Agents handling annotation distribution in a corporate semantic Web

Fabien Gandon

Agents handling annotation distribution in a corporate semantic Web

Author: Fabien Gandon, INRIA Sophia Antipolis - ACACIA Team
Address: INRIA - ACACIA Team, 2004, route des Lucioles - B.P.93, 06902 Sophia Antipolis France, **Phone:** [+33] (0)4 92 38 77 88, **Fax:** [+33] (0)4 92 38 77 83
E-mail: Fabien.Gandon@sophia.inria.fr
Abstract: This article describes a multi-agent software architecture to manage a corporate memory in the form of a corporate semantic Web. It summarizes the design rationale and the final architecture of the CoMMA system. It then presents and discusses an approach to manage distributed annotations, focusing on two problems: the allocation of newly posted annotations and the allocation of the tasks involved in distributed query-solving.
Keywords: multi-agent information systems, corporate memory, semantic Web, ontology

1 Introduction

The information technology explosion of the last decade led to a shift in the economy and market rules. Corporations had to adapt their organization and management, to improve their reaction and adaptation time. Information systems became backbones of organizations, enabling project-oriented management and virtual teams. Thus the industrial interest in methodologies and tools supporting capitalization and management of corporate knowledge grew stronger.

A corporate memory is an explicit, disembodied and persistent representation of knowledge and information in an organization, in order to facilitate their access and reuse by members of the organization, for their tasks [12]. The stake in building a corporate memory management system is the coherent integration of this knowledge dispersed in a corporation, with the objective to promote knowledge growth and knowledge communication within an organization [29].

ACACIA, my research team, was part of the European project CoMMA aiming at implementing a corporate memory management framework based on several emerging technologies: agents, ontology and knowledge engineering, XML, information retrieval and machine learning techniques. The project intended to implement this system in the context of two scenarios: (a) assisting the insertion of new employees in the company and (b) supporting the technology monitoring process. The technical choices of CoMMA were mainly motivated by three observations:

(1) A corporate memory is, by nature, an heterogeneous and distributed information landscape. Corporate memories are now facing the same problem of information retrieval and overload as the Web. The initiative of a semantic Web [5] is a promising approach where the semantics of documents is made explicit through ontology-based annotations to guide later exploitation. XML being likely to become an industry standard for exchanging data, we used it to build and structure the corporate memory; more precisely the Resource Description Framework (RDF) with its XML syntax allowed us to semantically annotate resources of a corporate memory and envisage it as a *corporate semantic Web*.

(2) The population of users of the memory is, by nature, heterogeneous and distributed in the corporation. Some agents can then be dedicated to interface users with the system. Adaptation and customization are a keystone and CoMMA relies on machine learning techniques in order to make agents adaptive to users and context.

(3) Tasks to be performed on corporate memories are, by nature, distributed and heterogeneous. Both the corporate memory and its population of users are distributed and heterogeneous. Therefore, it seemed interesting that the interface between these two worlds be itself heterogeneous and distributed. Programming progresses were achieved through higher abstraction enabling us to model systems more and more complex. Multi-agent systems (MAS) are a new stage in abstraction that can be used to understand, to model and to develop a whole new class of distributed systems [32]. MAS paradigm is well suited for designing software architectures to be deployed above distributed information landscapes: on the one hand, individual agents locally adapt to users and resources they are dedicated to ; on the other hand, cooperating agents enable the whole system to capitalize an integrated view of the corporate memory.

In section 2, is summarized the design rationale and the final architecture of the CoMMA system. Then, section 3 focuses on the problems caused by the distribution of the annotations structuring the corporate semantic Web. We shall see how some aspects of the conceptual foundations of the

semantic Web can support a multi-agent system in allocating and retrieving semantic annotations in a distributed corporate memory. In particular we shall see how agents exploit the underlying graph model, when deciding how to allocate new annotations and when resolving distributed queries. Two scenarios are presented here:

- *allocation of newly posted annotations*: section 3.3 shows how agents use the contract-net protocol together with a pseudo semantic distance between a new annotation and an archive of annotations for choosing the base where to store the annotation.
- *allocation of the tasks involved in solving a query*: section 3.4 shows how agents use the nested query-ref protocol together with a description of the overlap between the query needs and the statistics over an archive of annotations for distributing sub-tasks in the form of sub-queries.

2 CoMMA: Corporate Memory Management through Agents

2.1 Corporate semantic Web: an annotated world for agents

The field of multi-agent information systems is very active and most promising application areas are, among others, distributed Web-based collaborative work, information discovery in heterogeneous information sources and intelligent data management in the Internet or corporate intranets [22]. Some of the multi-agent information systems are specialized in information retrieval from heterogeneous information repositories. InfoMaster [20] uses a global schema and Carnot [9] a global ontology (Cyc) to build mappings for wrappers of heterogeneous sources. As in RETSINA [11], these systems rely on wrapper agents to provide an homogeneous view of the different sources while the integration is handled by middle agents planning query resolution, information integration and conflict resolution. Information Manifold [24] and InfoSleuth [26] have multiple ontologies but they do not handle mapping between them. SIMS [2] uses Description Logics to handle multiple ontologies and translate queries when there is no loss. Finally OBSERVER [25] takes into account the inter-ontology relationships to tackle the loss of information when translating queries. Another kind of system is dedicated to the management of digital libraries. It is represented, for instance, by SAIRE [27] and UMDL [31] that manage distributed large scale libraries of digital documents to offer means to find relevant documents and manage indexing. Finally some projects focus on knowledge management inside organizations. CASMIR [6] and Ricochet [7] focus on gathering information and adapting interactions to the user's preferences, learning interests to build communities and enable collaborative filtering inside an organization. KnowWeb [14] relies on mobile agents to support dynamically changing networked environment and exploits a domain model to extract concepts describing a documents and use them to answer queries. RICA [1] maintains a shared taxonomy in which nodes are attached to documents and uses it to push suggestions to interface agents according to user profiles. Finally FRODO [30] is dedicated to building and maintaining distributed organizational memories with an emphasis on the management of domain ontologies.

The CoMMA project presented here, belongs to this last category. It aimed at implementing and testing a corporate memory management framework based on agent technology. Thus, CoMMA did not target the Internet and the open Web but corporate memories accessible through an intranet based on Web technologies *i.e.* an intraweb. The system does not directly manage documents, but annotations about documents. We suppose documents are referenced by URI and we index them using semantic annotations relying on the semantic Web technologies

The article "Agents in Annotated Worlds" [13] shows that "annotated environments containing explanations of the purpose and uses of spaces and activities allow agents to quickly become intelligent actors in those spaces". This remark is transposable to information agents in complex information worlds: *annotated information worlds are, in the actual state of the art, a quick way to make information agents smarter*. If a corporate memory becomes an annotated world, agents can use the semantics of annotations and through inferences help users exploit its content.

RDF [23] uses a simple triple model and an XML syntax to represent properties of Web resources and their relationships. It makes no assumption about a particular application domain. *With RDF, we describe the content of documents through semantic annotations* and then use and infer from these annotations to successfully search the mass of information of the corporate memory. Just as an important feature of multi-agent systems is the ability to integrate legacy systems, an important feature of a corporate memory management framework is the ability to integrate the legacy

archives. An RDF annotation being either internal or external to the resources, existing documents may be kept intact and annotated externally.
Compared to the Web, a corporate memory has more delimited and defined context, infrastructure and scope: the corporation. In a corporate context we can more precisely identify stakeholders and the corporate community shares some common global views of the world. Thus an ontological commitment is conceivable to a certain extent. A methodology was proposed and tested to build *O'CoMMA* (Ontology of CoMMA) [17] on which is based the descriptions of the organizational state of affairs, of the users' profile and the annotations of the memory resources. O'CoMMA is formalized and shared thanks to RDF Schema (RDFS) [8] which is related to object models but with the properties being defined separately. Figure 1 shows a sample of RDF(S): the formalization of a hierarchy of concepts and properties and an example of annotation with literal and conceptual properties. Current keyword-based search engines are limited to terms denoting extensions of concepts; the introduction of ontologies enables software to access the intensional level. O'CoMMA is the keystone of the system: it is a full resource of the memory and it provides the building blocks for models, annotations and agent messages, with their associated semantics. To manipulate and infer from the ontology and annotations ACACIA developed CORESE [10] a prototype of a search engine enabling inferences on RDF by using the query and inference mechanisms available in the Conceptual Graphs formalism.

Figure 1 RDF(S) Sample

An *enterprise model* is an oriented, focused and somewhat simplified explicit representation of the organization. So far, the enterprise modeling field has been mainly concerned with simulation and optimization of the production system design, but lately enterprises realized that enterprise models have a role to play in their information system also. In CoMMA, the corporate model gives the system an insight in the organizational context and environment to tune its interactions and reactions. It is materialized as RDF annotations about the organization.
Likewise, the *users' profile* captures all aspects of the user that were identified as relevant for the system behavior. It contains administrative information and preferences that go from interface customization to topic interests. It positions the user in the organization: role, location and potential acquaintance network. In addition to explicitly stated information, the system derives information from past usage by collecting the history of visited documents and possible feedback from the user. From this, agents learn some of the user's habits and preferences [21]. These learnt criterions are used for interfaces or information push.

2.2 Architecture

Information agents are part of the intelligent agents. A MAS is a loosely coupled network of agents that work together as a society aiming at solving problems that would generally be beyond the reach of any individual agent. A MAS is heterogeneous when it includes agents of at least two types. A Multi-agent Information System (MAIS) is a MAS aiming at providing some or full range of functionalities for managing and exploiting information resources. The application of MAIS to corporate memories means that the cooperation of the agents aims at enhancing information capitalization in the organisation. *The CoMMA software architecture is an heterogeneous MAIS.*

The MAIS *architecture is a structure that portrays the different families of agents and their relationships*. A *configuration is an instantiation of an architecture with a chosen arrangement and an appropriate number of agents of each type*. One given architecture can lead to several configurations. In the case of a corporate memory, a given configuration is tightly linked to the topography and context of the place where it is deployed (organizational layout, network topography, stakeholders location), therefore it must adapt to this information landscape and change with it. The architecture must be designed so that the set of possible configurations covers the different corporate organizational layouts foreseeable. The configuration description is studied and documented at deployment time using adapted UML deployment diagrams to represent, hosts (servers, front-end...), MAS platforms, agent instances and their acquaintance graph. The architectural description is studied and fixed at design time. The architectural analysis starts from the highest level of abstraction (*i.e.* the society) and by successive refinements (*i.e.* nested sub-societies), it goes down to the point where the needed agent roles and interactions can be identified.
Our approach to design the CoMMA architecture shares with approaches like A.G.R. used in AALAADIN [15] or GAIA [32], the concern for an organizational approach where the MAS architecture is tackled, as in a human society, in terms of roles and relationships. The functional requirements of the system do not simply map to some agent functionalities but influence and are finally diluted in the dynamic social interactions of individual agents and the set of abilities, roles and behaviors attached to them. Considering the system functionality, we identified four dedicated sub-societies of agents as shown in figure 2 : (1) Sub-society dedicated to ontology and model (2) annotations-dedicated sub-society (3) User-dedicated sub-society (4) Connection-dedicated sub-society.

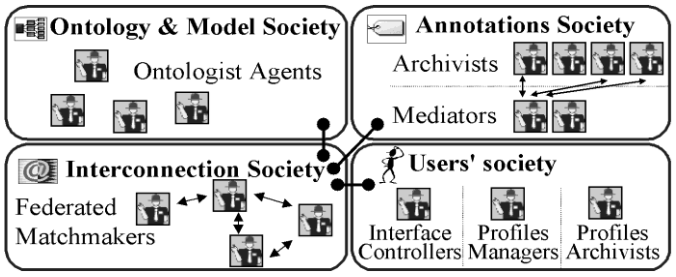

Figure 2 Sub-societies of CoMMA

Analyzing the resource-dedicated sub-societies (ontology, annotations and yellow pages for interconnection), we found that there was a recurrent set of possible organizations for these sub-societies: *hierarchical, peer-to-peer*, and *replication*. As discussed in [19] every organization has advantages and disadvantages ; depending on the type of tasks to be performed, the size and complexity of the resources manipulated, a sub-society organization will be preferred to another.
The agents from the *sub-society dedicated to the ontology and model* are concerned with the ontology and model exploitation during information retrieval activities and especially the queries about the hierarchy of concepts and the description of the organization where the system was deployed. Thus, they provide downloads, updates and querying mechanisms for other agents. For this sub-society, the three types of organization are conceivable. CoMMA implemented a replication society where each agent have a complete copy of the ontology/model and can resolve queries by itself. It is acceptable since, in the CoMMA prototype, the ontological commitment is centralized and the global ontology is updated and propagated over the agent society. Other options are interesting if the ontology/model is large or changes quite often and if a distributed mechanism in the MAS must support the consensus process as in FRODO [30].
The agents from the *annotation dedicated sub-society* are concerned with the exploitation of annotations structuring the corporate memory, they search and retrieve references matching users' queries. Here, only the hierarchical or the peer-to-peer society are conceivable: a replication society is not realistic since it would imply to replicate a full copy of the corporate memory for each resource agent. As we shall see in details latter, CoMMA implemented a hierarchical organization.
The agents from the *connection dedicated sub-society* are in charge of the matchmaking of the other agents based upon their respective needs and roles descriptions. CoMMA is implemented with JADE [3], an open source MAS platform compliant with the FIPA [16] specifications, that provides a Directory Facilitator Agent type. These agents are federable matchmakers organized in a peer-to-peer society managing the Yellow Pages.
The agents from the *user dedicated sub-society* are concerned with the interface, the monitoring, the assistance and the adaptation to the user. Because they are not related to a resource type like the previous ones, they cannot be studied using the typology we defined. We distinguished two

recurrent roles in this type of sub-society: (1) the user interface management: to dialogue with the users to enable them to express their request, to refine them and to present results in a comprehensive format (2) the management of user's profile: to archive and make the profiles available to other agents. More details are given in [18].

2.3 Roles, interactions and behaviors

From the architecture analysis we derived the characteristics of the identified roles, their interactions and finally we implemented the corresponding behaviors in a set of agent types.

Roles represent the position of an agent in a society and the responsibilities and activities assigned to this position and expected by others to be fulfilled. In the design junction between the micro-level of agents and the macro-level of the MAS, the role analysis is a key step. The previous part identified the following roles which characteristics are detailed in [19]:

- the *Ontology Archivist (OA)* maintains and accesses the ontology.
- the *Corporate Model Archivist (CMA)* maintains and accesses the enterprise model.
- the *Annotation Archivist (AA)* maintains and accesses an annotation repository.
- the *Annotation Mediator (AM)* manages and mediates among a set of Annotation Archivists.
- the *Directory Facilitator (DF)* maintains and accesses the yellow pages.
- the *Interface Controller (IC)* manages and monitors a user interface.
- the *User Profile Manager (UPM)* manages updates of profiles of users logged nearby.
- the *User Profile Archivist (UPA)* stores and retrieves users' profiles.

Following the role identification comes the specification of role *interactions*. Interactions consist in more than the sending of an isolated message. The conversation pattern needs to be specified with protocols and the agents must follow them for the MAS to work properly. Protocols are codes of correct behavior in a society for agents to interact with others. They describe a standard procedure to regulate information transmission between agents and institutionalize patterns of communication occurring between identified roles. The definition of a protocol starts with an acquaintance graph at role level, that is a directed graph identifying communication pathways between agents playing the considered roles. From that, we specify the possible sequences of messages. The acquaintance connections among the roles and the protocols adopted derive from both the organizational analysis and the use cases dictated by the application scenarios. The acquaintance graphs and the ACL message traces are depicted [19] using protocol diagrams [4], a restriction of the UML sequence diagrams, proposed within the AUML[1] initiative.

From the role and interaction descriptions the different partners of CoMMA proposed and implemented agent types that fulfill one or more roles. The *behavior* of an agent type combines behaviors implemented by the designers to accomplish the activities corresponding to the assigned roles. The behaviors come from the implementation choices determining the responses, actions and reactions of the agent. The implementation of the behavior is subject to the toolbox of technical abilities available to the designers, for instance, modules of the CORESE [10] search engine have been integrated in the behavior of the agents dedicated to the ontology, the models and the annotations. The implementation of CoMMA relying on JADE, the agent communication language is FIPA ACL, based on the speech act theory, which comes with standard protocols to be used or extended ; messages are encoded en RDF.

3 Annotations distribution

The submissions of queries and annotations are generated by agents from the user-dedicated society and routed to the annotation-dedicated society. The latter is a hierarchical society and, after presenting the problematics of distribution, we shall see how the allocation of new annotations and the solving of a query are handled in this society.

3.1 Issues of distribution

The duality the word 'distribution' reveals two important problems to be addressed :

- Distribution means dispersion, that is the *spatial property of being scattered about*, over an area or a volume; the problem here is to handle the naturally distributed data, information or knowledge of the organization.

[1] Agent Unified Modelling Language http://www.auml.org.

- Distribution also means the act of 'distributing or *spreading* or apportioning' ; the problem then is to make the relevant pieces of information go to the concerned agent (artificial or human). It is with both purposes in mind that we designed this sub-society.

From distributed database systems [28], we know that two types of fragmentation may be envisaged: *horizontal* and *vertical*. By drawing a parallel between data/schema and knowledge/ontology, we may say that in our case these two types of fragmentation would correspond to:

- *Horizontal fragmentation*: information is split according to the range of properties. For instance $site_1$ will store all the reports with a property *title* ranging from "Alarming criminality in agent societies" to "Mass errors in behavioral control" and $site_2$ will store all the reports ranging from "Naive implementation of resource distribution" to "Zeno paradox in iterative loops".
- *Vertical fragmentation*: information is split according to types of concepts and properties. For instance, $site_1$ will store all the *reports* with their *titles* and *authors* and $site_2$ will store all the *articles* with their *abstracts* and *keywords*.

Knowing this, the stake is to find mechanisms to decide *where to store newly submitted annotations* and *how to distribute a query* in order not to miss answers just because the needed information is split over several annotation archives. These two facets are linked since the performance of distributed query resolution is closely related to the choices made for the distribution of annotations.

The two agent types involved in the management and exploitation of the distributed annotations are:

- the *Annotation Archivist* (*AA*) is in charge of saving and querying the annotations of the corporate memory with its local resources. The *AA* is known by and communicates with its *AM*.
- the *Annotation Mediator* (*AM*) is in charge of solving queries on the memory and allocating new annotations to the best suiting *AA*. The *AM* is thus in charge of getting in touch and managing the agents involved in the resolution process, in particular its *AAs*, and other *AMs*.

They form the annotation-dedicated society, in charge of handling annotations and queries in the distributed memory. Both queries and annotation submissions are generated by agents from the user-dedicated society and routed to the annotation-dedicated society. The latter is a hierarchical society: the *AMs* are in charge of the *AAs*. An *AM* provides its services to other societies to solve their queries and, to do so, it requests the services of the *AAs*. On the other side, an *AA* is attached to a local annotation repository and when it receives a request, it tries to fulfil it with its local resources in a fashion that enables the *AM* to handle the distributed dimension of the problem. The agents playing the role of *AA* and *AM* are benevolent and, once deployed, temporally continuous.

3.2 Differentiating archivist agents

In order to determine which *AA* should be involved during the solving of a query or to which one an annotation should be given, an *AM* compares the content of the archives of the *AAs* thanks to a light structure called *ABIS* (Annotation Base Instances Statistics). As shown in Table 1, the *ABIS* captures statistics, maintained by the *AA*, on the population of triples of its annotation base:

- The number of instances for each concept type; if a concept type of the ontology is missing in that list, it means there is no instance of that concept type in the annotation base.
- The number of instances for each property type; if a property type of the ontology is missing in that list, it means there is no instance of that property type in the annotation base.
- the number of instances for each family of properties (as explained below).

A family of properties is defined by a specialized signature corresponding to at least one instance present in the archivist base $[\text{ConceptType}_x] \rightarrow (\text{PropertyType}_y) \rightarrow [\text{ConceptType}_z]$ where the concept types are possibly more precise than the signature of $PropertyType_y$. For instance, if there exists a property *Author* with the signature [Document] → (Author) → [Person] we may have families of properties such as [Article] → (Author) → [Student] or [Book] → (Author) → [Philosopher].

This means that for each of these specialized signatures, there exists, in the archive of the corresponding *AA*, at least one instance using exactly these types. If a family does not appear in the *ABIS*, it means there is no instance of this very precise type.

Concept	**Population**
Report	57

...

Relation	**Population**
Concern	23
Firstname	89

Manage	69

...

Relation	Population	Upper Bound	Lower Bound
Manager → Manage → Worker	12		
Manager → Manage → Employee	57		
Employee → Firstname → Literal	89	Alain	Laurent
Article → Concern → Web	23		

...

Table 1 ABIS (Annotation Base Instances Statistics)

The *ABIS* is built when annotations are loaded by the *AA*; the agent decomposes the corresponding Conceptual Graphs generated in CORESE into binary relations and identifies the possible isolated vertices to include them in the statistics. For literal properties, the bounding interval $[B_{low}, B_{up}]$ of their literal values is calculated. The *ABIS* is updated each time an annotation is loaded in the base and after applying rules, transitivity and symmetry completion to the base. The only problem that remains in the current prototype is to take into account reflexive properties: the reflexivity is calculated dynamically during the projection process and therefore it is not taken into account by basic statistics on the base content, however this does not disturb the current algorithms and can therefore be overlooked in the current prototype.

To reuse the terminology of distributed databases, the *ABIS* structure captures information about the vertical and horizontal contribution of an Annotation Archivist to the overall distribution. The *ABIS* captures the types of concepts and relations for which an *AA* contributes to the memory. It does not capture the structural information of the annotations but it is enough to get an idea of the type of knowledge an archive keeps for the memory; it is a way to compare the specialization of the *AAs* in terms of the content of their base.

When a system is deployed, *AAs* are started but they may have no annotation in their bases. Their statistics being void, the *ABIS* is not relevant to compare their bids. Moreover, when deploying the system, it is interesting to be able to specialize individual agents according to the topography of the company network (e.g. an *AA* on a machine of Human Resources department for users' profile, another for scientific documents in each research group, etc.). As shown in Table 2, the *CAP* (Card of Archives Preferences) is a light structure that captures the RDF properties for which the agent has a preference and, if specified, their range boundaries. Any specialization of these properties is then considered to be part of the preferences of the *AA*. The *CAP* can be used to initiate a base and keep a trace of its initial specialization. These fragmentation choices are made by the administrators when deploying the agents.

Relation	Upper Bound	Lower Bound
Report → Title → Literal	A	L
Article → Abstract → Literal	A	Z
Article → Concern → Computer Science		
Page Web → Concern → Computer Science		
Person → Name → Literal	M	R

Table 2 Card of Archive Preferences

3.3 Annotation allocation

This section presents how agents allocate newly posted annotations. An important hypothesis is that agents do not break up annotations, they store them as one block. It explains how agents use the contract-net protocol together with a pseudo-distance between a new annotation and an archive of annotations for choosing the base where to store the annotation.

3.3.1 Allocation protocol

Agents involved in the allocation of the new annotation are: the *IC*, through which the annotation is built and submitted ; the *UPM*, that observes and forwards the annotation ; the *AM*, that manages the allocation process ; the *AA*, that manages local annotation archives. The *IC*, and the *UPM* are essentially involved in the emission of the new annotation and will not be considered any further here. Although the agents considered in CoMMA are benevolent and fully collaborating, the lack of individual interest does not mean that a situation like a market or a negotiation cannot be imagined. Indeed the problem of the allocation of an annotation can be seen as a market where each *AA* offers its services and where the *AM* in charge of allocating a new annotation is looking for the best 'contract'. There exist two options during the evaluation process:

- *AMs* send the newly posted annotation to *AAs* which send back a bid; the best one wins the bid.
- *AAs* send their *ABIS* and *CAP* to *AMs*. Then *AMs* calculate the appropriateness of each *AA* when a new annotation is posted and proceed to contact the best *AA*.

There are pros and cons for each option. In the first option, the *AAs* must be trustworthy, must use the same calculation method for bids but agents do not need to exchange *ABIS* and *CAP* each time they change and only the result of the bid calculation will be exchanged. In the second option, the *AMs* ensure a fair and exact evaluation of the relevancy of the *AAs* with their own measure, but the *ABIS* and *CAP* will have to be exchanged and the bid calculation will be centralized.

CoMMA agents are benevolent, non-antagonistic and fully co-operative due to the fact that they deal with a corporate semantic intraweb memory. Therefore the first option seemed to be the best : the *AMs* deal with the comparison of the different bids of the *AAs* and grant the annotation to the best one while the *AAs* evaluate the similarity between the newly posted annotation and their base and send back this evaluation to the *AMs*.

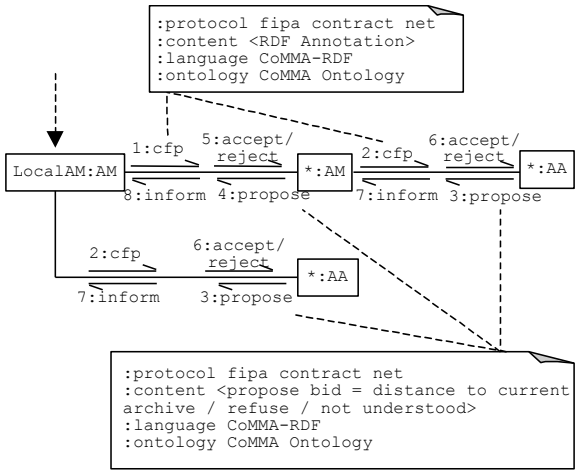

Figure 3 Acquaintance and protocol diagrams in annotation allocation

As shown in Figure 3, the protocol is a nested Contract-net with two levels: *local AM* → *local AAs* & [*other AMs* & → *local AAs*]. This diagram corresponds to the following scenario description:

1. The *IC* builds a well-formed and valid annotation using the O'CoMMA ontology.
2. The *IC* sends a request to the *UPM* it is acquainted with in order to archive the annotation.
3. The *UPM* contacts an *AM* called the *localAM*. It is the first one found by the *DF*, but it could be chosen using more complex criteria such as the workload.
4. The *LocalAM* calls upon the *DF* to look for other *AMs*.
5. The *LocalAM* sends a call for proposal to all the other *AMs*.
6. The *LocalAM* and the other *AMs* send a call for proposal to their *AAs*.
7. Each *AA* sends back the result of its bid calculation function to its *AM*.
8. The *LocalAM* and the other *AMs* choose the best candidate among their *AAs* and send a refusal to the others.
9. The other *AMs* send the result of their best candidate to the *LocalAM*.
10. The *LocalAM* compares results and grants the annotation to the best proposal and informs the others that their proposals are refused.
11. The *AMs* losers forward refusal to their best *AA*.
12. If the best proposal is an other *AM*, then this *AM* grants the annotation to its best *AA*, otherwise it is directly granted to the best *AA* of the *LocalAM*.
13. The *AA* winner adds the annotation to its base and informs its *AM* of the addition, if need be the *AM* informs the *LocalAM* of the addition.
14. The *LocalAM* forwards acceptation to the *UPM*.
15. The *UPM* forwards acceptation to the *IC*.

The whole process relies on the ability to bid and compare bids. This will be detailed in the following sections.

3.3.2 Pseudo semantic distance

In order to allocate a newly posted annotation, we need to measure how close it is from the *ABIS* and *CAP* of an *AA* to decide which *AA* should win the bid. Therefore we need to measure the distance between an annotation and a base, and to compare the different distances measured. The

calculation of this adequacy function or distance can take into account: the *CAP*, the *ABIS*, the resources available on the machine (CPU, hard disk space, etc.), the number of annotations already archived, some randomization in order to ensure uniform distribution, etc. We shall see thereafter the different steps to calculate the distance currently used in the prototype.

3.3.2.1 Definition of constants

This short section defines the constants at play in the formula proposed for the distance. $Max_L = 256$ is the maximum range for an ASCII byte code of a character. Max_C is the maximum path length in the subsumption hierarchy of the primitive concept types from the root to a leaf. It is calculated by recursive exploration of the subsumption hierarchy and it is usually called the depth of the ontology. Max_R is the depth of the ontology for the relation types. $N = Max_C \times 2 / Max_L$ is the constant used to normalize the distances when combining distances on literals and distances on primitive types. $W_C=4$, $W_R=8$ and $W_L=1$ are weights respectively for concept types, conceptual relation types and literals. They are used to balance the importance of these factors in the distance calculations.

3.3.2.2 Lexicographical distance between two literals

Some properties have a literal range type that we need to compare. Let $C_{X,i}$ the ASCII byte code of the i^{th} character in upper case ($C_{X,i} \in [0,\mathrm{Max_L}[$) of $Lit_X=C_{X,0},C_{X,1},C_{X,2},C_{X,3},...,C_{X,s}$ a literal coded by the sequence of length $s+1$ of the codes of its characters. Let :

$$\mathrm{Abscissa}(\mathrm{Lit_X}) = \sum_{i=0..s} \frac{C_{\mathrm{x,i}}}{\mathrm{Max_L}^{\,i}} \tag{1}$$

This *Abscissa* is positive or null since character codes are themselves positive or null:

$$\mathrm{Abscissa}(\mathrm{Lit_X}) \geq \sum_{i=0..l} \frac{0}{\mathrm{Max_L}^{\,i}} = 0 \tag{2}$$

The *Abscissa* is bounded by *B* with:

$$\mathrm{B} = \sum_{i=0..l} \frac{\mathrm{Max_L}-1}{\mathrm{Max_L}^{\,i}} = (\mathrm{Max_L}-1) \cdot \sum_{i=0..l} \frac{1}{\mathrm{Max_L}^{\,i}} \tag{3}$$

We recognize the sum of a finite geometric sequence.

$$B = (\mathrm{Max_L}-1) \cdot \frac{1-\left(\frac{1}{\mathrm{Max_L}}\right)^{l+1}}{1-\left(\frac{1}{\mathrm{Max_L}}\right)} = \mathrm{Max_L} - \frac{1}{\mathrm{Max_L}^{\,l}} < \mathrm{Max_L} \tag{4}$$

Thus $Abscissa(Lit_X) \in [0,Max_L[$, which explains the value of *N*. Lets now consider the difference :

$$D = \mathrm{Abscissa}(\mathrm{Lit_B}) - \mathrm{Abscissa}(\mathrm{Lit_A}) \tag{5}$$

where Lit_A and Lit_B are two literals. If they are the same, their abscissas are equal and $D=0$. Else let Lit_A come before Lit_B in alphabetical order. This means that the first difference in reading these strings is a character in Lit_A that comes alphabetically before the character at the same position in Lit_B. This can be formalized as : If $Lit_A < Lit_B$ then $\exists i \in [0..s]$ such that $\forall j < i \; C_{A,j}=C_{B,j}$ (*i.e.* the strings may have a common beginning) and $C_{A,i}<C_{B,i}$ *i.e.* the first characters that are different, follow the same alphabetical order as the two strings they belong to. The value of *D* (complementing the shortest string with characters of code 0 if necessary, so that both strings have the same length) is given by:

$$D = \sum_{\mathrm{k=0..s}} \frac{C_{B,k}-C_{A,k}}{\mathrm{Max_L}^{\,k}} = \frac{C_{B,i}-C_{A,i}}{\mathrm{Max_L}^{\,i}} + \sum_{\mathrm{k=i+1..s}} \frac{C_{B,k}-C_{A,k}}{\mathrm{Max_L}^{\,k}} \tag{6}$$

The absolute value of the remaining sigma sum is bounded by:

$$B' = \sum_{\mathrm{k=i+1..s}} \frac{\mathrm{Max_L}-1}{\mathrm{Max_L}^{\,k}} = \frac{\mathrm{Max_L}-1}{\mathrm{Max_L}^{\,i+1}} \cdot \sum_{\mathrm{k=0..s-i-1}} \frac{1}{\mathrm{Max_L}^{\,k}} \tag{7}$$

Here again we recognize the sum of a finite geometric sequence:

$$B' = \frac{\mathrm{Max_L}-1}{\mathrm{Max_L}^{\,i+1}} \cdot \frac{1-\left(\frac{1}{\mathrm{Max_L}}\right)^{s-i}}{1-\left(\frac{1}{\mathrm{Max_L}}\right)} = \frac{1-\left(\frac{1}{\mathrm{Max_L}}\right)^{s-i}}{\mathrm{Max_L}^{\,i}} < \frac{1}{\mathrm{Max_L}^{\,i}} \tag{8}$$

Therefore, as $C_{A,i} < C_{B,i}$ we proved that:

$$\frac{C_{B,i} - C_{A,i}}{\mathrm{Max_L}^{i}} \geq \frac{1}{\mathrm{Max_L}^{i}} > \left| \sum_{k=i+1..s} \frac{C_{B,k} - C_{A,k}}{\mathrm{Max_L}^{k}} \right| \quad (9)$$

And we can conclude that *D>0*, which means that

$$\mathrm{Lit_A} < \mathrm{Lit_B} \Leftrightarrow \mathrm{Abscissa(Lit_A)} < \mathrm{Abscissa(Lit_B)} \quad (10)$$

Based on the abscissa we define an Euclidean distance:

$$\mathrm{Dist_L(Lit_A, Lit_B)} = |\ \mathrm{Abscissa(Lit_B)} - \mathrm{Abscissa(Lit_A)}\ | \quad (11)$$

To avoid imprecision problem, the distance is implemented following the mathematically equivalent formula (12) instead of (11) where rounding of abscissa calculation is cumulated.

$$\mathrm{Dist_L(Lit_A, Lit_B)} = \left| \sum_{i=0..s} \frac{C_{\mathrm{B,i}} - C_{\mathrm{A,i}}}{\mathrm{Max_L}^{i}} \right| \quad (12)$$

As an example, if Lit_A = *"abandon"* and Lit_B = *"accent"* then:

- $Abscissa(Lit_A) \sim 65.25880898635597$
- $Abscissa(Lit_B) \sim 65.26274521982486$
- $Dist_L(Lit_A,Lit_B) \sim 0.0039362334688917144$

This distance is currently used in the system but a lexicographical distance cannot be qualified of a semantic distance. The overall distance used for bidding is a combination of this lexicographical distance and the following semantic distance; thus the overall distance is only said to be pseudo-semantic, and we shall see in the perspectives what can be envisaged to improve that point.

3.3.2.3 Pseudo-distance literal - literal interval

The *ABIS* and *CAP* provide bounding interval for literal properties. The pseudo-distance between a literal value Lit_X from an annotation and a range $[B_{low}, B_{up}]$ from those structures is given by:

$$\mathrm{Dist_I(Lit_X, [B_{low}, B_{up}])} = 0 \text{ if } \mathrm{Lit_X} \in [\mathrm{B_{low}, B_{up}}] \text{ else } = \mathrm{Min(Dist_L\ (Lit_X, B_{low}), Dist_L\ (Lit_X, B_{up}))} \quad (13)$$

This is only a pseudo-distance since it is not an application from Literal×Literal to $\Re^+$ but from Literal×[Literal,Literal] to $\Re^+$. For this reason the overall distance is only a pseudo-distance and likewise we shall see in the perspectives what can be envisaged to improve that point.

3.3.2.4 Distance between two ontological types

To compare two primitive types, the distance uses the length, in number of subsumption links, of the shortest path between these types in the hierarchies of their supertypes. The calculation of this distance is a problem equivalent to searching the least common supertype and the two distances from this supertype to the considered types:

$$\mathrm{Dist_H(T_1,T_2)} = \mathrm{SubPath(T_1,LCST)} + \mathrm{SubPath(T_2,LCST)} \quad (14)$$

where *LCST* is the Least Common Super Type of T_1 and T_2 and *SubPath(T,ST)* is the length, in edges, of the subsumption path from a type *T* to one of its supertypes *ST*. *LCST* is calculated by a recursive synchronized exploration of the ancestors of T_1 and T_2. This distance measures a semantic closeness since the least common supertype of two types captures what these types have in common. $\mathrm{Dist_H}$ complies to the four features of distances:

First: the distance from a type to itself is null.

$$\mathrm{Dist_H(T_1,T_1)} = 0 \quad (15)$$

Proof: the least common super-type of (T_1,T_1) is T_1 and the shortest path to go from a node to itself is not to cross an arc.

Second: the distance is symmetric.

$$\mathrm{Dist_H(T_1,T_2)} = \mathrm{Dist_H(T_2,T_1)} \quad (16)$$

Proof: the considered path is not directed and therefore the shortest path from T_1 to T_2 is also the shortest path from T_2 to T_1.

Third: a null distance can only be obtained if the two types are merged:

$$\mathrm{Dist_H(T_1,T_2)} = 0 \Rightarrow \mathrm{T_1=T_2} \quad (17)$$

Proof: since the only way to have a null distance is not to cross an arc, it is only possible if the two nodes at the extremity of the path are merged.

Fourth: the distance between two points is lesser than or equal to the distance of a path going through a third type.

$$\mathrm{Dist_H(T_1,T_3)} \leq \mathrm{Dist_H(T_1,T_2)} + \mathrm{Dist_H(T_2,T_3)} \quad (18)$$

Proof: this is proved *ad absurdio* : if $Dist_H(T_1,T_2) + Dist_H(T_2,T_3)$ was smaller than $Dist_H(T_1,T_3)$, this would mean that there exists a path $(T_1,...,T_2,...,T_3)$ shorter than the shortest path from T_1 to T_3, which is absurd. Therefore $Dist_H$ is a distance.
A simple algorithm is used for the calculation, the idea is to go up the hierarchy in parallel and find the ancestors with their distance and then find the *LCST* through which the path is the shortest. The algorithm takes into account multiple inheritance. If the hierarchy is not a connected graph, then it may happen that for two given concept types, there exists no path between them (it means that the hierarchies stop at different roots without having found any common ancestor). In that case, the distance returns the upper bound of the length equal to $(MaxDepthConceptHierarchy \times 2+1)$ for the concepts or $(MaxDepthRelationHierarchy \times 2+1)$ for the relations.

3.3.2.5 Distance between a concept type and a literal

The distance between a primitive type and a literal is a constant greater than any type distance. Let

$$Dist_{LC}(T_1,L_X) = (Max_C \times 2 + 1) \quad (19)$$

To consider a literal as a basic type at the top of the hierarchy, (19) has to be replaced by (20):

$$Dist_{LC}(T_1,Lit_X) = Depth(T_1) + 1 \quad (20)$$

where $Depth(T_1)$ is the length of the path from the root of the hierarchy to the primitive type T_1.

3.3.2.6 Pseudo-distance annotation triple - property family

Let $Triple_A = (T_{RA}, T_{A1}, T_{A2})$ a triple from an annotation and let $Triple_B = (T_{RB}, T_{B1}, T_{B2})$ a triple from the *ABIS*. In an RDF triple, T_{A1} and T_{B1} are primitive concept types, let

$$D_{C1} = W_C \times Dist_H (T_{A1}, T_{B1}) \quad (21)$$

Now, considering the types T_{A2} and T_{B2} :

- If both are primitive concept types then let :

$$D_{C2} = W_C \times Dist_H (T_{A2}, T_{B2}) \quad (22)$$

- If one is a primitive concept type T and the other is a literal L

$$D_{C2} = W_C \times Dist_{LC}(T,L) \quad (23)$$

- If both types are literals then from the *ABIS* we know $[B_{low}, B_{up}]$ and from the annotation we know the literal Lit_X. Let:

$$D_{C2} = W_L \times N \times Dist_I(Lit_X, [B_{low}, B_{up}]) \quad (24)$$

Finally we calculate the distance between the relation types, let

$$D_R = W_R \times Dist_H(T_{RA}, T_{RB}) \quad (25)$$

The final distance between the annotation triple and a property family of the *ABIS* is given by:

$$Dist_{TFABIS}(Triple_A, Triple_B) = D_{C1} + D_R + D_{C2} \quad (26)$$

If D_{C1} and D_{C2} were real distances then $Dist_{TFABIS}$ would be a Manhattan distance.

3.3.2.7 Pseudo-distance annotation triple - ABIS

The pseudo-distance between a triple and an *ABIS* is the minimal pseudo-distance between this triple and the *ABIS* triples.

$$Dist_{TABIS}(Triple,ABIS) = \min_{Triple_i \in ABIS}(Dist_{TFABIS}(Triple,Triple_i)) \quad (27)$$

3.3.2.8 Pseudo-distance annotation triple - CAP

The calculation of the pseudo-distance $Dist_{TCAP}(Triple,CAP)$ is the same as for the *ABIS* except for the primitive type distance: when comparing two triples, if the type of the annotation is a specialization of the type of the triple from the *CAP*, the length of the path between them is set to *0*. This is to take into account the fact that the *CAP* captures preferences and that anything more precise (as a specialization) is included in the preferences.

3.3.2.9 Pseudo-distance between annotation and an ABIS, a CAP and an archive.

The distance to an ABIS is the sum of the distances for the triples of the annotation:

$$Dist_{AABIS}(An_X, ABIS) = \sum_{Triple_j \in An_X} Dist_{TABIS}(Triple_j, ABIS) \quad (28)$$

where An_X is an annotation and *ABIS* is the *ABIS* of an *AA*.
The distance to a CAP is the sum of the distances for the triples of the annotation :

$$Dist_{ACAP}(An_X, CAP) = \sum_{Triple_j \in An_X} Dist_{TCAP}(Triple_j, CAP) \quad (29)$$

where An_X is an annotation and *CAP* is the *CAP* of an *AA*.
Finally, the distance to an archive is the sum of the distances to its *ABIS* and *CAP*:

$$Dist(An_X, AA_y) = Dist_{AABIS}(An_X, ABIS_y) + Dist_{ACAP}(An_X, CAP_y) \quad (30)$$

where AA_y is an archivist agent, An_X is an annotation and $ABIS_y$ and CAP_y are the *ABIS* and *CAP* of AA_y. Based on this final pseudo-distance, the *AM* can compare the bids given back by the *AAs* and allocate a newly submitted annotation to the closest agent, following a contract-net protocol.

3.3.3 Conclusion and discussion on the Annotation allocation

The implementation was done and tested in JADE; the debugger tools of JADE provide a message sequences diagram and Figure 4 shows and comments the snapshot of a trial sequence for the allocation of an annotation. It involves, one *IC* (*localIC*), one *UPM* (*localUPM*), one *AM* (*AM*), three *AA*s, one of which is part of a UPA role (*profileAA*) and two archiving document annotations (*localAA1* and *localAA2*).

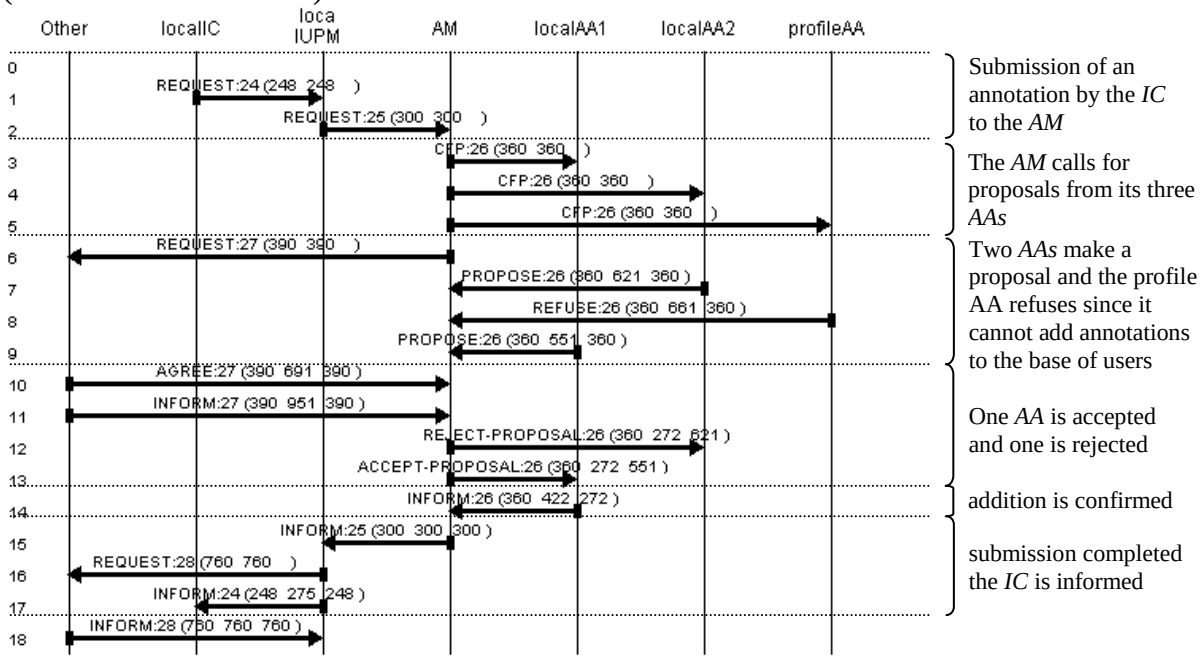

Figure 4 Sequence diagram of an allocation

The first tests on the prototype implemented showed an effective specialization of the content of the annotation archives with the problem that the choice of the specialization of the archives content must be very well studied to avoid unwanted unbalanced archives. This study could be done together with the knowledge engineering analysis carried out for the ontology building. However automatic assistance could also be envisaged in particular to take into account additional criteria and heuristics:

- If two archives are close with regard to their content, then the system could try to *balance their storage volume and workloads*. One criterion would be to consider all the archives within a margin from the best bid (e.g. the first 10%) as good candidates and choose the one with the smallest storage and workload.
- *Monitoring tools* pointing to archives very close with regard to their content (a situation which may be created on purpose to augment archiving space) or to archives with storage volume and workload much higher than others, *etc.* could be interesting for managing the annotation bases.

Finally the measure can use *different constructors to combine sub-measures*. In the literature we can find *Min(...), Max(...), Average(...), Sum(...), etc.* It would be interesting to test these different types of distances and study their repercussions on the allocation of annotations.

In distributed databases, overnight automatic reorganization of distributed bases have been studied [28]; this solution could be extended to distributed annotation archives, however it must be noticed that (a) one of our motivation for distributed knowledge based system was to leave knowledge where it was (wrappers, locally managed data, property, *etc.*), (b) complexity of automatic reorganizing is known to be prohibitive.

A major drawback of the current distance is the part which is not semantic *i.e.* the lexicographical distance between literal values. In fact, this part of the measure encounters the same problem as in information retrieval, so some ideas of this field could be reused:

- *TF*IDF approach*: to remove stop words and build vectors of the literal values on one side and of each notion of the ontology on the other side, using the labels (term and synonyms) and the natural

language definition. Then a cosine measure or equivalent could be used to choose the closest notion(s) and use it/them in evaluating the distance.

- *Label mining*: to mine the literal value to extract labels of concepts in literal; this could require to couple O'CoMMA with a general thesaurus such as Wordnet[2] to get both domain specific notions and general domain knowledge to avoid lack of common knowledge. Then a graph distance could be used again.
- *Natural language processing*: to produce conceptual structures from the literal and consider the whole annotation once each literal has been replaced.

Finally, the least-common super-type distance also calls for some discussions. In particular, the current distance considers the subsumption link as a universally unitary link *i.e.* a subsumption link always represent a path with a length of 1. Is this intuitively natural? Consider the taxonomy in Figure 5 case (1): the unique length of 1 for each subsumption shows how the current distance consider the taxonomic tree when measuring it.

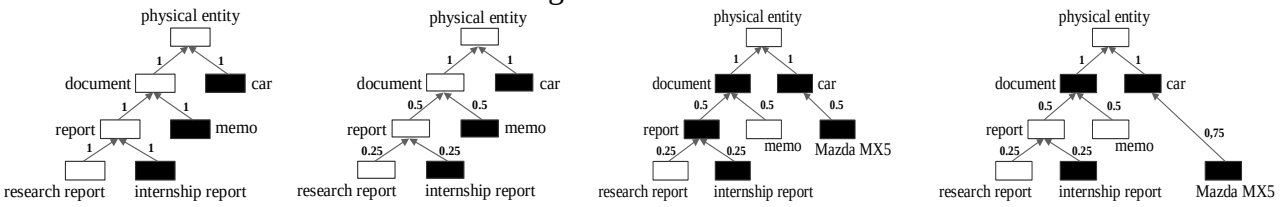

Figure 5 Length of a subsumption link

The first question is: "is it intuitively normal that the length of the link between a *car* and a *physical entity* be the same as the length of the link between a *report* and a *research report* ?" In my opinion the answer is already no, but the second question makes it even more obvious: "is it intuitively normal that the calculated distance of the path from *memo* to *internship report* be 3 *i.e.* the same distance of the path from *memo* to *car*"? Intuitively no. So the distance as it was defined, shows here some limitations in capturing the semantic proximity and it comes from the fact that the length of the subsumption link *i.e.* the intuitive semantic proximity between the subsumer and the subsumee is not homogeneous in the taxonomy of the ontology. A first idea is that since the upper layer is more abstract, the subsumption distance could be attenuated with the depth of the notion as shown in Figure 5 case (2). The length of the subsumption link would thus be given by:

$$Length(Subsumption(supertype,subtype)) = \left[\frac{1}{2}\right]^{depth(supertype)} \quad (31)$$

with *depth(type)* a recursive series defined by :

$$\begin{cases} depth(\top)=0 \quad \text{with } \top \text{ the root of a taxonomy} \\ depth(type)=1+\underset{s_t;Subsumption(s_t,type)}{Min} depth(s_t) \end{cases} \quad (32)$$

Using this subsumption length, the path from *memo* to *internship report* has a distance of *1.25* while the path from *memo* to *car* has a distance of *2.5*. This is much better, but we replaced the homogeneity hypothesis by a new one which is "the length of the subsumption links decreases with the taxonomic depth". This means that the taxonomy must be homogeneous in its choices of differentia at each and every level. To illustrate that point, let us consider Figure 5 case (3): the distance between *report* and *document* is 0.5 *i.e.* the same than between *car* and *Mazda MX5* while the distance between *report* and *internship report* is only 0.25. It is clear that the difference between *car* and *Mazda MX5* is more precise than between *report* and *document*, and so the concepts should be more distant. Thus what we would like to express is rather what is shown in Figure 5 case (4), where the important difference between *car* and *Mazda MX5* is captured in a longer link having a length of 0.75 just like the path from document to internship report. One can even envisage further improvements such as removing the hypothesis that the path between two brother types is given by the distance of the path through the father, but these additional abilities to tune the distance require some work to "evaluate the intuitive distance" which is clearly difficult, time-consuming, and most probably extremely subjective. Thus an option could be to initialize the taxonomy with the previous top down algorithm based on depth and then improve these distances using users' feedback.

[2] http://www.cogsci.princeton.edu/~wn/

3.4 Distributed query-solving

This second section describes how agents allocate the tasks involved in solving a query. It explains how agents use the nested query-ref protocol together with a description of the overlap between the query needs and the statistics over an archive of annotations for distributing sub-tasks in the form of sub-queries. It will present, first the protocol and the algorithm used for the CoMMA prototype and then a second algorithm partially implemented to validate improvements and support future work.

3.4.1 Allocating tasks

When a user expresses a query, the resolution of this query may involve several annotation bases distributed over several *AAs*. Results may thus be a merging of partial results found by the concerned *AAs*. To determine if and when an *AA* should participate to the solving of a query, *AAs* calculate the overlap between their *ABIS* and the properties at play in the query. The result is an *OBSIQ* (Overlap Between Statistics and Instances in a Query): a light structure which is void if the *AA* has no reason to participate to the query solving or which otherwise gives the families of properties for which the *AA* should be consulted. Table 3 gives a short example of an *OBSIQ* where we can see three families relevant for the resolution process.

Query Properties Overlapping with Statistics	Min	Max
Article → Title → Literal	"Multiple acts ..."	"The incredible..."
Article → Concern → Computer Science		
Person → Name → Literal	"Albertini"	"Hanstucken"
...		

Table 3 OBSIQ (Overlap Between Statistics and Instances in a Query)

The *OBSIQ* is marshalled using RDF(S)/ XML tags defined in the CORESE namespace and sent by the *AA* to the requesting *AM* in charge of coordinating the resolution process. Using the *OBSIQ* it requested before starting the solving process, the *AM* is able to identify at each step of the decomposition algorithm and for each sub-query it generates, which *AAs* are to be consulted.

3.4.2 Query solving protocol

Agents involved in the allocation of the resolution of a query are: the *IC*, through which the query is built and submitted ; the *UPM*, that observes and forward the query ; the *AM*, that manages the division of tasks, the breaking down and distribution of the query and the merging of partial results to compose the global answer ; the *AA*, that manages local annotation archives and tries to locally solve sub-queries issued by the *AM*, using the annotation base it is in charge of. The *IC* and *UPM* are essentially involved in the emission of the query and will not be considered any further here. The *AA* also plays a part of the *User Profile Archivist* role (using the user profiles that may be involved in a query such as documents→written_by→person→name→"gandon") and a part of the *Corporate Model Archivist* role (using the corporate model that may be involved in a query such as documents→written_by→group→activity→accountancy).

The communication protocol used for the query solving is a composition of the FIPA query-ref protocol to allow multiple stages with sub-queries being exchanged between the *AM* and the *AA*. The use of the *OBSIQ* structure enables the *AM* to engage in a dialogue only with agents potentially knowledgeable for the current part it is trying to solve.

The protocol is depicted in Figure 6 and corresponds to the following textual description:

1. Through the *IC*, the user builds a well-formed and valid query using the ontology O'CoMMA and it submits it.
2. The *IC* sends the request for solving the query to the *UPM* it is acquainted with.
3. The *UPM* contacts an *AM* called the *LocalAM*. It is the first one found by the *DF*, but it could be chosen on more complex criteria such as the workload.
4. The *LocalAM* calls upon the *DF* to look for other *AMs*.
5. The *LocalAM* sends to other *AMs* a query requesting *OBSIQs* for the given query.
6. The *LocalAM* and other *AMs* send a query requesting *OBSIQs* for the given query to their *AAs*.
7. The *AAs* calculate and send their *OBSIQs*.
8. The other *AMs* send to the *LocalAM* the list of non void *OBSIQs* and corresponding *AAs* (the other Annotation Mediators are matchmakers here).
9. The *LocalAM* interacts with the *AAs* to solve the query through a sequence of query-ref containing the sub-queries it generates.
10. The *LocalAM* sends back the global answer to the *UPM*.
11. The *UPM* answers to the *IC*.

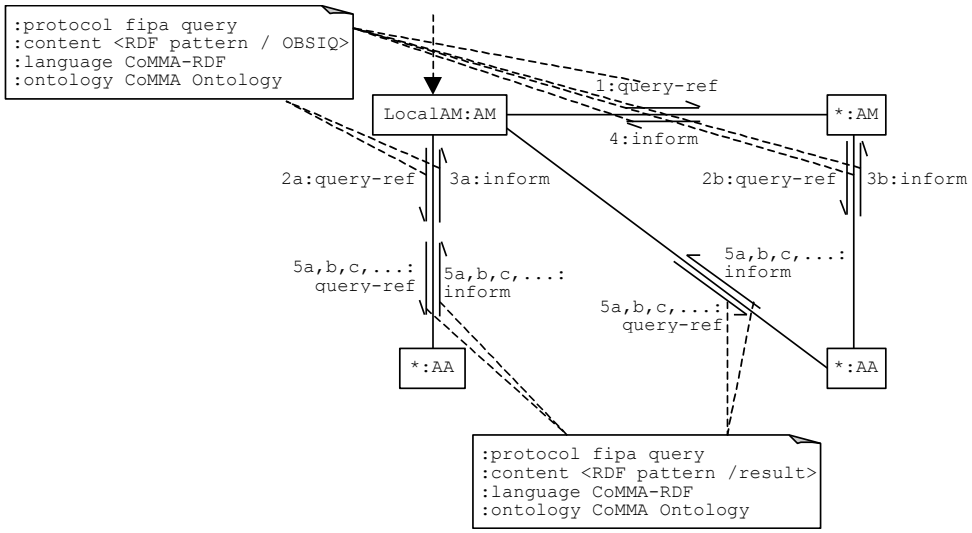

Figure 6 Acquaintance and protocol diagrams in query solving

The whole process relies on the ability to decompose the query and merge the partial results. This process is detailed in the following sections.

3.4.3 Distributed query solving

The decomposition algorithm consists of four stages: the pre-processing for query simplification, the constraint solving, the question answering and the final filtering. These stages, detailed in the following subsections, manipulate the query structure through the Document Object Model (DOM), an interface to manipulate an XML document as a tree or, more precisely, as a forest. In our case, the tree structure represents an RDF pattern and contains nodes representing resources or properties, except for the leaves that may be resources or literals (see top part of Figure 8). The resource nodes may have an URI and the *AMs* use them as cut points during the query solving to build smaller sub-queries that can be sent to the *AAs* to gather information scattered in several archives; URI are also joint points to merge partial results.

3.4.3.1 Query simplification

A pre-processing is done on the query before starting the decomposition algorithm. A query may hold co-references. Simple co-references *i.e.* two occurrences of a variable where one reference is a node of a tree of the query and the other one is the root of another tree, are merged by grafting the second tree on the first one - see Figure 7. Complex co-references would generate distributed constraint problems. They are erased and replaced by simple variables for the duration of the distributed solving; they are then reintroduced at the last stage for the final filtering.

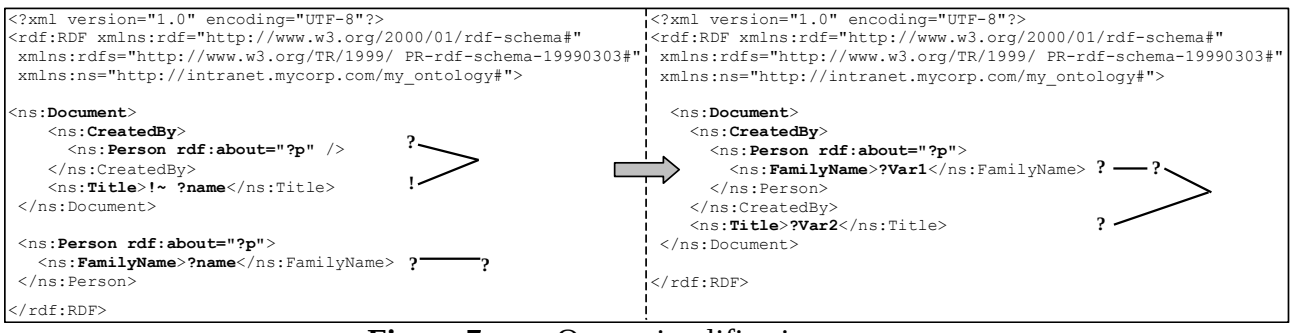

Figure 7 Query simplification process

3.4.3.2 Constraint solving

To cut down the network load, the decomposition starts with the solving of constraints, represented here by exclamation marks in the top part and the part 1 of Figure 8. The grouping of constraints limits the number of messages being exchanged by constraining the queries as soon as possible.

The *AM* groups constraints according to the concept instance used for their domain value. It chooses a group of constraints among the deepest ones and creates a sub-query by extracting this sub-tree and asking for the possible URIs of its root concept. Its purpose is to replace this sub-constraint in

the global query by the list of possible URIs for its root, and iteratively reduce the depth of the global query.

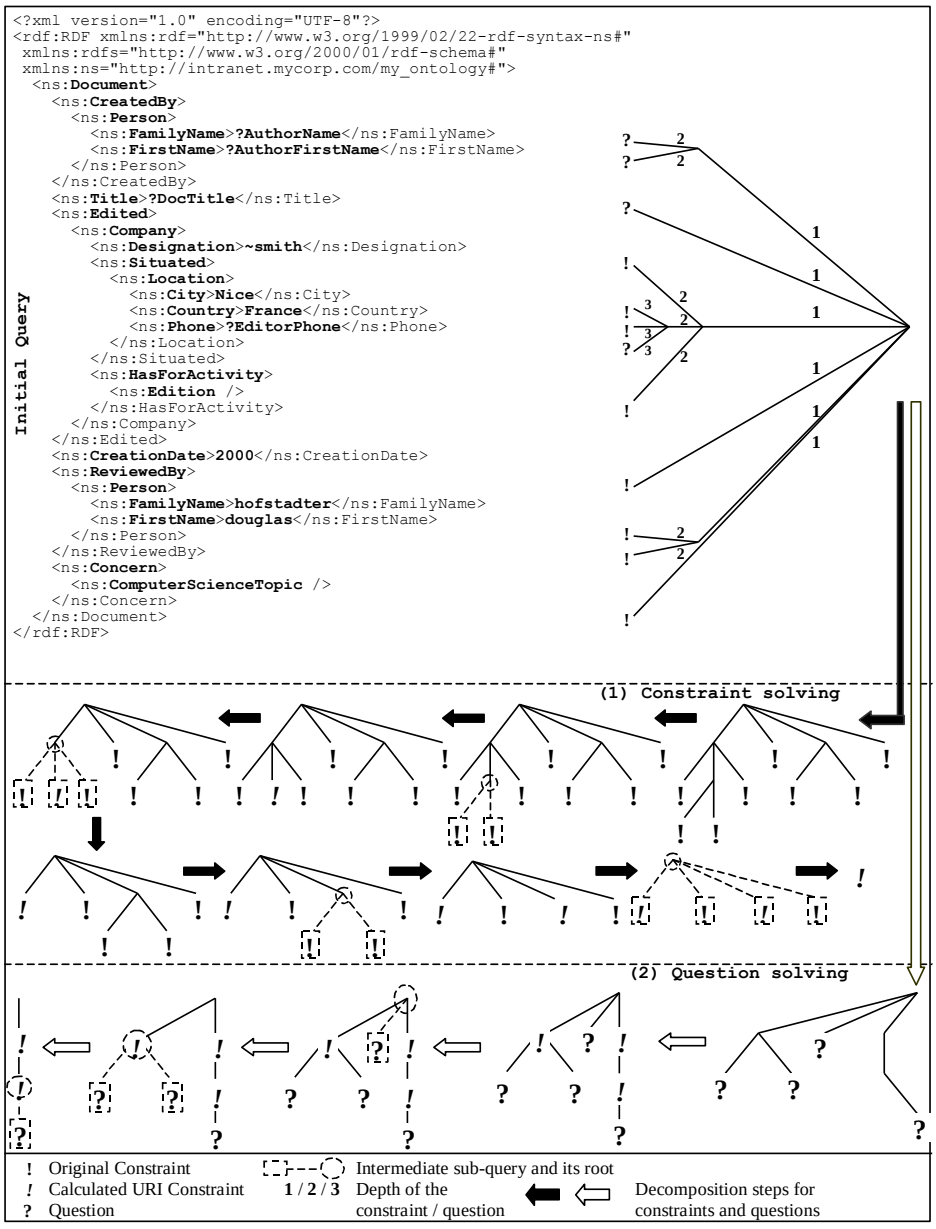

Figure 8 Constraint and question solving

Among the candidate *AAs*, the *AM* identifies the *AAs* concerned by a sub-query thanks to the *OBSIQ* they provided at the start, and it contacts them to try to solve the sub-query using their local resources:

- If a piece of RDF matches the query and provides the URI of its root concept, the *AA* sends it.
- If an annotation violates a constraint, it is dismissed.
- If an annotation answers partially, and if the root concept of the result has a URI, the *AA* returns the incomplete answer since the missing part can be found somewhere else thanks to URI.
- If an annotation answers partially, but does not have a URI at the root concept (existential quantification), then the *AA* does not return it since it cannot be completed elsewhere.
- If an annotation answers the query but does not have a URI for the root concept, the *AA* returns the whole annotation.

Partial results are merged in the local base of the requesting *AM*. The *AM* then reduces the original query using the URIs it has learnt and generates a list of smaller queries. For each one of these queries, it applies this algorithm again until no constraint is left.

3.4.3.3 Question answering

After the constraint solving, the *AM* has, in its base, complete annotations and partial answers with URIs. Using these unique identifiers, it is able to request the information asked by the user for each

one of the resources identified. Therefore, after solving the constraints, the *AM* emits queries to fill the question fields represented by question marks in the top part and the part 2 of Figure 8. The *AM* starts from the root of the original query since its potential URIs should have been found by now. It generates a sub-query each time it finds a question during its walk through the tree. Some URIs may still be missing for unconstrained nodes and intermediate queries are issued to solve them.
If the initial query was not constrained at all, there is no URI to constrain the root when starting the question solving. Thus the flow of data to solve it would potentially be too high and could result in a network jam. In that case, the *AM* switches to a degraded mode and simply asks the *AAs* to solve the whole query locally, potentially loosing some answers.

3.4.3.4 Filtering final results

Once the questions have been answered, the *AM* projects the original query on the base it has built and extracts the final correct full answers. This stage enables it to finalize the merging of partial results and to take into account the possible cross-references occurring in the constraints, that were discarded during the pre-processing. The *AM* then sends back the result to the requester agent.

3.4.3.5 Overall algorithm and implementation details

This section gives some details of the implementation of this first algorithm. The query decomposition was implemented inside CORESE using two special techniques for walking through the XML DOM tree and provide an RDF DOM: nested recursive programming based on polymorphism was used for propagation of tests and modifications; factory-oriented programming was used to generate an up-to-date RDF typing (concept, properties) of the XML nodes of the DOM as the *AM* walks through the tree and modifies it. Factories are called each time an algorithm moves in the tree and provide instances with the most precise type.

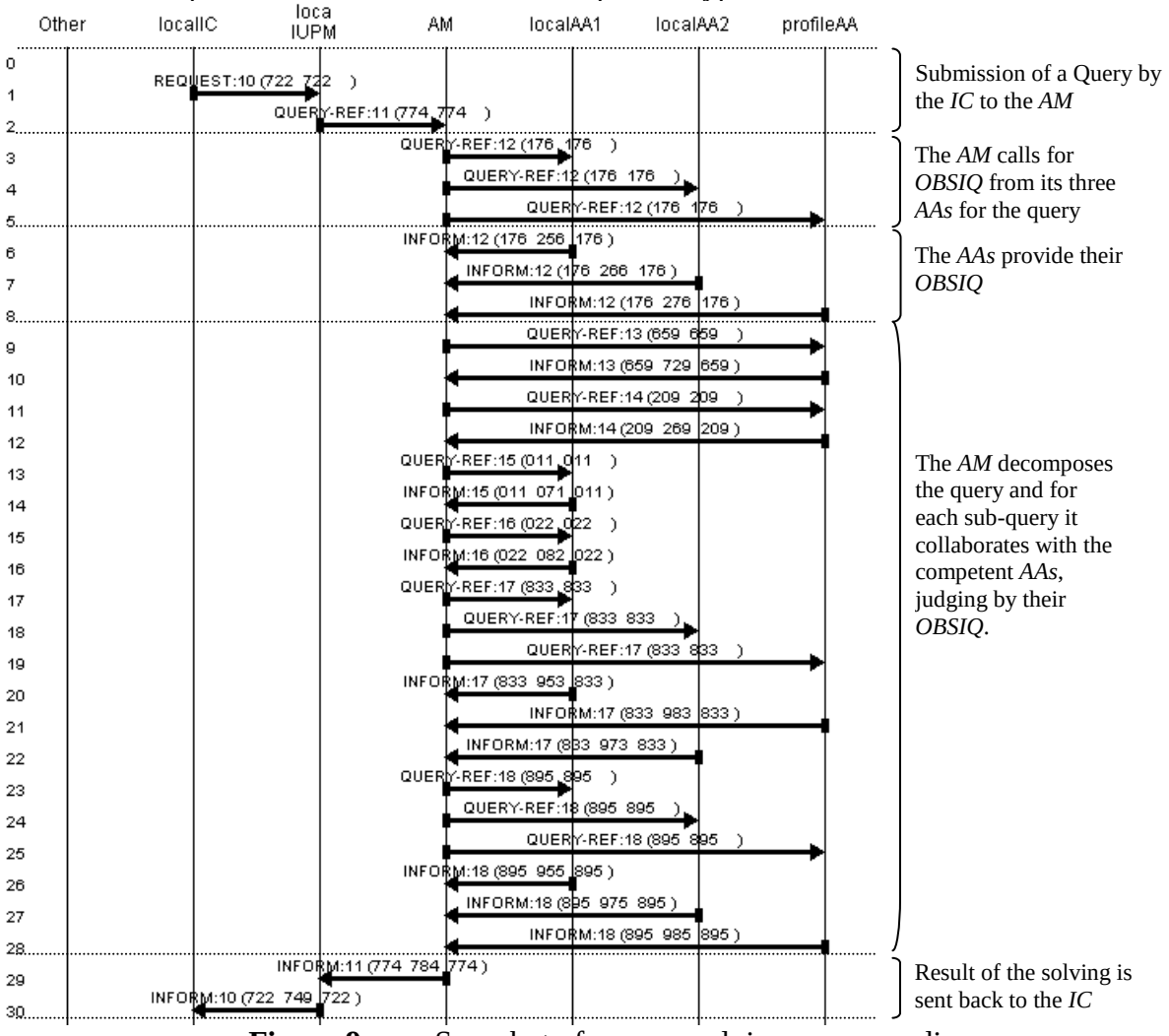

Figure 9 Snapshot of a query solving sequence diagram

However, one of the problems caused by the complex recursive DOM walking and query-solving algorithm is that it could not easily be rewritten in JADE using the nested behaviors since this

would have required to 'derecursify' the algorithm. It is a case of agentisation. So instead of designing a query-solving behavior for the *AM*, the behavior that was designed, is able to dialog with the query-solving algorithm. It uses synchronized object that reifies the dialogue, and the *AM* and its solver run in two different threads dialoguing through a shared synchronized object. It is a case of wrapping a complex algorithm, although this wrapping is close to a transducer approach.
The implementation was done and tested in JADE; the debugger tools of JADE provide a message sequences diagram and Figure 9 shows and comments the snapshot of a trial sequence for the solving of a query. It involves, one *IC* (*localIC*), one *UPM* (*localUPM*), one *AM* (*AM*), three *AA*s, one of which is part of a *UPA* role (*profileAA*) and two archiving document annotations (*localAA1* and *localAA2*).

3.4.4 Conclusion and discussion on the query solving algorithm

3.4.4.1 Discussion on the first algorithm

The first algorithm made some simplifying choices and left some open issues. The first problem that could happen is a query with no constraint as in Figure 10.

```
1   <document>
2     <title>?</title>
3     <author>
4       <person>
5         <name>?</name>
6         <first_name>?</first_name>
7       </person>
8   </document>
```

Figure 10 Unconstrained query

If such a query was decomposed into unconstrained sub-queries, the network will be overloaded since they may generate huge answers (e.g. all the persons with their names). The problem is that there is no simple way to avoid this effect since the query is by itself too large. Other projects have used 'watchdogs' agents to allow such queries while monitoring high traffic queries and prevent them from jamming the network.
A second problem is the one of a query containing only isolated concepts, the algorithm does not work properly; yet these queries are perfectly acceptable, the Figure 11 shows a query requiring annotations where the concepts documents and knowledge engineering are used.

```
1   <document/>
2   <KnowledgeEngineering/>
```

Figure 11 Vertex-based query

Such a query should be recognized and solved in degraded mode. It was not done in CoMMA since the query interface of the Interface Controller does not allow the user to create such a query.
Finally, in the case of a query with cross constraints as in Figure 12, decomposition of constraints was not distributed; but is removed before the distributed solving and reintroduced at the end. However the simplification of the query makes it much less constrained and augments the number of messages exchanged for the resolution. Improvements could be found in considering the work done, for instance, on the semi-joint operator used in Distributed Data Bases [28].

```
1   <document>
2     <title>~?x</title>
3     <author>
4       <person>
5         <name>?x</name>
6         <first_name>fabien</first_name>
7       </person>
8     <date>1999</date>
9   </document>
```

Figure 12 Query with cross references

To start improving the algorithm, let us considered one of the sources of multiplication of messages: the anonymous existential quantificator. The first algorithm is that really treats the absence of URI as a universal statement of the existence of an instance. For example in Figure 13, there are two existential quantifications: one says that "*there exists a report* that has for title Effectiveness of..." and another one says that " *there exists a person* called John Smith who wrote ...".

```
1   <report>
2     <title>Effectiveness of Factories in AOP</title>
3     <author>
4       <person>
5         <name>Smith</name>
6         <first_name>John</first_name>
7       </person>
8     </author>
9   </report>
```

Figure 13 Existential quantification in an annotation

However these statements have a context of annotation and even more, they are issued by an *AA* uniquely identified by its address. The idea to improve this, was to recreate URI for such statement, denoting the source of the existential quantification; these URI being based on the address of the agent and the internal generic IDs of CORESE. The annotation in Figure 13 is thus virtually considered as the annotation in Figure 14:

```
<report ID="acacia_archivist@fapollo:1099/JADE#genID54">
  <title>Effectiveness of Factories in AOP</title>
  <author>
    <person ID="acacia_archivist@fapollo:1099/JADE#genID79">
      <name>Smith</name>
      <first_name>John</first_name>
    </person>
  </author>
</report>
```

Figure 14 URIs for existential statements

Using this approach, the *AM* now knows which agent issued a statement and thus if further sub-queries use this ID, they will be sent only to the relevant agent.

The second point addressed by the improved algorithm is the choice of the order in solving the constraint. The previous algorithm chose the first deepest constrained concept to start with; this is because most constraints are given in the leaves by literal values or imposed by regular expressions. The new algorithm tries to determine the most constrained concept in the query and starts with this concept in order to cut down the number of results as soon as possible. Moreover, instead of generating multiple sub-queries to cover all the possible URIs for a concept, the solver now generates one sub-query with a disjunctive list of URIs thus delegating some of the combinatory work. Together these three improvements now allow the *AM* to reduce the number of sub-queries, the size of results and to detect sooner if the conjunction of the constraints in a query cannot be satisfied: this corresponds to a case where no URIs at all (original or generated) could be found for a constrained concept.

This algorithm also supports future improvements such as the use of heuristics to sort the types of constraints and to start with the strongest ones: *"known URI" > "imposed literal values" > "literal value constrained by regular expression" > etc.* Additionally, the system could use the statistics on the base about the frequency of relations and choose the less frequent to cut down number of possibilities as soon as possible.

The new algorithm has been partially implemented and tested (a centralized version). It works by propagation: as soon as a concept is solved (*i.e.* its potential URIs are known), its neighbors become constrained by their relation with a concept having known URIs, and so on. If no constrained concept is found, a pre-processing searches for loosely constrained concepts to initiate the process; this is where the heuristics could be introduced to sort constraints.

Another problem was raised because the multi-instantiation in RDF allows one object to be described from several points of view. When a query requires knowledge from different facets of instantiation, this may cause a loss of information in the current distributed algorithm. Consider the two annotations in Figure 15 and Figure 16 that represent two different facets of myself.

```
<CoMMA:Researcher rdf:ID = "http://www-sop.inria.fr/acacia/personnel/Fabien.Gandon/">
 <CoMMA:FamilyName>Gandon</CoMMA:FamilyName>
 <CoMMA:FirstName>Fabien</CoMMA:FirstName>
 <CoMMA:IsInterestedBy> <CoMMA:ComputerScienceTopic/> </CoMMA:IsInterestedBy>
</CoMMA:Researcher>
```

Figure 15 Researcher facet of Fabien Gandon

```
<CoMMA:Lecturer rdf:ID = "http://www-sop.inria.fr/acacia/personnel/Fabien.Gandon/">
 <CoMMA:IsInterestedBy> <CoMMA:MusicTopic/> </CoMMA:IsInterestedBy>
</CoMMA:Lecturer>
```

Figure 16 Lecturer facet of Fabien Gandon

Annotation in Figure 15 expresses that I am a Researcher interested in computer science, while annotation in Figure 16 expresses I am a lecturer interested in music. If you consider that these two views should not be merged and that for instance I should not be considered as a Lecturer interested in computer science, then everything is fine. Conversely, if you believe properties are attached to an instance and not to a facet then there is a problem: consider the query in Figure 17, it looks for lecturers interested in computer science, with their name and first name.

```
<CoMMA:Lecturer>
 <CoMMA:FamilyName>?name</CoMMA:FamilyName>
 <CoMMA:FirstName>?firstname</CoMMA:FirstName>
```

```
4     <CoMMA:IsInterestedBy> <CoMMA:ComputerScienceTopic/> </CoMMA:IsInterestedBy>
5    </CoMMA:Lecturer>
```

Figure 17 Query on merged facets

If annotations in Figure 15 and Figure 16 are gathered in one annotation, CORESE will have no problem solving the merge between the two instances. If they are in different annotations or even on different sites, then the distributed algorithm will fail because the following constraint is false on both annotations taken separately : **Lecturer — Is Interested By — Computer Science Topic**. Indeed, the projection on the first annotation will fail because *Lecturer* does not match *Researcher* and the projection on the second annotation will fail too, because *Computer Science Topic* does not match *Music Topic*. A solution to remedy to that problem is to decompose the constraint into two constraints solved one after the other: (a) the *relation using its generic signature* and the available URIs if they have been solved by previous decomposition steps and (b) the *typing constraint* using the previously retrieved URIs.

Finally as the *ABIS* describes the range of literal values, it could be interesting to find a mechanism to describe the set of genuine URIs (not the one generated for existential quantification). The full list of URIs may be too long but compression techniques could be used such as regrouping the URI with a common root together. This could also be used to merge multi-instantiation annotations. The description could be used in annotation distribution as an additional clustering feature.

4 Conclusion

The stake of the work presented here was to make a proof of concept of the interest of merging ontology and distributed artificial intelligence to provide distributed mechanisms managing distributed knowledge. Knowledge distribution is both the spatial property of knowledge of being scattered about and the act of diffusing knowledge. The work presented here focused on the tasks of managing both facets of distribution to allocate storage and distribute exploitation and, since they are linked to one another in their performances, it chooses two complementary strategies to implement them:

- the *allocation of an annotation* is based on a contract-net where bids are the result of a distance between the semantic contribution of the annotation and the description of the semantic contribution of an archive to the memory content. This protocol tends to maintain the specialization of the base.
- the *solving of a query* exploits the description of the semantic contribution of an archive to the memory to allocate sub-tasks to the relevant agents in charge of the archives. This protocol tries to limit the number of messages exchanged during the distributed query solving process while enabling the answers to be found even if the needed information is split over several bases.

In the first case, the ontology is used as a shared space enabling to define a shared distance function, the results of which can be compared in a distributed protocol. In the second case, the ontology provides primitives to describe the knowledge possessed by an agent and thus to rule on the pertinence of its involvement in a cooperation. Thus it is clear that here, the ontology is a corner stone of the intelligent distributed mechanisms to manage distributed knowledge.

The prototype implemented in JAVA was evaluated by end-users from a telecom company (T-Nova System Deutsch Telekom) and a construction research center (CSTB) with archives containing up to 1000 annotations. Interface and ergonomics problems were raised by the users but usefulness and the potential of the functionalities offered by the system were unanimously acknowledged. In particular, the ontology-oriented and agent-oriented approach were appreciated by the end-users for their powerfulness and by the developers for the loosely-coupled approach they support to specify, to implement and to deploy a distributed system integrating results from several research fields. Indeed, for the integration phase the agent technology proved to be extremely valuable: the different agents have been developed by distant partners having the needed experience and starting from shallow agents ; but since the agents are loosely-coupled software components and that their roles and interactions have been specified using a consensual ontology, the integration and setup of a first prototype was achieved in less than two days.

All the aspects investigated here showed that the ontological consensus lays a semantic foundation on which further consensus may be built ; to us, it is clear that in the distributed artificial intelligence paradigm, the ontology can provide the keystone for designing social cooperation mechanisms.

Acknowledgements

I warmly thank my colleagues of the ACACIA team and the CoMMA consortium for the fruitful discussions we had, and the European Commission that funded CoMMA (IST-1999-12217).

References

[1] J.L. Aguirre, R. Brena, F. Cantu-Ortiz, (2000). Multiagent-based Knowledge Networks. To appear in the special issue on Knowledge Management of the journal Expert Systems with Applications.

[2] Y. Arens, C.A. Knoblock, W. Shen. Query reformulation for dynamic information integration. Journal of Intelligent Information Systems, 6(2):99-130, 1996.

[3] F. Bellifemine, A. Poggi, G. Rimassa, Developing multi- agent systems with a FIPA-compliant agent framework. Software Practice & Experience, 2001, 31:103-128

[4] F. Bergenti, A. Poggi, Exploiting UML in the Design of Multi-Agent Systems. Engineering Societies in the Agents World - LNAI, Springer, 2000, vol. 1972, 106-113.

[5] T. Berners-Lee, J. Hendler, O. Lassila, The Semantic Web, Scientific American, May 2001, 35-43.

[6] B. Berney, and E. Ferneley, (1999), "CASMIR: Information Retrieval Based on Collaborative User Profiling", In Proceedings of PAAM'99, pp. 41-56. www.casmir.net

[7] C. Bothorel, and H. Thomas, (1999), "A Distributed Agent Based-Platform for Internet User Communities", In Proceedings of PAAM'99, Lancashire, pp. 23-40.

[8] D. Brickley, RV Guha, Resource Description Framework (RDF) Schema Specification. W3C Candidate Recommendation, 27 March 2000 (http://www.w3.org/TR/2000/CR-rdf-schema-20000327/)

[9] C. Collet, M. N. Huhns, and W. Shen. Resource integration using a large knowledge base in CARNOT. IEEE Computer, pages 55-62, December 1991

[10] O. Corby, R. Dieng, C. Hebert, A Conceptual Graph Model for W3C Resource Description Framework. In Proc. International Conference on Conceptual Structures, 2000, Springer LNAI 1927, 468-482

[11] K. Decker, K.P. Sycara., Intelligent adaptive information agents. Journal of Intelligent Information Systems, 9(3):239-260, 1997.

[12] R. Dieng, O. Corby, A. Giboin, M. Ribière. Methods and Tools for Corporate Knowledge Management. In Decker and Maurer, International Journal of Human-Computer Studies, special issue on knowledge Management, vol. 51, 567-598, Academic Press, 1999.

[13]P. Doyle, B. Hayes-Roth, Agents in Annotated Worlds, Proc. Autonomous Agents, ACM, 1998, 173-180

[14] M. Dzbor, J. Paralic and M. Paralic, Knowledge Management in a Distributed Organisa-tion, In Proc. of the BASYS'2000 - 4th IEEE/IFIP International Conference on Information Technology for Balanced Automation Systems in Manufacturing, Kluwer Academic Pub-lishers, London, September 2000, ISBN 0-7923-7958-6, pp. 339-348

[15] J. Ferber, O. Gutknecht, A meta-model for the analysis and design of organizations in multi-agent systems. IEEE Computer Society, Proc. 3rd ICMAS, 128-135, 1998.

[16] Foundation for Intelligent Physical Agents, FIPA Specifications, 2001, (http://www.fipa.org/)

[17] F. Gandon, Engineering an Ontology for a Multi-agent Corporate Memory System, Proc. International Symposium on the Management of Industrial and Corporate Knowledge 2001, 209-228.

[18] F. Gandon, R. Dieng, Corby, Giboin, A Multi-agent System to Support Exploiting an XML-based Corporate Memory, Proc. Practical Application of Knowledge Management 2000, 10.1-10.12.

[19] F. Gandon, A. Poggi, G. Rimassa, P. Turci, Multi-Agent Corporate Memory Management System, Journal of Applied Artificial Intelligence, Taylor & Francis, Issue 9-10/October - December 2002, p. 699-720

[20] M. Genesereth, A. Keller, O. Duschka, Infomaster: An Information Integration System, in proceedings of 1997 ACM SIGMOD Conference, May 1997.

[21] A. Kiss, J. Quinqueton, Multiagent Cooperative Learning of User Preferences, Proc. European Conference on Machine Learning Principles and Practice of Knowledge Discovery in Databases, 2001.

[22] M. Klush, Intelligent Information Agent: Agent-Based Information Discovery and Man-agement on the Internet, Springer, pages IX-XVII, 1999

[23] O. Lassila, R. Swick, Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, 22 February 1999, (http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/)

[24] A.Y. Levy, D. Srivastava, and T. Kirk, Data model and query evaluation in global infor-mation systems. Journal of Intelligent Information Systems 5(2):121-143, September 1995

[25] E. Mena, V. Kashyap, A. Sheth, A. Illarramendi, "OBSERVER: An approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies," Proceedings of the 1st IFCIS International Conference on Coopera-tive Information Systems (CoopIS '96), Brussels, Belgium, June 1996

[26] M. Nodine, J. Fowler, T. Ksiezyk, B. Perry, M. Taylor, A. Unruh, Active Information Gathering In Infosleuth™; In Proc. Internat. Symposium on Cooperative Database Systems for Advanced Applications, 1999

[27] J. B. Odubiyi, D. J. Kocur, S. M. Weinstein, N. Wakim, S. Srivastava, C. Gokey, J. Graham, SAIRE – A Scalable Agent-Based Information Retrieval Engine, Proc. Autonomous Agents 97

[28] M.T. Özsu, P. Valduriez, Principles of Distributed Database Systems, 2nd ed., Prentice-Hall, 1999.

[29] Steels, Corporate Knowledge Management. Proc. International Symposium on the Management of Industrial and Corporate Knowledge, 1993, 9-30

[30] L. Van Elst, A. Abecker, Domain Ontology Agents in Distributed Organizational Memories. Proc. Workshop on Knowledge Management and Organizational Memories, IJCAI 2001.

[31] P.C. Weinstein, W.P. Birmingham, E.H. Durfee. Agent-Based Digital Libraries: Decen-tralization and Coordination. IEEE Communication Magazine, pp. 110-115, 1999

[32] M. Wooldridge, N. Jennings, D. Kinny, The Gaia Methodology for Agent-Oriented Analysis and Design, Autonomous Agents and Multi-Agent Systems, 2000, 3(3):285-312.