Information Agents That Learn to Understand Each Other Via Semantic Negotiation

Salvatore Garruzzo and Domenico Rosaci

DIMET, Università Mediterranea di Reggio Calabria
Via Graziella, Località Feo di Vito
89060 Reggio Calabria, Italy
{salvatore.garruzzo, domenico.rosaci}@unirc.it

Abstract. A key issue in Distributed Applications, that widely use Information Agents for implementing several typologies of services, is that of making reciprocally understandable the meaning of terms contained in the exchanged messages, in those cases where agents use different, heterogeneous ontologies. A possible way for facing this issue is offered by the *semantic negotiation*, a framework in which agents try to understand each other by negotiating the semantic of the terms. Several models and protocols of semantic negotiation have been proposed in the last years. However, most of these approaches are not able to support semantic negotiation without requiring agents either to share knowledge or to use a global common ontology, and none of them provides a semantic negotiation protocol that allows the whole agent community to contribute to the semantic understanding process between each agent pair. In this work, we propose the HIerarchical SEmantic NEgotiation (HISENE) protocol, based on the idea that an agent a should be able to partition the set of the other agents on the basis both of their personal expertise of the application domain, as well as on the particular capability that each of them shows in understanding a. We also give an implementation of the proposed protocol in the standard Java Agent DEvelopment Framework (JADE).

1 Introduction

In human discussions, the meaning of terms contained in the statements are not always reciprocally clear for both the interlocutors. Often, one of them uses a term that the other one either does not understand or considers ambiguous. Generally, human beings try to solve these situations by *negotiating* the semantics of the involved terms, where the negotiation implies several operations performed by the two interlocutors as, for instance, a query that one of them could pose for having a description of a non-understood term, a response provided by the other interlocutor, containing the requested description, etc. This scenario, very usual in human context, has today a counterpart in Distributed Applications field, where distributed software entities, generally called *information agents*, operates on the behalf of human beings to perform operations that would be

© IFIP International Federation for Information Processing 2006

too onerous to be completed manually, as information searching, e-commerce and e-learning activities, software exchanging and so on. On the one hand, each information agent generally stores an internal representation, called *ontology*, of the domain of interest for its human owner. On the other hand, agents communicate between each other in a distributed Multi-Agent System (MAS) to perform their activities. As an example, consider the case of an e-commerce scenario in which an agent, operating on the behalf of a human customer, negotiates for a product with another agent operating on the behalf of a human seller. This communication is performed effectively in the case the two agents share the same ontology, i.e. if both of them know the same terms and give the same meanings to the terms. Otherwise, the problem arises for an agent interpreting some terms unknown or ambiguous contained in messages arriving from the other agent. It is important to consider that nowadays communications among agents have become a key issue for the development of the whole Web, and not just some particular application domain as e-commerce and e-learning. A suitable example for understanding this fact is represented by the general case of Web Services, that can be viewed as (server) agents that provide services to other (client) agents. It is necessary that, as Web Services become more prevalent, client agents should be able to compose together disparate Web Services. However, in order to enable such compositions, it is not enough just agreeing on common protocols (e.g. SOAP) but also the messages' contents need to be mutually understandable: this means that there should be an agreement on the semantics of the terms used in the messages.

Although we have observed over the last years an important evolution towards the standardization of agent communication languages (ACL's), as KQML [5] and FIPA ACL [6], it is worth to point out that the focus of these standards is mainly on the syntax of messages and the semantics of performatives, while the semantics of the content of a message is specified by the ontology which is used. This means that, in order to correctly understand the content of a message, the receiving agent has to understand the terms contained in the ontology of the sending agent. In a MAS, this is possible if either all the agents share the same ontology, or every agent knows each other's ontology. However, none of these situations are desirable, since: (*i*) every agent generally deals with its own particular task and thus requires its own specialized ontology; (*ii*) making every agent of an open MAS, whose size can quickly increase in time, always acquainted with every other agent's ontology would lead to a untenable situation.

A possible way of facing the problem to solve the difficulties of an agent in understanding the messages coming from other agents having different ontologies is offered by the *semantic negotiation*. This is a process by which agents in an agent community try to reach mutually acceptable definitions (i.e., mutually acceptable agreements on terms).

Several models and protocols of semantic negotiation have been proposed in the last years [2, 4, 7, 11, 12]. However, most of these approaches are not able to support semantic negotiation without requiring agents either to share knowledge or to use a global common ontology, and none of them provides a semantic negotiation protocol that allows the whole agent community to contribute to the semantic understanding process between each agent pair. In this work, we introduce the idea that two agents involved in a communication process can require the help of other agents in order to solve possible understanding problems. In this context, the notion of *expertise* of an agent introduces a measure of the capability of the agent to explain non-understood terms to each other agent. Moreover, we also define the notion of *understanding capability* of an agent a with respect to another agent b, that measures the capability of a to explain terms that b does not understand. Therefore, the expertise of an agent a is the capability of a to effectively explain non-understood terms to the whole community, while the understanding capability with respect to b is relative to the only agent b. These two notions allow the possibility to introduce the synthetic measure of *negotiation degree*, defining the potential capability of a to negotiate the semantic of terms belonging to b. Therefore, in our framework, an agent can ask help to other agents for understanding a term on the basis of their negotiation degree; for this purpose, he groups the agents in different *partitions* $p_1, p_2, .., p_n$, ordered by a decreasing level of negotiation degree. We propose a semantic negotiation protocol, called *HIerarchical SEmantic NEgotiation* (HISENE), that is suitable to be applied for implementing such a semantic negotiation in the standard Java Agent DEvelopment Framework (JADE) [8]. An important advantage that this protocol introduces is that each agent can contact the other agents in different stages, by following the rational criteria of firstly negotiating with the agents belonging to the partition p_1, contacting agents of the partition p_2 only if none of the agents in p_1 is able to positively answer, then contacting agents of the partition p_3 only if none of the agents in p_2 succeeds, and so on. Moreover, each contacted agent can start, in its turn, another semantic negotiation, in order to understand unknown term; however, in order to avoid the presence of a loop, each term is processed only once by each agent. This leads to use in an efficient way the network communication resources. The plan of the paper is the following: Section 2 describes some related work; Section 3 gives some preliminary notions on the JADE framework; Section 4 deals in detail with the HISENE protocol, while Section 5 describes a simple example of how HISENE works; Section 6 draws some final conclusions. The Appendix describes the JAVA implementation of the main components composing the package HISENE, built on top of the JADE framework.

2 Related Work

In a MAS, each agent is specialized in solving a particular task, so it requires its own ontology. In order to allow agents having different ontologies to understand each other, some approaches have proposed in the past the use of a *common shared ontology.* As an example, the approach proposed in [11] provides the agents with a set of shared concepts, in which they can express their private knowledge. The communication vocabulary is formalized as an ontology, shared by the entire MAS, and in which every private concept of each individual agent can eventually be defined. Concept names used in an agent's private ontology, are not understandable to other agents. However, their definitions in terms of ground concepts are understandable. The use of definition terms, instead of the concepts, enables optimal communications between agents.

Moreover, the approach presented in [2] introduces a computational framework for the detection of ontological discrepancies between two agents in multi-agent systems. In this method, presuppositions are extracted from the sender's messages, expressed in a common vocabulary, and compared with the recipient's ontology, which is expressed in type theory. Discrepancies are detected by the receiving agent if it notices type conflicts, particular inconsistencies or ontological gaps. Depending on the kind of discrepancy, the agent generates a feedback message in order to establish alignment of its private ontology with the ontology of the sender. The dialogue framework is based on a simple model of interaction.

Another approach using a common knowledge is that presented in [12], where authors introduce a machine learning methodology and algorithms for multi-agent knowledge sharing and learning in a peer-to-peer setting. Agents can use a set of shared concepts in which they can express their private knowledge.

The work [7] proposes to consider the use of shared keys to solve the problem of using different names for the same object; in particular, a probabilistic matching approach is introduced. Semantic negotiation is described as a process by which a client and a service can negotiate mutually shared references.

There are some other approaches that do not require the use of a shared ontology. As an example, in [4], to allow agents to interoperate, authors have developed a matchmaking system that, rather than requiring agents to share ontologies, exploits an agent-independent, domain-specific ontology, called a *global ontology*. Besides the global ontology, the proposed system, when an agent joins the platform, applies an information-extraction engine to the agent's code to extract useful information, that includes recognized names of concepts the agent uses (e.g. class names, parameter names, etc.). Instead of having a shared ontology, the proposed system maintains a mapping of the local ontologies of all agents to the independent global ontology. The main difference between this approach and a shared ontology approach is that an agent's programmer does not need to know anything about any other agent's local ontology, nor he does need to know about the global ontology, but it is the system that does the necessary mapping.

The main difference between the approaches described above and that one we propose in this paper is that in our approach, agents do not need to share either a common ontology or to maintain a global ontology, in order to understand each other, but they try to solve their understanding problems availing the help of other agents that are considered experts in the involved domain and that have similar ontologies. Obviously, by using this approach, the understanding can be obtained only by waiting that the agent community evolves in time, allowing the formation of expert agents and understanding relationships among agents, due to the continuous interaction. The main advantage that our method presents is that the mutual understanding among agents is not statically related to a global ontology, but it can dynamically improve by following the agent interactions and monitoring the agent communications.

Other approaches exist in the literature, that we consider alternative to our one. As an example, in [3], the problems brought by the schema heterogeneity in Digital Libraries are discussed. The proposed architecture integrates the ontology, agent and P2P technologies together to support the schema mapping. The goal is to allow agents embedded in different libraries to communicate semantically. As another example, in [1], authors present a technique to generate elementary speech act sequences in a dialogue game between an electronic assistant and a computer user. The work focuses on the conversational process of the understanding of the meaning of a vocabulary shared by two dialogue participants, where the computer interface is considered to be a cooperative agent. Another proposal is that contained in [10]. In this work, agents in an open agent system jointly agree on an axiomatic semantics for the agent communications language utterances they will use to communicate. This work assumes that the agents involved all start with a common semantic space, and then together assign particular locutions to specific points in this space. Such a structure would not appear to permit an incremental construction of the semantic space itself.

3 Preliminaries

Agents in a multi-agent system can communicate by means of messages. Information inside a message is represented as a *content expression* consistent with a proper content language and encoded in a proper format. Taking into account that agents have their own way of internally representing the information, it is quite clear that the representation used in a content expression is not suitable for the inside of an agent. For this reason, agents need to convert their internal representation into a content expression representation, and vice versa. Moreover, the problem of different ontology explained in Section 1 determines the impossibility of message understanding.

JADE is a software framework fully implemented in Java language to realize distributed multi-agent systems complied with the FIPA specifications. JADE offers a number of advantages such as: (*i*) each agent "lives" in a runtime environment on a given host; (*ii*) communications are held by means of ACL messages; (*iii*) information can be represented as an instance of an application-specific class (a Java object). Moreover, the support for content languages and ontologies provided by JADE is designed to automatically perform all the above conversion operations, thus allowing developers manipulating information within their agents as Java objects.

In order for JADE to perform the proper semantic checks on a given content expression it is necessary to classify all possible elements in the domain of discourse (i.e. elements that can appear within the content of an ACL message) according to their generic semantic characteristics. This classification is derived from the ACL language defined in FIPA which requires that the content of each ACLMessage must have a proper semantics according to the performative of the

ACLMessage. The JADE *content reference model* considers only four types of elements which can be used as meaningful content of an ACL message, namely:

Predicates, that are boolean expressions saying something about the status of the world. As an example, the expression

$(studies - in\ (Student : name\ Jim)(University : name\ MIT))$

states that "the student Jim studies in the University MIT". Generally, inside predicates there are referenced some expressions called *concepts*, that indicate entities with a complex structure e.g. $(Student : name\ Jim : age\ 21)$.

Agent Actions, indicating actions that can be performed by some agents, e.g. $(sell\ (Book : title\ ``AnnaKarenina'')\ (Person : name\ Jim))$

states that the person Jim sells the book "Anna Karenina".

Identifying Relational Expressions (IRE), that are expressions that identify the entities for which a given predicate is true, e.g. $(all\ ?x\ (studies - in$ $?x\ (University : nameMIT)$ identify all the students for which the predicate $(studies - in\ (Student : name\ x)(University : name\ MIT))$ is true.

ContentElement Lists, that are lists of elements of the above three types.

In the following, we introduce a technique for supporting semantic negotiations among JADE agents that uses the ontology support libraries.

4 The HISENE Protocol

In our framework, we suppose that an integer coefficient e_i, called *expertise coefficient* (that we will call in the following e-coefficient, for shortly) of i, is associated with any agent i of the MAS, representing the degree of expertise that the whole agent community gives to i. Moreover, another integer coefficient u_{ij}, called *understanding capability coefficient* (that we will call in the following u-coefficient, for shortly) of j with regards to i, is associated with each pair of agents (i, j), representing the degree of understanding that the agent j presents with regards to the agent i. Each agent i stores all the u-coefficients in a local database, called *Understanding Coefficient DataBase* ($UCDB_i$), while all the e-coefficients are stored in a global database called *Expertise Coefficient DataBase* ($ECDB$), by means of a yellow pages service provided by a specific agent.

These two coefficients are used by each agent i of the MAS to determine a partitioning in the set of the agents belonging to the MAS. We call AS_i the set of all the agents belonging to the MAS, except the agent i. We call AS_i^k, $k = 1, 2, .., p_i$, the k-th partition determined by the agent i in the agent set AS_i. The agent i decides how many partitions p_i have to be considered; moreover, the criterium for assigning each agent j, belonging to AS_i, to a partition AS_i^k, is represented by a function $p(j)$ that receives the agent j as input and yields as output, on the basis of the overall *negotiation degree* of j, the number of the partition which j has to be assigned to. More in particular, the agent i assigns a weight w_e^i (resp. w_u^i) to the e-coefficient (resp. u-coefficient), representing the importance the agent i gives to the expertise (resp. understanding capability), defines a threshold parameter t_k for each partition $k = 1, 2, ..p_i$, and then

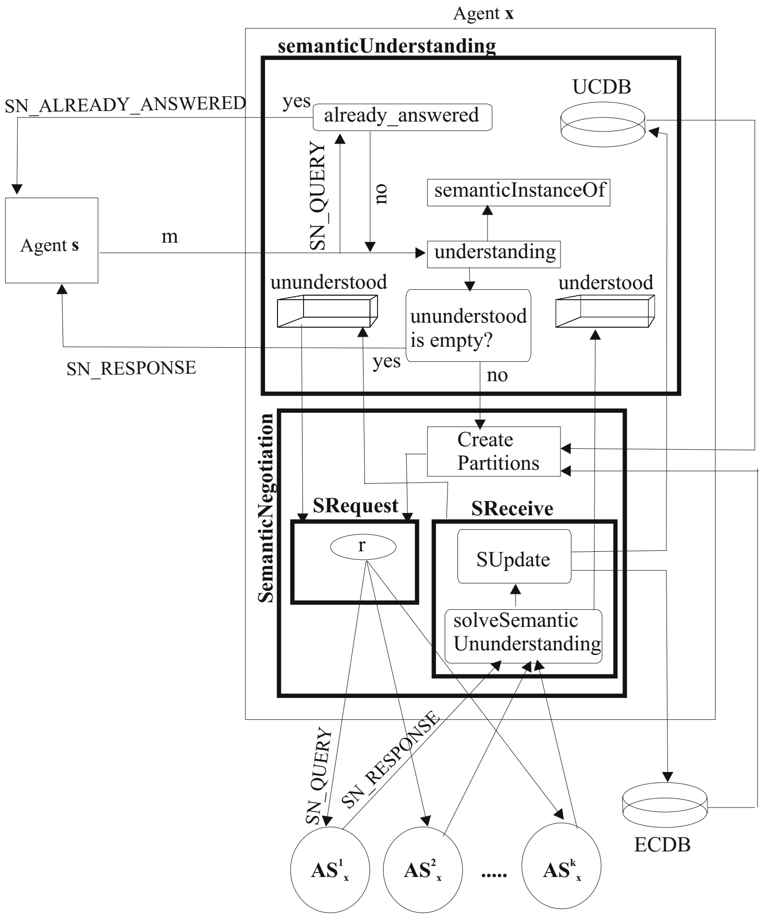


Fig. 1. Semantic Negotiation's Protocol

computes the *negotiation degree* n_{ij} of j as $w_e^i \cdot e_j + w_u^i \cdot u_{ij}$. Then, the function $p(j)$ is calculated as: $p(j) = z$ if $t_{z+1} \leq n_{ij} < t_z$.

Now, we describe the protocol (see Fig. 1) supporting the semantic negotiation followed by an agent x that receives a message m from another agent s.

This message can be an *ordinary* ACL message (i.e., a message with performative INFORM, QUERY_IF, PROPOSE, etc.) or a *semantic negotiation message* (i.e., a message with either performative SN_QUERY_FOR, or SN_RESPONSE, or SN_UNKNOWN or SN_ALREADY_ANSWERED). In the case of an ordinary message, the message's content is composed by a list of r content elements $e_1, e_2, .., e_r$ (see Section 3), where in the case of a semantic negotiation message we have three possibilities: (i) the messages's performative is SN_QUERY: In this case, the content is composed by an AID indicating the agent ia that is interested to the query's result (this agent could be different from the sender s of the message, because the sender could simply be an agent that received in its turn the query from ia and, not being capable to answer the request, decided to request the help of x); (ii) the message performative is SN_RESPONSE: In this case, the content of the message is a list of pairs $(e_1, sl_1), (e_2, sl_2), .., (e_r, sl_r)$ where e_i is a content element in the ontology of x and sl_i is a list of content elements synonyms of e_i in the ontology of the messages's sender: these synonyms could help x to understand e_i; (iii) the message performative is SN_UNKNOWN, meaning that s says that it is unable to give an answer to a previous request of x, or SN_ALREADY_ANSWERED, meaning that s has already answered to a previous request of x: In this case, the content of the message is void.

In order to understand the content of the message m, the agent x executes a $semanticUnderstanding$ behaviour. This latter operates as follows:

1. If the message's performative is SN_QUERY, x first invokes the boolean function $already\_answered(m)$. This function returns $true$ if all the content elements belonging to the message's content have already been processed in response to previously received SN_QUERY messages having as interested agent the same one specified in m; otherwise (i.e. if there are only some content elements already processed for that interested agent) these elements are deleted from the message and the function returns $false$. If the function $already\_answered(m)$ returns $true$, the behaviour is completed and a message with performative SN_ALREADY_ANSWERED is sent to s; otherwise, it continues as follows: First, the function $understanding(m)$ is executed. This function, for each content element e_i, $i = 1, 2, .., r$ contained in m, determines if e_i is an instance of some schema S_k, $k = 1, .., n$ belonging to the x's ontology. This check is performed by invoking, for each pair (e_i, S_k),$i = 1, .., r$, $k = 1, .., n$ the boolean function $semanticInstanceOf(e_i, S_k)$ that returns $true$ if e_i is (semantically) an instance of S_k.

   The function $semanticInstanceOf$ performs a schema matching between the schema of e_i and S_k, and can be implemented by using one of the several schema-matching methods existing in the literature as, for instance, those proposed in [9]. The function $understanding(m)$ for each content element e_i that matches with at least one of its schemas, inserts into a list $understood$ the pair (e_i, sl_i), where sl_i^l is the l-th schemas of the x's ontology matching with e_i, and inserts into another list $ununderstood$ each element e_i that does not matches with any of its schemas S_k; then, if the list $ununderstood$ is empty, the behaviour $semanticUnderstanding$ sends a message with performative SN_RESPONSE to the agent s, containing as content all the elements of the list $understood$; otherwise, if some elements are present into $ununderstood$, the behaviour $semanticNegotiation$ is executed for trying to understand the meanings of these elements.
2. When the $semanticNegotiation$ behaviour is executed, another function $createPartitions$ is firstly invoked. This function reads the e-coefficients (resp. u-coefficients) from $ECDB$ (resp. $UCDB$) and, on the basis of the partition weights set by the agent owner, determines the agent partitions. Then, SRequest and SReceive behaviours are executed. SRequest is a OneShotBehaviour that, for each partition level k, sends a message r to each agent

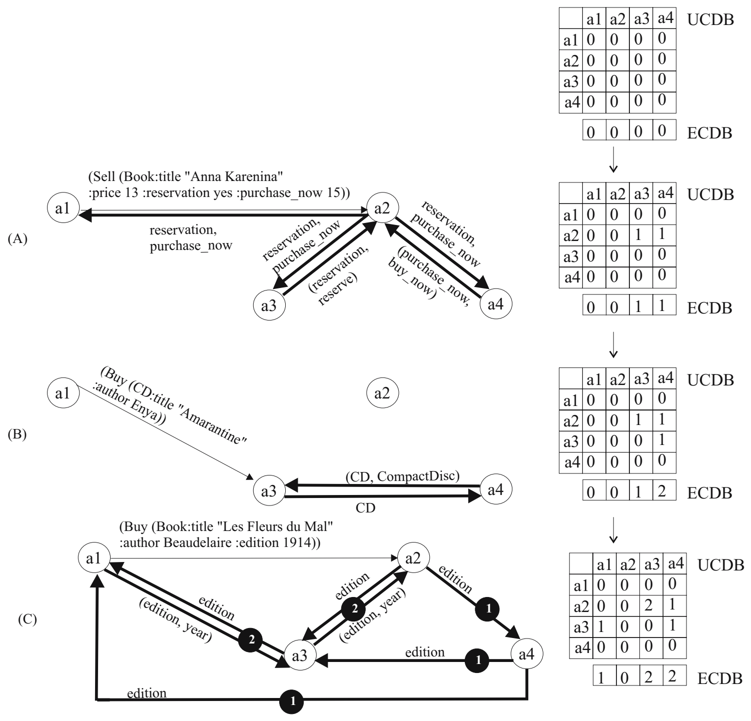
contained in the k-th partition, until either the list *ununderstood* becomes empty or a timeout t_1 is reached. r contains SN_QUERY as performative and the content element list *ununderstood* as content. SReceive is a CyclicBehaviour in which the agent x waits for messages containing a performative SN_RESPONSE, arriving from the contacted agents belonging to the $AS_x^1, AS_x^2, .., AS_x^k$. As said above, each received message m_a arriving from an agent a has as content a list of pairs $(e_1, sl_1), (e_2, sl_2), .., (e_h, sl_h)$ where e_i is a content element belonging to *ununderstood* and sl_i is a list $[s_i^1, s_i^2, .., s_i^l]$ of content elements synonyms of e_i, thus they are l possible meanings for e_i. Therefore, the function $solveSemanticUnunderstanding(e_i, s_i^g)$, $g = 1, 2, .., l$ is called for each pair (e_i, s_i^g): this function, if at least one s_i^g is an instance of some schemas belonging to the x's ontology, performs two operations: (i) deletes e_i from the list *ununderstood*, (ii) adds s_i^g to the list sl_i contained in *understood*.

Finally, the function *SUpdate* is called, that increases of one unit both the u-coefficient u_{xa} and the e-coefficient e_a.

5 An Application Example: Agents That Buy and Sell

In this Section, we present an application of the semantic negotiation technique we have previously described to the simple situation of a small e-commerce agent community, composed by four agents, denoted by $a1$, $a2$, $a3$, $a4$. Figure 2 shows the evolution of the community during three consecutive semantic negotiation stages represented in subfigures 2.A, 2.B and 2.C. In each subfigure, the global database $ECDB$ is represented by a row vector containing the four expertise coefficients e_{a1}, e_{a2}, e_{a3}, e_{a4}, associated to $a1$, $a2$, $a3$ and $a4$, respectively, while the four local databases $UCDB$ are synthetically represented by a matrix $UCDB$ where each element $UCDBij$ contains the u-coefficient u_{ij}. At the beginning, both the understanding capability and expertise coefficients are equal to 0. We also suppose that all the agents give the same importance both to the understanding capability and the expertise, therefore all the weights w_e and w_u are equal to 0.5. Furthermore, each subfigure represents each message sent by an agent i to an agent j by an arrow oriented from i to j. A thin line is used to represent ordinary messages, while a double line is exploited for the semantic negotiation messages. Each arc is labelled with the message's content. Due to layout reasons, we omit to represent the negotiation messages with performative SN_UNKNOWN or SN_ALREADY_ANSWERED.

In Fig. 2.A, we see that the agent $a1$ sends a PROPOSE message to $a2$, containing a predicate that says he desires to sell by auction a book having the title "Anna Karenina", with initial price equal to 13 US dollars, with a reservation price (i.e., the lowest price $a1$ accepts for selling the book, that is obviously secret), and with the possibility (represented by the element *purchase_now*) for a buyer to purchase immediately the book without participating to the auction, paying a price equal to 15 US dollars. The agent $a2$ receives the message, but it is unable to understand the terms *reservation* and *purchase_now*, since they are



	a1	a2	a3	a4
a1	0	0	0	0
a2	0	0	0	0
a3	0	0	0	0
a4	0	0	0	0

UCDB

0	0	0	0

ECDB

	a1	a2	a3	a4
a1	0	0	0	0
a2	0	0	1	1
a3	0	0	0	0
a4	0	0	0	0

UCDB

0	0	1	1

ECDB

	a1	a2	a3	a4
a1	0	0	0	0
a2	0	0	1	1
a3	0	0	0	1
a4	0	0	0	0

UCDB

0	0	1	2

ECDB

	a1	a2	a3	a4
a1	0	0	0	0
a2	0	0	2	1
a3	1	0	0	1
a4	0	0	0	0

UCDB

1	0	2	2

ECDB

Fig. 2. An example of Semantic Negotiation

not present in his ontology. Then, he decides to exploit the semantic negotiation protocol and, since both the understanding capability and expertise coefficients are equal to 0, the only agent partition that he can build is AS^0_{a2} containing all the other agents $a1$, $a3$ and $a4$. Suppose that when the timeout of $a2$ is reached (i) only $a3$ and $a4$ have sent a SN_RESPONSE message (ii) $a3$ proposes, as a synonym of *reservation*, the term *reserve* (iii) $a4$ provides the term *buy_now* for explaining the term *purchase_now*. Now, suppose the ontology of $a2$ contains both *reserve* and *buy_now*: in this case, it is now able to completely understand the message of $a1$ and to respond to it in an adequate way. Moreover, both the u-coefficients $u_{a2,a3}$, $u_{a2,a4}$, and the e-coefficients e_{a3}, e_{a4} become equal to 1.

The subfigure 2.B shows the agent $a1$ sending a PROPOSE message to $a3$, saying that he desires to buy a CD having title "Amarantine" of the author "Enya". However, $a3$ does not understand the term *CD* and thus he decides to exploit the semantic negotiation protocol. First, he builds the two partitions $AS^0_{a3} = \{a4\}$ and $AS^1_{a3} = \{a1, a2\}$ since $p(a4) = 0.5 \cdot 1 + 0.5 \cdot 0 = 0.5$ and $p(a1) = p(a2) = 0$. Then, $a3$ begins the semantic negotiation only with a_4 and receives a SN_RESPONSE message by this latter that explains that a synonym for *CD* is *CompactDisc*, that we suppose to be present in the ontology of $a3$.

Then, $a3$ can end the semantic negotiation process, since he is now able to understand the message of $a1$. As a consequence of this process, the e-coefficient e_{a4} becomes equal to 2 and the u-coefficient $u_{a3,a4}$ becomes equal to 1.

In the subfigure 2.C is depicted the next situation, in which the agent $a1$ sends to $a2$ a PROPOSE message, saying that he desires to buy a book with title "Les Fleurs du Mal", with author "Beaudelaire" and edition 1914. Since $a2$ does not understand the term *edition*, he decides to exploit the semantic negotiation protocol, and he first constructs the partitions $AS^0_{a2} = \{a4\}$, $AS^1_{a2} = \{a3\}$ and $AS^2_{a2} = \{a1\}$, since $p(a4) = 0.5 \cdot 1 + 0.5 \cdot 2 = 1.5$, $p(a3) = 0.5 \cdot 1 + 0.5 \cdot 1 = 1$ and $p(a1) = 0$. Then, $a2$ first asks the help of $a4$, but this latter is not able to autonomously provide an explanation for the term *edition*, then he sends a semantic negotiation message to both $a1$ and $a3$. The black circle labelled with 1 on the arc involved above means that all these arcs are related to the first attempt of negotiation of $a2$. Suppose that both these messages do not arrive to their destination due to a break of the connections $a4$-$a3$ and $a4$-$a1$. When the timeout of $a2$ for $a4$ is reached, $a2$ begins a new semantic negotiation with $a3$ that, in its turn, is not able to provide an explanation for the term *edition*, and thus he requires the help of $a1$ and $a4$. $a4$ is not able to be reached, due to the connection's break, while $a1$ responds with a synonym *year* for *edition*. This leads to set to 1 both the expertise e_{a1} and the understanding capability $u_{a3,a1}$. Now, $a3$ is able to send to $a2$ the explanation *year* for the term *edition* and, supposing *year* to be in the ontology of $a2$, the semantic negotiation of $a2$ can be terminated. All the arcs involved in this second negotiation tentative of $a2$ contains a black circle labelled with 2. As a consequence of the negotiation process, both e_{a3} and $u_{a2,a3}$ become equal to 2.

Now, observe the final situation represented in the tables $UCDB$ and $ECDB$ of the subfigure 2.C. The most "expert" agents are $a3$ and $a4$, and this is completely justified by the fact that they have solved for two ways semantic understanding's problems. The $UCDB$ rows corresponding to agents $a1$ and $a4$ have all their elements equal to 0, reflecting the fact that no other agents have helped them to understand any terms. The agent $a2$ has been helped 2 times by $a3$ and 1 time by $a4$, and this is represented by the corresponding values in the $UCDB$ row of $a2$. The agent $a3$ has been helped once by $a1$, and this is represented by the only one no zero coefficient in the $UCDB$ row of $a3$.

6 Conclusions

Semantic negotiation is a powerful framework for solving understanding problems among agents having personal ontologies that are not completely homogeneous. However, a key problem in semantic negotiation protocol is making the right choice of the agents with which it is most suitable to negotiate. In this work, we present a semantic negotiation protocol that makes effective the process of selecting the negotiation partners, by defining two measures, called expertise and understanding capability, that reflects two of the most important features that should be considered in making this selection, that are (i) the capability of an agent to respond to semantic negotiation answers arriving from whatever agent, representing the degree of expertise that the agent has in the community and (*ii*) the capability of an agent to respond to semantic negotiation answers arriving from a particular other agent, that defines the degree of comprehension that the former agent has with respect to the latter one. We define an agent negotiation protocol that allows to compute these measures by observing the results of the agent negotiation. Furthermore, we have implemented this protocol in the JAVA language as component of the middleware JADE, giving the possibility to use it for realizing JADE agents able to negotiate the semantic of the terms. Our ongoing research deals with the possibility of including in the protocol more sophisticated features as, for instance, the possibility that an agent gives a negative feedback when he receives a unsatisfactory response by another agent.

References

1. R.-J. Beun and R.M. van Eijk. A Cooperative Dialogue Game for Resolving Ontological Discrepancies. In *Workshop on Agent Communication Languages*, pages 349–363, 2003.
2. R.-J. Beun, R.M. van Eijk, and H. Prust. Ontological Feedback in Multiagent Systems. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 110–117, Washington, DC, U, 2004. IEEE Computer Society.
3. H. Ding and I. Sølvberg. Towards the schema heterogeneity in distributed digital libraries. In *ICEIS (5)*, pages 307–312, 2004.
4. D.W. Embley. Toward Semantic Understanding: An Approach Based on Information Extraction Ontologies. In *CRPIT '04: Proceedings of the fifteenth conference on Australasian database*, pages 3–12, Darlinghurst, Australia, Austra, 2004. Australian Computer Society, Inc.
5. T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, Maryland, USA, 1994. ACM Press.
6. http://www.fipa.org, 2005.
7. R. Guha. Semantic Negotiation: Co-identifying objects across data sources. In *AAAI '04 Spring Symposium Series: Proceedings of the Semantic Web Services*, March 2004.
8. http://www.jade.tilab.org, 2005.
9. E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, 2001.
10. C. Reed, T.J. Norman, and N.R. Jennings. Negotiating the Semantics of Agent Communication Languages. *Computational Intelligence*, 18(2):229–25, 2002.
11. J. van Diggelen, R.-J. Beun, F. Dignum, R.M. van Eijk, and J.-J.Ch. Meyer. Optimal communication vocabularies and heterogeneous ontologies. In R.M. van Eijk, M.-P. Huget, and F. Dignum, editors, *Developments in Agent Communication*, LNAI 3396. Springer Verlag, 2004.
12. A.B. Williams. Learning to Share Meaning in a Multi-Agent System. *Autonomous Agents and Multi-Agent Systems*, 8(2):165–193, 2004.

Appendix: The Package jade.hisene

In this Appendix we present a java implementation of the semanticUnderstanding behaviour (see Fig. 3) and the semanticNegotiation behaviour (see Fig. 4) as described in Section 4. These behaviours are part of the jade.hisene package that we are writing and which is in an advanced state of development. Due to the length of the code we don't present the private methods. However, they are of a simple implementation.

```
package jade.hisene;
import jade.core.*;
import jade.core.behaviours.OneShotBehaviour;
import jade.lang.acl.ACLMessage;
...

public class semanticUnderstanding extends OneShotBehaviour {
    private ACLMessage msg;
    private List understood, ununderstood;

    public semanticUnderstanding (Agent a, ACLMessage msg) {
        super(a);
        this.msg = msg;
    }

    public void action() {
        if (msg.getPerformative() == Semantic.SN_QUERY && alreadyAnswered(msg)){
            ACLMessage reply = msg.createReply();
            reply.setPerformative(Semantic.SN_ALREADY_ANSWERED);
            reply.setContent(msg.getContent());
            myAgent.send(reply);
        } else {
            understanding(msg);
            if (!ununderstood.isEmpty()) {
                ACLMessage sn_query_msg;
                sn_query_msg = setUnderstood(msg, understood);
                sn_query_msg = setUnunderstood(msg, ununderstood);
                sn_query_msg.setPerformative(Semantic.SN_QUERY);
                ((Semantic)parent).addSubBehaviour(new semanticNegotiation(myAgent, sn_query_msg));
            } else {
                ACLMessage reply = msg.createReply();
                reply.setPerformative(Semantic.SN_RESPONSE);
                reply.setContent(msg.getContent());
                reply = setUnderstood(reply, understood);
                myAgent.send(reply);
            }
        }
    }
    // Private Methods Section
    ...
}
```

Fig. 3. The semanticUnderstanding behaviour

```
package jade.hisene;
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.ACLMessage;
...

public class semanticNegotiation extends ParallelBehaviour{

    private ACLMessage msg;
    private Stack partitions;
    private Behaviour srequest = new SRequest(myAgent, msg, partitions);
    private Behaviour sreceive = new SReceive(myAgent, msg);

    public semanticNegotiation (Agent a, ACLMessage msg) {
        super(a, WHEN_ANY);
        this.msg = msg;
    }

    public void onStart() {
        createPartitions();
        addSubBehaviour(srequest);
        addSubBehaviour(sreceive);
    }

    public int onEnd() {
        removeSubBehaviour(sreceive);
        ACLMessage reply = msg.createReply();
        reply.setPerformative(Semantic.SN_RESPONSE);
        reply.setContent(msg.getContent());
        myAgent.send(reply);
        return 0;
    }
    // Private Methods Section
    ...
}
```

Fig. 4. The semanticNegotiation behaviour