Exploiting interaction contexts in P2P ontology mapping

Paolo Besana, Dave Robertson and Michael Rovatsos
Centre for Intelligent System and their Applications
School of Informatics
University of Edinburgh

Abstract

Agents in peer-to-peer networks may gather into virtual communities, interacting continuously with agents that represent disparate actors, each of them with different interests, needs and views, and having dissimilar ontologies. Mapping all the combination of ontologies in advance is not feasible simply because all the possible combinations cannot be foreseen. Mapping complete ontologies at run time is a computationally expensive task. The framework proposed in this paper maps only the terms encountered in a dialogue, or those needed to map them. The efficiency in the mapping process is increased by accumulating experience and exploiting it in order to reduce the number of mapping candidates to verify, focusing only on the most likely ones.

1 Introduction

Open and Peer-to-Peer networks can form the bases for the ascent of virtual communities populated by agents. Agents, acting on the behalf of their owner, will enter and exit these communities: some will offer or sell services and knowledge, others will look for them.

For example, in a possible future scenario when e-paper will be reality and comfortably readable by humans, we can think of communities composed of libraries, publishers, readers and reviewers. The agents representing these actors will gather together and interact: readers will query libraries for books, and will download copyrighted e-documents with an expiration date, or buy them from publishers. Libraries will buy rights for lending the documents from publishers, or will ask to other libraries for external lending. Authors will contact publisher offering their books, reviewers will be contacted by publishers or readers for an opinion.

Some agents will work under the direct control of the user they represent, such as the reader, with limited autonomy. Other agents will have different levels of autonomy, depending on the role they have in the interaction: for example the interactions for lending a book or the sale of a licence may be completely autonomous, while other interactions, such as those involving the choice of what book to buy for a library, will be kept by under the supervision the real world actors.

1.1 Protocol-Based Coordination

Most of the interactions between agents will follow a rather predetermined path: there is usually no need to plan them every time from the scratch, and there is often little need to reason on the proper reaction for each of the received messages: *protocols*, described in languages such as the Lightweight Coordination Calculus (LCC) [8, 9], can be very conveniently exploited for the interactions.

An LCC protocol, such as the one shown in figure 1, describes a dialogue among different agents. It is based on process calculus, expressed with horn clauses. Sending and receiving messages are the basic behaviours, while more complex behaviours can be expressed using connectives that allow the creation of sequences, choices or parallelisations. The protocols are logic programs that can be directly executed once they are received or loaded. A protocol contains a clause for each role, that describes the possible steps for an agent that has taken the specified role in the dialogue. Each step can have a precondition that must satisfied before proceeding, and postconditions that must be satisfied after

```
a(reader(L), R) ::=
download(Title) => a(library,L) <-- want(Title)
then
( ask(accept(Licence)) <= a(library,L)
  then (tell((accept(Licence)) => a(library,L) <-- acceptable(Title, License)
      then document(Title, File)) <= a(library,L))
      or cancel => a(library,L))
)
or unavailable <= a(library,L).

a(library, L) ::=
download(Title) <= a(reader(_),R)
then
(ask(accept(Licence)) => a(reader(_),R) <-- under_licence(Licence,Title)
  then ((tell(accept(Licence)) <= a(reader(_),R)
      then document(Title,File) => a(reader(_) <-- fetch(Title,Document))
      or cancel <= a(reader(_),R))
)
or unavailable => a(reader(_),R).
```

Figure 1: LCC protocol for an interaction between a library and a reader

the step is taken. Protocols should be coupled with ontologies that describes the terms used in the messages and in the constraints.

Protocols make few assumptions on coordination skills of the agents. Fewer assumptions means that more agents can take part in the communities: "socially simple" agents can just execute predefined protocols, still offering their specialised knowledge or services. More sophisticated agents, on the other hand, can synthesise protocols, planning the interactions needed for more complex tasks.

1.2 Ontologies mismatch

In the described scenario, each agent will define its knowledge, services and goals using an ontology. A library agent will have an ontology to classify its books, to define the features of books (author, title, publisher...). A publisher agent will have an ontology to classify its books and to define their features, as well as to define the details about the licences and pricing. A reader or a reviewer will have an ontology to classify books.

Each agent will likely have different ontologies. Sometimes the differences simply reflect the different views of the developers, often they mirror the different needs and interests of their owner.

Imposing the same ontology on all agents is difficult and probably infeasible. First some "social" problem arise: who imposes it? Why should others accept it? Besides, differences in the interests and needs can make hard to create a consistent ontology that contain all the concepts. Finally, it is difficult to keep track of the evolution of an ontology: some agents may keep the pace of the updates, while others may remain with out of date versions. Different versions of the same ontology can sometimes be treated as different ontologies [5].

1.3 Ontology Mapping

A more flexible approach to tackle this heterogeneity consists in finding mappings between the ontologies: most mapping processes are aimed at statically aligning or merging complete ontologies [6, 4, 7]: two or more ontologies are reconciled and the result is stored for possible future use. The research is focused on the process of mapping, and it is interested in mapping as many concepts as possible in the ontologies.

In open multi-agent systems, such as the one described in the introduction, agents interact with many different agents, and it is impossible to foresee all the possible combinations of ontologies. At the same time, mapping whole ontologies at every interaction is often not practicable: it is generally a lengthy process while interactions are often required to be rapid and can also occour simultaneously. Moreover, agents may have ontologies that cover dissimilar domains, with only parts overlapping. The mapping process tries to find relations for all the concepts, and fails on most of them, wasting resources.

1.4 Proposed approach

However, a complete ontology mapping is not a requirement for interactions: agents need to understand each other just enough to carry out their task. Once the task is performed, the mutual understanding is no longer important, as agents interact continuously with different agents, and the mapping found once might be useless other times. In fact, agents need to share only the parts of their knowledge contextual to the interaction in which they are involved.

In this paper we present a framework to map dynamically, and only when needed, the portions of the ontologies required for perform an interaction dialogue.

2 General definitions and assumptions

2.1 Agent model

Each agent a_i has its own communication environment e_i, consisting of the ontology O_i that defines the terms available to the agent and of the axioms it can use to reason. An environment can be seen as the context of an agent, as described in [2], but renamed to avoid name conflicts with the concept of context used in this paper. A definition is valid only within an environment, and an agent can reason over concepts defined in other environments only if mapped to some concepts in its own.

We can model an agent as being composed by two layers: a *communication layer*, and a *reasoning layer*. The communication layer is independent from the ontology. It interprets the protocols described in section 1.1 and handles the transmission and reception of messages. The reasoning layer contains all the agent's skills and knowledge, and it is accessible from the communication layer through access points.

During the execution of a protocol script, the communication layer of the agent a_i asks the reasoning layer to satisfy the constraints $\Psi_k = \{C_1, ..., C_n\}$ that define the preconditions of a step s_k. In the example, the library agents must satisfy the constraint `available(Title)` before proceeding in the dialogue.

2.2 Ontologies

The ontologies referred in this paper are taxonomies with roles. Concepts can subsume each other, as well as be related by roles, which can be with or without value restrictions. At this stage concepts defined using more complex constructors, such as *intersection*, *union* or *complement*, would not be understood by the system.

2.3 Communication model

An interaction k between agents a_i and a_j is an exchange of messages m. Each message is sent coupled with the interaction protocol P and consists of a sequence of terms:

$m = \langle t_1, \ldots, t_n \rangle$

For brevity, when talking about an agent, terms defined in other environments will be called *external terms*, and will be referenced as t_i, while terms defined in the agent's environment will be called *internal terms*, and will be referenced as w_i, as in the ontology in figure 3.

For any interaction k every involved agent a_i publishes an ontology subset O_{ki} , valid in the context of the interaction, to explain the terms it has inserted in the dialogue. For example, the library agent should publish an ontology to define the different possible licences under which the documents can be downloaded.

2.4 Semantic bridges

The semantic relations between terms defined in different environments are defined as semantic bridges (or mappings or simply bridges) [7]. A bridge b is the tuple:

$b = \langle relation, t, w, c(true), c(false) \rangle$

where $relation$ is the hypothesised semantic relation between the terms t and w (equivalence or subsumption), while $c(true)$ is the confidence level that the bridge is correct and $c(false)$ that the bridge is wrong. This because the two confidence levels might not sum to one, as ignorance should be taken into account.

Within an environment, a bridge b_h is more generic ($\succeq$) than another bridge b_g, if the external term *t* is the same in both and the internal term w_h of b_h subsumes the internal term w_g in b_g ($w_h \sqsupseteq w_g$):

$b_h \succeq b_g \leftrightarrow (w_h \sqsupseteq w_g) \wedge (t_h = t_g)$

Conversely b_h is more specific than bridge b_g if w_g subsumes w_h ($w_h \sqsubseteq w_g$):

$$b_h \preceq b_g \leftrightarrow (w_h \sqsubseteq w_g) \wedge (t_h = t_g)$$

During the interaction k the bridges are stored in the set B_k.

For example, using the ontology in figure 3, the bridge $\langle t \sqsubseteq w_2 \rangle$ is more generic than the bridge $\langle t \sqsubseteq w_5 \rangle$ because w_2 is more generic and subsumes w_5.

3 Framework

The mapping process is the core of the framework. During an interaction k, it receives an external term t, uses its own ontology and returns the most specific mapping found b_{tk}. It is an iterative process, as described in figure 2, with a primary loop and a secondary loop.

At every iteration i, a semantic bridge b_{tki} is created for the external term t. The bridge b_{tki} is more specific than the bridge $b_{tk(i-1)}$, verified in the previous iteration:

$$b_n \preceq b_{n-1} \preceq \ldots \preceq b_1$$

In the main loop, at every iteration *i*, the framework executes three steps: it first generates hypotheses for the most generic mappings that imply the mapping $b_{tk(i-1)}$, then filters the most probable hypotheses, and finally collects evidence for the remaining hypotheses, selecting the most reliable hypothesis.

The loop ends when it becomes impossible to generate hypotheses that imply those proved in the previous step, or none of the hypotheses generated can be proved. The bridge created in the last iteration, and therefore the most specific, is returned and added to the set B_k of all the bridges used in the interaction k. These mappings are then used to translate the requests from the communication layer to the reasoning layer.

3.1 Generate the hypotheses

At each iteration *i* the function GENERATE-HP receives the external term *t* and the bridge $b_{tk(i-1)}$ containing the mapping proved in the previous iteration, and returns a set of hypotheses Ω about the most generic mappings that imply $b_{tk(i-1)}$.

For example, if $b_{tk1} = (t \sqsubseteq w_1)$, $b_{tk2} = (t \sqsubseteq w_2)$, given the ontology in figure 3, then for the iteration #3:

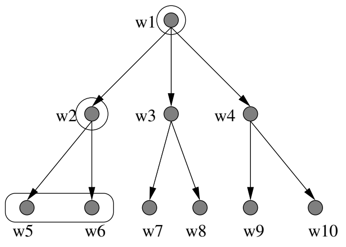

Figure 3: O_i ontology

$$\Omega_3 = \{\langle\{\sqsubseteq, \sqsupseteq, \equiv\}, t, w_5\rangle, \langle\{\sqsubseteq, \sqsupseteq, \equiv\}, t, w_6\rangle\}$$

The generation must include, as hypotheses, the cases in which the term in the message should not be mapped: when it is a number, or when it is a new term received by the agent as a reply to a question. For instance, if an agent asks to a library who are the most downloaded authors, the reply is likely to be a set of new terms for the inquiring agent. Mapping the terms to a class `person` is logically consistent, but it is meaningless for the purpose of the dialogue.

3.2 Filter the hypotheses

The framework uses *prefilters* to prune hypotheses: prefilters should process quickly the hypotheses using heuristics about past and current interactions, without requiring any symbolic reasoning or semantic analysis of the mappings proposed. The aim is to minimise, on average, the number of wrong hypotheses to check.

The function FILTER-HP receives a set of hypotheses Ω and returns a subset Ω' of hypotheses. It combines the results of different prefilters, extracting the intersection of the subsets generated by each filter.

If none of the filtered hypotheses is proved, the function may be called more than once in the same main iteration, and at each round the function relaxes the filter to obtain a wider set of argument trees.

A filter f_i is characterised by its *breadth* and *confidence* and is composed by two functions, FILTER() and FEEDBACK(): FILTER() is the function that actually prunes the hypotheses, while FEEDBACK() is called after a term is successfully mapped, and receives the new bridge that can be used to improve the prefilter's heuristics.

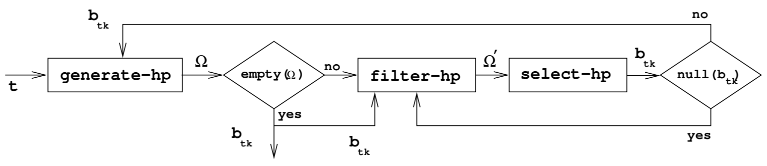

Figure 2: framework process

The *breadth* is the "band-pass" of the filter: the narrower, the fewer hypotheses are left to verify. If the function FILTER-HP must be repeated within the same iteration, the narrowest filter used in the previous call is removed from the set of applied filters, until no filters are left.

The element *confidence* indicates how likely is it that the correct hypothesis is in the subset selected by the FILTER().

3.3 Select the best hypothesis

In this step, the system processes the set of hypotheses filtered by the previous step, and tries to extract the most likely one. If the system fails to select any hypothesis, it goes back to the previous step, relaxes the filter if possible, and tries to obtain a wider set of hypotheses, as described in the previous section.

This step is composed by three actions.

3.3.1 Collect evidence

The framework processes the filtered hypotheses using *selection rules*. A selection rule generates an argument in favour or against a hypothesis. Rules can be based on syntactic matching or can be based on semantic matching. A possible rule can search for common substrings. Another possible rule can check if the terms in the hypothesis are synonyms using an external oracle, such as WordNet. A more complex rule can check, when terms are properties, if they have equivalent range and domain, generating more hypotheses.

Algorithms developed and tested for static mapping can be used by the rules to collect evidence for or against a hypothesis: working only on a subset of hypotheses, these algorithms do not need to compute useless mappings. Therefore, algorithms like S-Match or CTXMatch can be used for this purpose.

Arguments gathered in favour or against a hypothesis are organised in a tree: the root is the hypothesis to verify. The branches connected to the root are arguments attacking or supporting the hypothesis in the root. A rule can produce an argument that requires further verification. For example, to check the equivalence between two properties, the system may use a rule that checks if the two properties have equivalent domains, as the rule `r2` in figure 4. The rule generates the argument `<eqDom,t1,w2>`. To accept the argument as a support for the hypothesis, the system must verify the equivalence between the domains `t2` ad `w3` of the properties, for example using the rule `r1`. If the terms are not equivalent, then the argument becomes an attack to the hypothesis.

Formally, a rule is the tuple:

$$r_i = \langle id, \pi, I, proc, c(hp|arg), c(\neg hp|\neg arg)\rangle$$

where π is the pattern of the proposition in the hypothesis to verify, the set I is the information the system needs to collect in order to generate the argument. The information is about terms semantically related to the terms in the proposition (such as, for example, the superclasses, the subclasses or the instances). Information about internal terms are easily accessible, while the

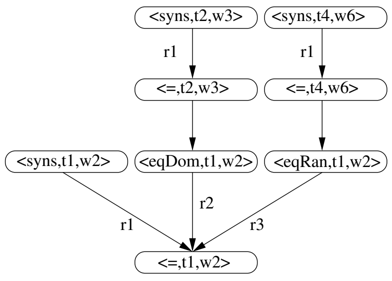

Figure 4: An argument tree

agents may need to ask to the other agent for information about external terms, if the information is not directly available in the published ontology.

The rule specifies two confidence levels, that measures how strong is the support or the attack of the argument generated by the rule: $c(hp|arg)$ is the confidence level that the hypothesis is true, given that the supporting argument is true, while $c(\neg hp|\neg arg)$ that it is false, given that the supporting argument is false.

The system processes one argument tree at a time, and returns an updated version of the tree: it finds the first argument to prove, extracts the proposition, chooses the matching rule and calls the procedure specified in the rule, creating an argument added to the tree.

3.3.2 Combine evidence

The arguments in the tree are combined to obtain two confidence levels for the hypothesis: one that the hypothesis is true, and one that it is false.

If a hypothesis has just one argument, the confidence that the hypothesis is true is the *a posteriori* confidence that the hypothesis is true given that the argument is true:

$$c(hp) = c(hp|arg)c(arg)$$

where the *a posteriori* confidence $c(hp|arg)$ is given by the rule, while the confidence of the argument being true $c(arg)$ is computed for the argument, either by the procedure itself, or recursively from other arguments. Similar considerations apply for the *a posteriori* confidence that the hypothesis is false. When there is more than one argument, the *a posteriori* confidences must be combined.

It cannot be assumed that the confidences about the truth values of a hypothesis sum to one: if a rule establishes that a hypothesis is true with 40% of confidence, it does not imply that the hypothesis is false with 60% of confidence. Therefore, the theory used to combine confidences should be able to express ignorance about the truth value of the hypothesis.

One possible approach is Dempster-Shafer theory [10], which computes the probability of a proposition supported by evidences. In Dempster-Shafer theory the belief in a proposition and the belief in its negation are not required to sum to one: the theory introduces the concept of plausibility, that is the extent to which the available evidence fails to refute the proposition. The interval between the belief and the plausibility is the ignorance about a proposition.

Moreover the theory provides a formula, called Dempster's rule of combination, to combine the evidences m_i for a proposition A.

In Dempster-Shafer theory, $c(hp)$ is interpreted as the belief that the hypothesis is true, while $1 - c(\neg hp)$ is the plausibility that the hypothesis is true.

3.3.3 Harvest Hypothesis

At the end of an iteration *i* , the hypothesis with the highest confidence is selected. If there are more than one hypothesis within a narrow band of confidence, the system first tries to apply more rules - if available - to gather more evidence for the conflicting hypotheses. If the conflict remains, it takes the strongest hypothesis, even if the interval is minimal. Then the procedure restarts, until no more hypotheses can be generated.

The bridge obtained in the last iteration is extracted and pushed into the set B_k of the bridges for the interaction.

3.4 Relation between filters and rules

As the heuristics available to the prefilters improves using the feedback obtained from proved hypotheses, the confidence about the result increases and the set of hypotheses filtered narrows: in the long run, the prefilters can replace the rules, at least for some external terms, making the mapping process quicker.

4 Statistical prefilters

As noted in the section 3.2, prefilters must operate rapidly, making it difficult to apply complex, symbolic or inductive inference methods. Nevertheless, it is important that mapping performance should improve with experience. One way to avoid this dilemma is to harness the large volume of event-based data available as feedback from previous interactions in order to determine statistical patterns.

4.1 Statistical contexts

As hinted in section 1.4, contexts can be the key to focus the search of the correct mappings: an external term is unlikely to be mapped to a term that is completely unrelated from the context of the interaction.

There are many different proposals, from various disciplines, for the theoretical definition of contexts [3], although remains the problem of how to create a context for a conversation.

Yet, contexts can be seen as possible patterns in interactions: some terms tend to appear together in different dialogues about similar topics. Some of these terms are actually contextual to the topic of the conversation (*document*, *download*,...), and do not appear in conversations about other subjects, while other terms are auxiliary to any kind of conversation (*ask*, *inform*,...).

Following this intuition, the terms can be clustered together, and each cluster is a possible context for an interaction. The contexts are created and updated using the feedback from the framework and they are used to predict which are the most likely terms that can occour during a conversation, excluding hypotheses relative to unrelated terms.

4.1.1 Definition

More formally, a *context* is a set of internal elements η, identified by a unique number and characterised by the number N of dialogues used to build this context.

Each term element η_i in S is a pair:

$\eta_i = \langle w, \mu_C \rangle$

where w is the term in the agent's ontology and μ_C is the grade of membership of the term in the context: terms can appear in contexts with different frequencies: some will occour in every dialogue classified by the context, other will appear more rarely. The same term can appear in different contexts with different grades of membership.

The function $\mu_C(K)$ returns the grade of membership to a context C of a set K of terms:

$\mu_C(K) = \frac{1}{|K|} \sum_{w \in K} \mu_C(w)$

4.1.2 Use

Contexts are used to classify dialogues as they are performed. After a new term is mapped during the interaction, the system tries to reclassify the dialogue: it searches the contexts that maximise the function $\mu_C(W)$, where W is the set of internal terms in the bridges contained in B_k.

When an interaction starts, the framework will use the protocol ontology to map the terms in the clauses relative to its role in the uninstantiated LCC dialogue. Following the example described in the introduction, if the reader agent wants to download an ebook, it first obtains the protocol, shown in figure 1, needed for the interaction with the library, and then it tries to map the terms `download`, `library`, `want`, `ask`, `accept`, `document`, `unavailable` contained in the clause about the role `reader`.

At the beginning of this process, there are few mapped terms and it is difficult to classify the dialogue properly, and more than one context can classify it. When all the uninstantiated dialogue is mapped, the number of terms in W reduces the number of contexts that can classify the dialogue.

Then, as the interaction proceeds, the number of terms in W increases, further reducing the number of contexts that can classify the dialogue.

The selected contexts are used to filter the generated hypotheses set: if some terms in the set never appear in the contexts, then it is possible to exclude these hypotheses, adding evidence for the remaining hypotheses.

Therefore, exploiting the received protocol, the system is already able, at the start of a conversation, to focus on a reduced set of terms that are more likely than others to show up during the conversation, and this focus increases with the conversation.

When a dialogue is finished, the internal terms W used in it are stored and added to a context. The first dialogue ever to be performed by the agent creates the first context. The terms in W are inserted in the context that classifies them better. If no context classifies them well enough, then a new context is created.

4.2 Past mapping experience

Another possible pattern to identify is that some external terms, independently from the ontology that defines them, have always the same semantic relations with the same terms in the agent's ontology. Each stored mapping is coupled with a confidence level: the more often a particular mapping is proved, the higher its confidence is.

4.2.1 Definition

The set of previous mappings Λ contains a tuple λ_i for each mapping proved in the past, composed by three elements:

$\lambda_i = \langle b, s_m, n_a \rangle$

b is the hypothesis verified in the past, s_m is the cumulative confidence of the hypothesis being true, and n_a is the number of time the term mapped in the hypothesis has appeared in dialogues. The cumulative confidence is always related to the number of appearances of the term: if a term has appeared very rarely, the system cannot rely much on the past mapping.

4.2.2 Use

When the system must select hypotheses for an external term, it can look in past mappings for the term: it then keeps the hypotheses implied by the past mappings, and discards the others.

For instance, during a certain iteration in the mapping process, given the generated hypotheses $\Omega = \{\mathtt{t} \sqsubseteq \mathtt{w}_2, \mathtt{t} \sqsubseteq \mathtt{w}_3, \mathtt{t} \sqsubseteq \mathtt{w}_4\}$, the past mappings $\Lambda = \{\langle \mathtt{t} \sqsubseteq \mathtt{w}_5, 4, 5\rangle\}$, and the ontology in figure 3, the strategy should discard both $\mathtt{t} \sqsubseteq \mathtt{w}_3$ and $\mathtt{t} \sqsubseteq \mathtt{w}_4$ and keep $\mathtt{t} \sqsubseteq \mathtt{w}_2$ as the past mapping $\mathtt{t} \sqsubseteq \mathtt{w}_5$ implies it.

Mappings established for a particular external ontology, and received as feedback from the framework, are stored for future use. When mappings are encountered repeatedly, the confidence s_m in the past mapping λ_i is increased by the confidence in the bridge, while n_a is incremented by one in all the stored bridges relative to the same external term.

There is no issue about inconsistency, as conflicting past mappings are used only as suggestions about the order in which the hypotheses should be checked: conflicting hypotheses are tolerated by collecting evidence in favour or against them.

5 Related work

The QOM project (Quick Ontology Mapping), described in [1], addresses the problem of trading quality for efficiency, in a partially similar way. As in this framework, the possible mapping candidates are filtered using different strategies in order to reduce the time spent computing similarities between unrelated terms.

However, it is oriented toward mapping whole ontologies: there is no concern about the contexts of interactions, and the filters are only based on the ontologies themselves (hierarchy, node labels, etc).

6 Conclusion

In this paper we presented a framework to allow agents in open, non deterministic networks, to interact with other agents that do not share the same ontology. In the framework, only the portion of ontologies relevant to the interaction are mapped: the system searches the relation between terms encountered in the dialogue and terms defined in its own ontology. The efficiency of this search is improved exploiting the structure of the ontologies and the statistical patterns threaded in the dialogues.

Although at an early stage, with many details still to be resolved, the approach seems promising, especially coupled with the protocol architecture.

References

[1] Marc Ehrig and Steffen Staab. Qom - quick ontology mapping. In *International Semantic Web Conference*, pages 683–697, 2004.

[2] Fausto Giunchiglia. Contextual reasoning. Technical report, IRST, Istituto per la Ricerca Scientifica e Tecnologica, 1992.

[3] Fausto Giunchiglia. A context-based framework for mental representation. Technical Report 9807-02, IRST Istituto per la ricerca scientifica e tecnologica, july 1998.

[4] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-match: an algorithm and an implementation of semantic match. In *In Proceeding of the European Semantic Web Symposium*, pages 61–75, 2004.

[5] A. Hameed, A. Preece, and D. Sleeman. *Ontology Reconciliation*, pages 231– 250. Springer Verlag, Germany, 02 2003.

[6] Yannis Kalfoglou and Marco Schorlemmer. Ontology mapping: the state of the art. *Knowledge Engineering Review*, 2003.

[7] Silva Nuno and Joao Rocha. Mafra - an ontology mapping framework for the semantic web. In *Proc. of the 13th European Conf. on Knowledge*, 1999.

[8] D. Robertson. Multi-agent coordination as distributed logic programming. In *International Conference on Logic Programming*, Sant-Malo, France, 2004.

[9] David Robertson. A lightweight coordination calculus for agent systems. 2004.

[10] Yager. *Advances in the Dempster-Shafer Theory of Evidence*. John Wiley, New York, 1994.