used instead of the *always* one.

Equation 1 is correct, hence the total execution time can be predicted in advance. Tables from 8 to 10 demonstrate that Equation 1 is correct: the difference between the expected execution time and the measured execution time (taking the median of 5 experiments as reference value) is some tens of milliseconds in most experiments, and does not exceed 395 milliseconds. These discrepancies are physiological: measurement errors of a few milliseconds easily justify them.

CooL-AgentSpeak can be exploited in all those scenarios where no hard real-time constraints must be met. The obtained results show that, although not suitable for scenarios with hard real-time constraints, CooL-AgentSpeak using the AROMA matching method can be used in practice whenever the agents (or their human owners) can wait for a few seconds to get the answer to their request. In scenario 1 using AROMA, setting the timeout to 200 milliseconds is enough to be sure to get an answer in about 3 seconds even when the time-consuming *always* retrieval policy is used. Considering the dimension of the ontologies involved in that scenario, and the advantage that a software or human agent could gain by obtaining an answer instead of a goal failure, the price to be paid seems widely acceptable.

8. Related Work

The literature describing the integration of concepts coming from the Semantic Web into agent-oriented engineering artifacts (methodologies, models, and languages) is recent and rather limited. In particular, the one dealing with the integration of ontology services into agent-oriented programming languages amounts to a few proposals, besides those already discussed in Section 2.2. In this section we briefly overview the existing literature and compare works on the integration of ontologies into agent-oriented programming languages to CooL-AgentSpeak.

8.1. Methodologies

In [56], an ontology-based methodology called MOBMAS is presented with the aim to support the analysis and design of multi-agent systems. MOBMAS is the first methodology that explicitly identifies and implements the various ways in which ontologies can be used in the MAS development process and can be integrated into the MAS model definitions.

The authors of [32] focus on Model Driven Development (MDD) and propose a model transformation process for MDD of Semantic Web enabled MASs.

When support for Semantic Web technology and its related constructs are considered at the meta-level, agent meta-models should include meta-entities to model MASs which work in the Semantic Web environment. In [33], an agent meta-model is proposed to define the required constructs of a Semantic Web enabled MAS in order to provide semantic capability modeling and interaction of agents both with other agents and semantic web services.

The recent work presented in [31] extends MaSE [14] by integrating early requirement specification and ontology concepts into its standard flow, in order to define the type of the objects used in MaSE diagrams.

8.2. Models

Among the most recent proposals of exploiting Semantic Web technologies within organizations and institutions, we may cite [54] where agents dynamically manage the interdependencies that arise during their interactions thanks to an ontological approach to coordination. A framework for the rapid development of organizational simulations, OOS, is introduced in [15]. It provides a structured and efficient way to deploy many different organizational designs, by using an ontology to describe organization structures, environment characteristics and agent capabilities, and provides semi-automatic means to generate simulations from the ontology instances. Domain ontologies are exploited for regulating institutions as normative systems in [26], where institutions use ontologies to relate the abstract concepts in which their norms are formulated, to their concrete application domain. The integration of ontologies within institutions in order to establish the acceptable illocutions, and of a dialogic framework defining the participant roles in the institution and the relationships among them, is discussed in [18]. The proposal made in [34] goes even further, since it assumes no design-time ontological alignment of the agents.

8.3. Languages

Recent work by C. Fuzitaki, Á. Moreira, and R. Vieira [24] stems from [46] and proposes the core of a logic agent-oriented programming language based on DL-Lite [10]. With respect to [46], 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 CooL-AgentSpeak has been implemented.

In [12] and [11], 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 [11]. A `Go!` agent typically comprises several threads that implement different aspects of the agent's behavior 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, their work mainly aims at defining a mapping between OWL-Lite constructs and labeled theories in the `Go!` language, losing references to the external ontologies which define the agents' vocabulary. Conversely, our work implicitly assumes that ontologies exist outside the agents' "minds" and makes explicit the references to external ontologies so as to realize semantic integration among agents as envisioned by the work on the Semantic Web, through ontologies made available on the web.

Neither [24] nor [11,12] take ontology matching into account as a means for inferring "cross-ontological knowledge" and none of them consider "cross-ontological reasoning" for exchanging behavioral knowledge.

9. Conclusions and Future Work

To the best of our knowledge, CooL-AgentSpeak represents the first attempt to seamlessly integrate "cross-ontological" reasoning into an agent-oriented programming language. This feature proves useful in all those applications where agents modeling their knowledge according to different ontologies must interoperate sharing not only beliefs but also behavioral knowledge, as exemplified in the scenarios discussed in Sections 6 and 7.

Of course, "cross-ontological" knowledge and reasoning may lead to unwanted behavior. Even those correspondences of maximum confidence might be semantically wrong and this might cause a wrong match to be used with possible disastrous consequences. 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 neighboring concepts in the ontology, to the comments that label the concept itself, and to other contextual information [40]. Hence, these matching algorithms will not return the correspondence $\langle bank, bank, 1\rangle$ if they realize 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 lower than 100% even for the best performing algorithms[16]. In order to cope with this intrinsic limitation of ontology matching techniques available today, we will extend the agents' strategies in order to tag some events as sensitive, and avoid using ontology matching techniques and/or to retrieve external plans for dealing with them. Another research direction we are pursuing to cope with these limitations is the improvement of two matching algorithms proposed by the authors – [40], based on natural language processing techniques and on interpretation of conjunctions, disjunctions and negations appearing in the concepts names as boolean operators, and [41], exploiting upper ontologies as bridges between the ontologies to match – and their integration among the matching methods offered by the Ontology Artifact.

The extension with the Ontology Artifact with control access policies is part of our close future work and is aimed at supporting those scenarios mentioned in the Introduction, where agents do not want to share their ontologies with others.

Finally, the exploitation of CooL-AgentSpeak in real scenarios such as the one faced by the MUSE project [7] will demonstrate its applicability outside the boundaries of academia. MUSE ("MUltilinguality and SEmantics for the Citizens of the World") addresses some of the challenges raised by multilinguality in the Public Administration by exploiting domain ontologies within a MAS, and speech to text, text to speech, and machine translation techniques. Procedural rules describing what a citizen must do to face different situations (identity card first issue, identity card renewal for personal data change, renewal for address change, renewal for loss or theft, just to make some examples) are currently represented as plans in "plain" Jason, even if their triggering event is already in a one-to-one correspondence with concepts in the domain ontology. Explicitly adding ontological information to them and exploiting CooL-AgentSpeak features would allow different Municipality's Registry Offices to exchange their procedural rules that – although being basically the same, due to their compliance to the current regulations – often differ in some minor details that make them difficult to share and compare. MUSE will be experimented in the Registry Office of Genoa Municipality, and its extension with CooL-AgentSpeak features is one of the forthcoming planned activities.

[16]See for example the results of the last Ontology Alignment Evaluation Initiative, `http://oaei.ontologymatching.org/2011/results/index.html`, accessed on November 2013.

References

[1] D. Ancona and V. Mascardi. Coo-BDI: Extending the BDI model with cooperativity. In J. A. Leite, A. Omicini, L. Sterling, and P. Torroni, editors, *Proceedings of DALT 2003*, volume 2990 of *LNCS*, pages 109–134. Springer, 2004.

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

[3] F. Baader and W. Nutt. Basic description logics. In F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors, *Description Logic Handbook*, pages 43–95. Cambridge University Press, 2003.

[4] S. Bechhofer and R. Volz. Patching syntax in owl ontologies. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proceedings of ISWC 2004*, volume 3298 of *LNCS*, pages 668–682. Springer, 2004.

[5] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[6] R. H. Bordini and Á. F. Moreira. Proving BDI properties of agent-oriented programming languages. *Annals of Mathematics and Artificial Intelligence*, 42(1-3):197–226, 2004.

[7] M. Bozzano, D. Briola, D. Leone, A. Locoro, L. Marasso, and V. Mascardi. MUSE: MUltilinguality and SEmantics for the citizens of the world. In *Proceedings of IDC 2012*. Springer, 2012.

[8] D. Briola, A. Amicone, and D. Laudisa. Ontologies in industrial enterprise content management systems: the EC2M project. In J. Lloret, editor, *Proceedings of COGNITIVE 2013*, pages 153–160. IARIA, 2013.

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

[10] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: tractable description logics for ontologies. In *Proceedings of AAAI 2005*, pages 602–607. AAAI Press, 2005.

[11] K. L. Clark and F. G. McCabe. Go! a multi-paradigm programming language for implementing multi-threaded agents. *Annals of Mathematics and Artificial Intelligence*, 41:171–206, 2004.

[12] K. L. Clark and F. G. McCabe. Ontology schema for an agent belief store. *International Journal of Human-Computer Studies*, 65:640–658, July 2007.

[13] J. David, F. Guillet, and H. Briand. Matching directories and OWL ontologies with AROMA. In P. S. Yu, V. J. Tsotras, E. A.

Fox, and B. Liu, editors, *Proceedings of CIKM 2006*, pages 830–831. ACM, 2006.

[14] S. A. DeLoach. Multiagent systems engineering: A methodology and language for designing agent systems. In *Proceedings of AOIS'99*, 1999.

[15] V. Dignum. Ontology support for agent-based simulation of organizations. *Multiagent and Grid Systems*, 6(2):191–208, 2010.

[16] A. L. Dimeas and N. D. Hatziargyriou. A multi-agent system for microgrids. In G. A. Vouros and T. Panayiotopoulos, editors, *Proceedings of SETN 2004*, volume 3025 of *LNCS*, pages 447–455. Springer, 2004.

[17] H. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In A. B. Chaudhri, M. Jeckle, E. Rahm, and R. Unland, editors, *Proceedings of NODe 2002*, pages 221–237. Springer, 2002.

[18] M. Esteva, D. de la Cruz, and C. Sierra. ISLANDER: an electronic institutions editor. In *Proceedings of AAMAS 2002*, pages 1045–1052. ACM, 2002.

[19] J. Euzenat. *Alignment API and Alignment Server*, 2013. Online, accessed on November 2013. `http://alignapi.gforge.inria.fr/`.

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

[21] Foundation for Intelligent Physical Agents. Fipa ontology service specification. Online, accessed on November 2013. `http://www.fipa.org/specs/fipa00086/XC00086D.pdf`, 2001.

[22] Foundation for Intelligent Physical Agents. FIPA ACL message structure specification. Online, accessed on November 2013. `http://www.fipa.org/specs/fipa00061/SC00061G.pdf`, 2002.

[23] J. Fowler, B. Perry, M. Nodine, and B. Bargmeyer. Agent-based semantic interoperability in InfoSleuth. *SIGMOD Rec.*, 28(1):60–67, Mar. 1999.

[24] C. Fuzitaki, Á. Moreira, and R. Vieira. Ontology reasoning in agent-oriented programming. In A. da Rocha Costa, R. Vicari, and F. Tonidandel, editors, *Advances in Artificial Intelligence 2010*, volume 6404 of *LNCS*, pages 21–30. Springer, 2011.

[25] E. J. González, A. F. Hamilton, L. Moreno, R. L. Marichal, and J. Toledo. Ontologies in a multi-agent system for automated scheduling. *Computers and Artificial Intelligence*, 23(2):157–177, 2004.

[26] D. Grossi, H. Aldewereld, J. Vázquez-Salceda, and F. Dignum. Ontological aspects of the implementation of norms in agent-based electronic institutions. *Computational & Mathematical Organization Theory*, 12(2-3):251–275, 2006.

[27] J. A. Hendler. Agents and the semantic web. *IEEE Intelligent Systems*, 16(2):30–37, 2001.

[28] M. Horridge and S. Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21, 2011.

[29] M. N. Huhns and M. P. Singh. Ontologies for agents. *IEEE Internet Computing*, 1(6):81–83, 1997.

[30] N. R. Jennings, K. P. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[31] L. M. Jeroudaih and M. S. Hajji. Extensions to some AOSE methodologies. *World Academy of Science, Engineering and Technology*, 64:383–388, 2010.

[32] G. Kardas, A. Goknil, O. Dikenelli, and N. Topaloglu. Model transformation for model driven development of semantic web enabled multi-agent systems. In *Multiagent System Technologies*, volume 4687 of *LNCS*, pages 13–24. Springer, 2007.

[33] G. Kardas, A. Göknil, O. Dikenelli, and N. Y. Topaloglu. Metamodeling of semantic web enabled multiagent systems. In *Multiagent Systems and Software Architecture, the Special Track at Net.ObjectDays*, pages 79–86, 2006.

[34] A. Katasonov and V. Y. Terziyan. Semantic approach to dynamic coordination in autonomous systems. In *Proceedings of ICAS 2009*, pages 321–329. IEEE Computer Society, 2009.

[35] T. Klapiscak and R. H. Bordini. JASDL: A practical programming approach combining agent and semantic web technologies. In M. Baldoni, T. C. Son, M. B. van Riemsdijk, and M. Winikoff, editors, *Proceedings of DALT 2008*, volume 5397 of *LNCS*, pages 91–110. Springer, 2009.

[36] A. Magkanaraki, S. Alexaki, V. Christophides, and D. Plexousakis. Benchmarking rdf schemas for the semantic web. In I. Horrocks and J. A. Hendler, editors, *Proceedings of ISWC 2002*, volume 2342 of *LNCS*, pages 132–146. Springer, 2002.

[37] J. Martinez-Gil, E. Alba, and J. F. Aldana-Montes. Statistical study about existing OWL ontologies from a significant sample as previous step for their alignment. In *Proceedings of CISIS 2010*, pages 980–985. IEEE Computer Society, 2010.

[38] V. Mascardi and D. Ancona. 1000 years of Coo-BDI. In C. Sakama, S. Sardiña, W. Vasconcelos, and M. Winikoff, editors, *Proceedings of DALT 2011*, volume 7169 of *LNCS*, pages 95–101. Springer, 2011.

[39] V. Mascardi, D. Briola, A. Locoro, D. Grignani, V. Deufemia, L. Paolino, N. Bianchi, H. de Lumley, D. Malafronte, and A. Ricciarelli. A holonic multi-agent system for sketch, image and text interpretation in the rock art domain. *International Journal of Innovative Computing, Information and Control*, 10(1), 2014.

[40] V. Mascardi and A. Locoro. BOwL: exploiting boolean operators and lesk algorithm for linking ontologies. In S. Ossowski and P. Lecca, editors, *Proceedings of SAC 2012*, pages 398–400. ACM, 2012.

[41] V. Mascardi, A. Locoro, and P. Rosso. Automatic ontology matching via upper ontologies: A systematic evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 22(5):609–623, 2010.

[42] J. Mayfield, Y. Labrou, and T. Finin. Evaluation of KQML as an agent communication language. In *Proceedings of ATAL 1995*, pages 347–360. Springer, 1995.

[43] G. Miller. WordNet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[44] 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 J. A. Leite, A. Omicini, L. Sterling, and P. Torroni, editors, *Proceedings of DALT 2003*, volume 2990 of *LNCS*, pages 135–154. Springer, 2003.

[45] Á. F. Moreira, R. Vieira, and R. H. Bordini. Speech-act based communication: Progress in the formal semantics and in the implementation of multi-agent oriented programming languages. In C. Sakama, S. Sardiña, W. Vasconcelos, and M. Winikoff, editors, *Proceedings of DALT 2011*, volume 7169 of *LNCS*, pages 111–116. Springer, 2012.

[46] Á. F. Moreira, R. Vieira, R. H. Bordini, and J. F. Hübner. Agent-oriented programming with underlying ontological reasoning. In M. Baldoni, U. Endriss, A. Omicini, and P. Torroni, editors, *Proceedings of DALT 2005*, volume 3904 of *LNCS*, pages 155–170. Springer, 2006.

[47] M. Nagy, M. Vargas-Vera, and E. Motta. Dssim-ontology mapping with uncertainty. In P. Shvaiko, J. Euzenat, N. F. Noy, H. Stuckenschmidt, V. R. Benjamins, and M. Uschold, editors, *Proceedings of OM-2006*, volume 225 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.

[48] N. Negroponte. *Being Digital*. Alfred A. Knopf, Inc., 1995.

[49] M. Obitko and V. Snášel. Ontology repository in multi-agent system. In M. H. Hamza, editor, *Proceedings of AIA 2004*. IASTED, 2004.

[50] A. S. Rao and M. P. Georgeff. BDI agents: From theory to practice. In V. R. Lesser and L. Gasser, editors, *Proceedings of ICMAS'95*, pages 312–319. The MIT Press, 1995.

[51] A. Ricci, M. Piunti, and M. Viroli. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, 23(2):158–192, Sept. 2011.

[52] A. Ricci, M. Piunti, M. Viroli, and A. Omicini. Environment programming in CArtAgO. In A. El Fallah Seghrouchni, J. Dix, M. Dastani, and R. H. Bordini, editors, *Multi-Agent Programming: Languages, Tools and Applications*, pages 259–288. Springer, 2009.

[53] A. Ricci, M. Viroli, and A. Omicini. Programming MAS with artifacts. In R. P. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Proceedings of PROMAS 2005*, volume 3862 of *LNAI*, pages 206–221. Springer, Mar. 2006.

[54] B. L. Smith, V. A. M. Tamma, and M. Wooldridge. An ontology for coordination. *Applied Artificial Intelligence*, 25(3):235–265, 2011.

[55] C. Tempich and R. Volz. Towards a benchmark for semantic web reasoners - an analysis of the DAML ontology library. In Y. Sure and Ó. Corcho, editors, *Proceedings of EON 2003*, volume 87 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[56] Q.-N. N. Tran and G. Low. MOBMAS: A methodology for ontology-based multi-agent systems development. *Information & Software Technology*, 50(7-8):697–722, 2008.

[57] 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. *Journal of Artificial Intelligence Research*, 29:221–267, 2007.

[58] W3C. OWL Web Ontology Language Overview – W3C Recommendation 10 February 2004.

[59] M.-H. Wang, C.-S. Lee, K.-L. Hsieh, C.-Y. Hsu, G. Acampora, and C.-C. Chang. Ontology-based multi-agents for intelligent healthcare applications. *Journal on Ambient Intelligence and Humanized Computing*, 1(2):111–131, 2010.

[60] S.-Y. Wang, Y. Guo, A. Qasem, and J. Heflin. Rapid benchmarking for semantic web knowledge base systems. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *Proceedings of ISWC 2005*, volume 3729 of *LNCS*, pages 758–772. Springer, 2005.

[61] T. D. Wang, B. Parsia, and J. A. Hendler. A survey of the web ontology landscape. In I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, editors, *Proceedings of ISWC 2006*, volume 4273 of *LNCS*, pages 682–694. Springer, 2006.

[62] Y. Zou, T. Finin, L. Ding, H. Chen, and R. Pan. Using semantic web technology in multi-agent systems: a case study in the TAGA trading agent environment. In *Proceedings of ICEC'03*, pages 95–101. ACM, 2003.

```
/* FMA agent */

+!start : true <-
!endothelium_of_arteriole(Source1)[o(ont1)];    /* OK MATCH; NO LOCAL; OK TRUSTED */
!endothelium_of_arteriole(Source2)[o(ont1)];
!urethral_gland(Source3)[o(ont1)];              /* OK MATCH; NO LOCAL; OK TRUSTED */
!urethral_gland(Source4)[o(ont1)];
!tarsal_plate_of_eyelid(Source5)[o(ont1)];      /* OK MATCH; NO LOCAL; OK TRUSTED */
!tarsal_plate_of_eyelid(Source6)[o(ont1)];
!root_of_tooth(Source7)[o(ont1)];               /* OK MATCH; OK LOCAL; OK TRUSTED */
!root_of_tooth(Source8)[o(ont1)];
!epithelium_of_ciliary_body(Source9)[o(ont1)];  /* OK MATCH; OK LOCAL; OK TRUSTED */
!epithelium_of_ciliary_body(Source10)[o(ont1)];
!splenic_lymph_node(Source11)[o(ont1)];         /* OK MATCH; OK LOCAL; NO TRUSTED */
!splenic_lymph_node(Source12)[o(ont1)];
!internal_thoracic_vein(Source13)[o(ont1)];     /* OK MATCH; OK LOCAL; NO TRUSTED */
!internal_thoracic_vein(Source14)[o(ont1)];
!conceptWithNoMatch1(Source15)[o(ont1)];        /* NO MATCH; OK LOCAL; NO TRUSTED */
!conceptWithNoMatch1(Source16)[o(ont1)];
!conceptWithNoMatch2(Source17)[o(ont1)];        /* NO MATCH; OK LOCAL; NO TRUSTED */
!conceptWithNoMatch1(Source18)[o(ont1)];
!conceptWithNoMatch3(Source19)[o(ont1)];        /* NO MATCH; OK LOCAL; NO TRUSTED */
!conceptWithNoMatch1(Source20)[o(ont1)];

+!epithelium_of_ciliary_body("www.epith_of_ciliary_body.org")[o(ont1),source(self)] <-
   .print("Using fma plan for epithelium_of_ciliary_body").
+!splenic_lymph_node("www.splenic_lymph_node.org")[o(ont1),source(self)] <-
   .print("Using fma plan for splenic_lymph_node").
+!internal_thoracic_vein("www.internal_thoracic_vein.org")[o(ont1),source(self)] <-
   .print("Using fma plan for internal_thoracic_vein").
+!root_of_tooth("www.root_of_tooth.org")[o(ont1),source(self)] <-
   .print("Using fma plan for root_of_tooth").
+!conceptWithNoMatch1(only_local)[o(ont1),source(self)] <-
   .print("Using fma plan for conceptWithNoMatch1").
+!conceptWithNoMatch2(only_local)[o(ont1),source(self)] <-
   .print("Using fma plan for conceptWithNoMatch2").
+!conceptWithNoMatch3(only_local)[o(ont1),source(self)] <-
   .print("Using fma plan for conceptWithNoMatch3").
```

Fig. 5. CooL-AgentSpeak plans of FMA agent (scenario 1, AROMA matching method).

```
/* NCI agent */

+!arteriole_Endothelium("www.Endothelium_of_arteriole.org")[o(ont2)] <-
   .print("Using nci plan for arteriole_Endothelium").
+!urethra_Gland_MMHCC("www.Urethra_Gland.org")[o(ont2)] <-
   .print("Using nci plan for urethra_Gland_MMHCC").
+!tarsal_Plate("www.tarsal_Plate.org")[o(ont2)] <-
   .print("Using nci plan for tarsal_Plate").
+!ciliary_Epithelium("www.ciliary_Epithelium.org")[o(ont2)] <-
   .print("Using nci plan for ciliary_Epithelium").
+!radix_Dentis("www.radix_Dentis.org")[o(ont2)] <-
   .print("Using nci plan for radix_Dentis").
```

Fig. 6. CooL-AgentSpeak plans of NCI agent (scenario 1, AROMA matching method).