A Mediator-Based Architecture for Capability Management

Nacer BOUDJLIDA (email: nacer@loria.fr)
LORIA-University Henri Poincaré Nancy 1, Campus Scientifique
BP 239, 54506 Vandœuvre Lès Nancy CEDEX (F)
(In Proc. of the 6th IASTED Intern'l Conf. Software Engineering and Applications, Cambridge, USA, Nov. 2002, p.45-50.)

ABSTRACT
The capture, the structuring and the exploitation of the expertise or the capabilities of an "object" (like a business partner, an employee, a software component, a Web site, etc.) are crucial problems in various applications, like cooperative and distributed applications or e-business and e-commerce applications. The work we describe in this paper concerns the advertising of the capabilities or the know-how of an object. The capabilities are structured and organized in order to be used when searching for objects that satisfy a given objective or that meet a given need. One of the originality of our proposal is in the nature of the answers the intended system can return. Indeed, the answers are not Yes/No answers but they may be cooperative answers in that sense that when no single object meets the search criteria, the system attempts to find out what a set of "complementary" objects that do satisfy the whole search criteria, every object in the resulting set satisfying part of the criteria. In this approach, Description Logics is used as a knowledge representation formalism and classification techniques are used as search mechanisms.

KEY WORDS
Knowledge Representation, Database, Description Logic, Classification, Mediation, Distribution.

1 Introduction

In many situations and applications, one is faced to the problem of discovering "entities" that satisfy given requirements. On the other hand, most often, information retrieval in a distributed context, like the World Wide Web, usually lacks selectivity and provides large answer-sets due to the fact that the available information sources are not well structured nor well classified. As a result, most of the systems return yes/no answers (i.e. either they find out answers to a given query or they fail). The work we describe here concerns the publishing of the capabilities of given entities (i.e. what functionalities a given entity is offering, what expertise it has and so on). The capabilities are organized and structured in order to be exploited when searching for entities that satisfy an objective or that meet given needs (searching for a component in the context of component-based design and component-based programming, searching for a business partner with a given expertise, and looking for an employee whose records and expertise satisfy a given work position profile are application examples of our work). It is clear that these needs require the capture and the organization of the entities capabilities together with the classification of the entities and their capabilities. In this work, we adopted an object-oriented knowledge representation using description logics. From the system architecture point of view, we opted for a model based on distributed and cooperative mediators (or traders). A significant originality of our approach resides in the type of answers we aim at providing. Indeed, when no single entity satisfies the search criteria, the systems attempts to determine a set of *complementary* entities who, when grouped together, satisfy the criteria.

The presentation of this work is structured as follows. In section 2, we expose some motivations and possible application domains together with the architecture of the target system. Section 3 briefly introduces elements of description logics. Section 4 shows how description logics is used for entities capabilities management and retrieval. The current implementation status is also presented in section 4, while concluding remarks are in section 5.

2 Motivations and Architecture

The dynamic discovery of services or capabilities an "entity" offers, has different application domains. Component-based programming, electronic business (*e-business*) and even enterprise knowledge management [1] are among these application domains. Indeed, in the first domain, a software component is usually described providing the services (or functionalities) it offers. Usually, the component services are only described by there interfaces (the service *signatures* that mention the inputs, the outputs and the possible exceptions). It is clear that the only syntactic description of a component is not satisfactory when looking for a specific service: an additional semantic description is required. Moreover, the elicitation of possible relationships among services may contribute to find out "the best" service or the "the best complementary" services that satisfy a search query.

In *e-business*, our work can be applied in the constitution of business alliances or when looking for business partners. For example, within a given project, our target system may help in retrieving possible partners, i.e. partners that have the required expertise to be associated in the project development. Furthermore, we also feel that the

work depicted hereafter may also serve for semantic-based discovery of Web services [2, 3].

The achievement of our goals clearly requires conceptualizing and structuring the knowledge that concern the "entities" in a given application domain. In [4], "entities" are design fragments that are described thanks to keywords, the relationships between the fragments are constituted by metrics that measures the similarity between fragments. In [5], entities are object-oriented software components and description logics [6, 7] (see section 3) is used, notably, to describe their intended semantics together with possible constraints involving objects methods. In [8], "entities" are software objects and the capabilities of a software object are specified giving a syntactic part (signatures of the methods or operations the object offers) and a semantic part expressed as logical expressions (a pre-condition and a post-condition are associated with every object's method). The syntactic conformance of a method to a query is trivial and the semantic conformance uses theorem proving techniques. The work we are reporting on is a continuation of [8]: we investigate alternatives to first-order logic for describing the semantics of an "entity" and establishing relationships among entities. Description logics, as a candidate knowledge representation formalism, has the notable merit that a single mechanism, the classification mechanism, serves at the same time to build and to query extendible domain descriptions.

From another standpoint and considering the aimed goals, an entity has to be fitted with means to describe itself, i.e. to explicit (we'll say to publish or to export) its capabilities to enable foreign entities to explore the published capabilities. A very natural way, we choosed a trader (also called mediator) based architecture [9]. The principles of this type of architecture is briefly recalled hereafter.

In this kind of architecture, an "entity", called *exporter*, publishes its capabilities at one or more mediators sites (arrow (a) on figure 1). Entities, called *importers*, send requests to the mediator asking it to find out exporters fitted with a given set of capabilities (arrow (b) on figure 1). The mediator explores its capability and knowledge base to try to satisfy the request. The capability search process is founded on the exported capabilities and on relationships between them, these relationships being transparently established by the mediator. When the request can be satisfied by some exporters known from the mediator, the references of those exporters are sent back to the importer (arrow (c) on figure 1). Nevertheless, satisfying the request falls into different cases: *(i)* there exists exporters that fully satisfy the request; *(ii)* there exists exporters that partly satisfy the request, but when "combining" or composing the capabilities of different exporters one can fully satisfy the request; *(iii)* no single exporter nor multiple exporters satisfy the request. In the later situation, the mediator may initiate a cooperation process with other mediators to attempt to satisfy the request (arrow (d) on figure 1). The cooperation depth, i.e. the number of mediators that may be involved in the cooperation process and the number of exporters in a combination, may be controlled and constrained by the request. For instance, while searching for business alliance partners, one can constrain the search process to at most two possible partners: that means that the answer to the request cannot be constituted from more than two exporters (identified by a single mediator or by cooperative mediators).

The ultimate goal of this work is the design and the development of a set of services to export, import and mediate as well as mechanisms for mediator cooperation. The coming sections detail the adopted approach and foundations, beginning with a short introduction to description logics (DL).

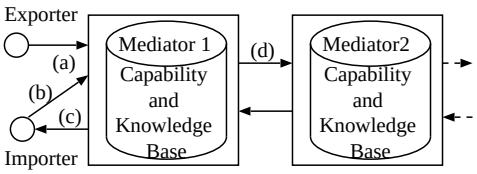

Figure 1. The Mediator-based Architecture.

3 An Introduction to DL

DL[1] is a family of knowledge representation languages where description of a world is built using *concepts*, *roles* and *individuals*. Two levels of knowledge are considered: *(i)* the terminological level (also called *TBox*) where concepts and roles are represented and manipulated, and *(ii)* the assertion (or fact) level (also called *ABox*) where individuals are represented and manipulated. The *subsumption relationship* enables organizing the concepts and the roles according to there degree of generalization and a knowledge base can then be viewed as a hierarchy of concepts possibly associated with a hierarchy of roles. Further, the classification and the instanciation mechanisms are the basic reasoning mechanisms in DL.

Section 3.1 introduces basic definitions and sections 3.2 and 3.3 respectively define the subsumption relationship and the classification process.

3.1 (Informal) Basics of DL

In DL, *concepts* model classes of individuals (sets of individuals) and they correspond to generic entities in an application domain. An *individual* is an instance of a concept. *Roles* model binary relationships among the individual classes. A concept is specified thanks to a structured description that is built giving *constructors* that introduce the roles associated with the concept and possible restrictions associated with some roles. Usually, the restrictions constrain the range of the binary relationship that is defined by a role and the role's cardinal[2].

[1] The essentials of this section is inspired from [10]. [6] is a "fabulous" site on the topic.

[2] The cardinal of a role fixes the minimum and the maximum numbers of elementary values of the role. *Elementary values* are instances of a

PERSON $\dot{\leq}$ $\top$
SET $\dot{\leq}$ $\top$
MAN $\dot{\leq}$ PERSON
WOMAN $\dot{\leq}$ (and PERSON (not MAN))
member $\dot{\leq}$ toprole
head $\dot{\leq}$ member
TEAM $\doteq$ (and SET (all member PERSON)
(atleast 2 member))
SMALL-TEAM $\doteq$ (and TEAM (atmost 5 member))
MODERN_TEAM $\doteq$ (and TEAM (atmost 4 member)
(atleast 1 head) (all team head WOMAN))

Figure 2. DL Example (inspired from [11]).

Concepts are primitive concepts or defined ones. Primitive concepts may be considered as atoms that may serve to build new concepts (the defined concepts). Similarly, roles may be primitive roles as well as defined roles. In the figure 2, PERSON and SET are primitive concepts: they are introduced using the $\dot{\leq}$ symbol and they are linked to a "TOP" concept ($\top$)[3]; TEAM, SMALL-TEAM and MODERN-TEAM are defined concepts (they are introduced using the $\doteq$ symbol. The *and* constructor enables defining concepts as a conjunction of concepts: these concepts are the immediate ascendants of the defined one. The *all* constructor constrains a role's range and the *atleast* and *atmost* constructors enable specifying the role's cardinals. Finally, the *not* constructor only applies to primitive concepts. On figure 2, a TEAM is a SET of PERSONs and it is composed with at least two members. MAN and WOMAN are incompatible primitive concepts and the set of women is in the MAN *complement* set [12, 13].

As for classical logic, a formal semantics may be associated with the defined concepts and roles. Indeed, using an interpretation domain, one can then define the notions of concept and role interpretations as well as the notions of concept satisfiability, equivalence and incompatibility (see [10, 6, 7] for further details).

3.2 The Subsumption Relationship

Subsumption is the fundamental relationship that may hold among described concepts. Intuitively, a concept C (PERSON, for example) *subsumes* a concept D (MAN, for example) if the set of individuals represented by C contains the set of individuals represented by D. More formally, C subsumes D and it is denoted as $D \sqsubseteq C$ (or D is subsumed by C) if and only if $D^I \subseteq C^I$ for every possible interpretation I. C is called the *subsuming* concept and D the *subsumed* one.

concept or basic values like integers, strings and so on.

[3] Intuitively the TOP concept is the "most general one" and it contains all the individuals while the BOTTOM concept ($\perp$) is the most specific one and is empty.

The subsumption relationship defines a hierarchical structure among a set of concepts and it may graphically be represented as an acyclic oriented graph rooted at $\top$ (see the left hand-side of figure 3). The graph-nodes are concepts and the edges are instances of the relationship. The classification process operates on that hierarchy of concepts.

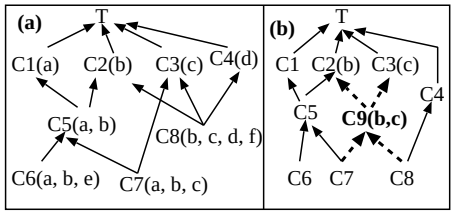

Figure 3. (a) A hierarchy H of concepts; (b) H after inserting concept C9.

3.3 The Classification Process

The classification process aims at determining the position of a new concept in a given hierarchy. It operates according to a three-phase loop: *(i) Instanciation*: create a concept C that is a concept to be inserted or to be retrieved in the hierarchy; *(ii) Classification*: traverse the hierarchy, retrieve the most specific concepts that subsume C (the "immediate ascendants" of C), retrieve the most general concepts that are subsumed by C (the "descendants") and insert C into the hierarchy if it does not exist; *(iii) Operation*: update the links in the hierarchy while inserting the new concept; the link updates may generate a loop in this process. The right hand-side of figure 3 illustrates the process as applied to the concept $C_9(b, c)$.

In this framework, a query is represented as a concept Q to be classified in a given hierarchy. The result of the query is the set of instances of the concepts that are subsumed by Q. One should notice that description logics has been used in the database domain [12, 14][4], not only for querying but also for database schema design and integration [15], for reasoning about queries (query containment, query refinement, query optimization, ...) [16]. From our concern, description logics is used for query purposes with the special objective to produce more than "Yes or No" results. For example, in our approach, the evaluation of $C_9(b, c)$ as a query against the graph concept hierarchy on figure 3, would not return an empty result but it would return "a wider answer" that contains $C_7(a, b, c)$ and $C_8(b, c, d, f)$. Let us now detail the approach.

4 Capability Management

The possible application domains being introduced together with the adopted architecture and the representation formalism, this section describes our contribution to the

[4] See also [6] for the proceedings of the various "Knowledge Representation meets DataBase" (KRDB) Workshops.

mediator-based capability management. Section 4.1 shows how description logics is used to represent an application domain knowledge, section 4.2 introduces classification algorithms that have been implemented, section 4.3 explains how these algorithms serve in capability retrieval and, finally, section 4.4 reports on the current status of the implementation.

4.1 Application Domain Representation

As figured by the meta-model in figure 4 (expressed in a UML class diagram notation), we assume that a mediator manages one or more domain knowledge represented as concept hierarchies. We consider that a given domain may be described thanks to a set of activities (we'll call *functions*)[5] and that an activity is described by the set of the required skills or capabilities to carry it out.

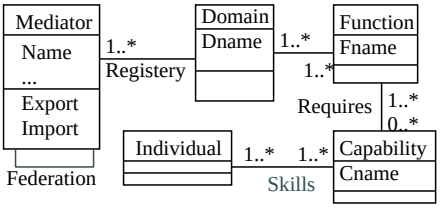

Figure 4. Class Diagram.

Using the description logics terminology, a domain is an immediate descendant of the TOP node concept, a function is a descendant (immediate or not) of the corresponding domain(s) and a capability is a property (an attribute) of a function. Let C_F be the set $\{c_1, c_2, \cdots, c_n\}$ of capabilities that are required by a function F. In the graph representation a dependency edge exists from a function F_2 to a function F_1 if the capabilities that are required by F_1 includes the ones required by F_2. The ascendants of F are the *Most Specific Subsuming Concepts (MSSC)*, i.e. they constitute a set F_{MSSC} of functions $F_1, \cdots, F_n$ such that $\forall i, 1 \leq i \leq n, C_{F_i} \supseteq C_F$. A similar way, the descendants of F are the *Most General Subsumed Concepts (MGSC)*, i.e. they constitute a set F_{MGSC} of functions $F_1, \cdots, F_m$ such that $\forall i, 1 \leq i \leq m, C_{F_i} \subseteq C_F$.

In the hierarchical representation of a domain, every described function satisfies $F_{MSSC} \supseteq F \supseteq F_{MGSC}$. Moreover, we assume that a list of individuals is attached to every node: that list contains references to the instances of the object class represented by the node. As an example, the figure 5 shows a partial representation of the software engineering domain limited to two functions, analysis and programming, programming being itself specialized with regard to individual capabilities in given programming languages. On that figure, the programming capabilities classification are denoted $F_0, \cdots, F_5$ and the references to the individuals that possess the skills required by every F_i are not represented. Further, the leaves of the hierarchy are linked to a bottom node whose usefulness is described in the next section.

[5] Domain decomposition into functions may be user-defined as it may be based on standard categorizations or on defined ontologies, for example.

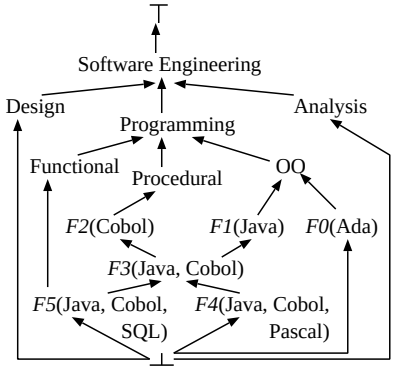

Figure 5. (Partial) Software Engineering Hierarchy.

4.2 The Classification Algorithms

As stated before, classification is a process that enables discovering whether a subsumption relationship holds between a concept X, and those present in a hierarchy H. The classification process is decomposed into 3 steps:

1. Retrieve the most specific subsuming concepts of X (denoted MSSC(X), further);
2. Retrieve the most general concepts subsumed by X (denoted MGSC(X), further);
3. (Possibly) Update the links between X and its MSSC and its MGSC.

Let us detail steps 1 and 2, step 3 being directly derived from the two first steps. In the *first step*, the set of concepts that subsume X is computed thanks to a top-down traversal of the concept hierarchy as sketched by the following algorithm skeleton inspired from [11] and where C denotes the current traversed node.

```
Function Explore4Mssc(C, X)
 If C does not subsume X Then return({})
 Else // C is temporarily the MSSC
   If C is a leaf of the hierarchy Then return({C})
   Else mssc = {}
     For every descendant D of C do
       mssc = mssc ∪ Explore4Mssc(D, X) od
     If mssc = {} Then return(C) Else return(mssc)
```

In the *second step*, *mgsc*, the set of concepts that are subsumed by X are determined by, roughly, exploring the descendants of the X's most specific subsuming concepts.

The analysis of the above algorithm skeleton leads to the following: when the algorithm returns a non empty set *mssc*, *mgsc* is computed exploring the descendants of the concepts in *mssc*; but if *mssc* is empty, an additional graph traversal is required for computing *mgsc*. From the results of this analysis, we decided to propose two distinct algorithms, one for computing *mssc* and an other for computing

mgsc. Distinguishing the computing of *mssc* from the one of *mgsc* reduces the number of visited nodes and it favours early graph-pruning.

Indeed, in the preceding algorithm, while top-down computing *mssc*, since a concept may have many super-concepts, some nodes may be visited as many times as their respective number of "father-nodes". Moreover, when a node C is visited, either the concept it represents is a super-concept of X (in such a case C is added to the result set) or it is not and consequently, its descendants have not to be visited: the hierarchy may be pruned at that node. The following adapted algorithm introduces node marks to avoid multiple visits or unuseful visits of a node.

Function Explore4Mssc2(*C, X*)
 If *C is marked Then return({})*
  // The node is already visited
 Else Mark *C*
  If *C does not subsume X*
  Then Mark all the descendants of *C; Return({})*
   Return({})
  Else // *C is temporarily the MSSC*
   // The remaining is unchanged

Considering again the initial algorithm that computes *mssc*, if the computation fails (i.e. it returns an empty set), computing *mgsc* requires another top-down traversal of the hierarchy. But, if X, the domain function to be classified is described by novel capabilities, i.e. by capabilities that are not present in any node of the hierarchy, then MGSC(X) is empty: so the graph traversal was unuseful and can be avoided using a bottom-up traversal strategy rather than a top-down one. The following algorithm enables the bottom-up computing of MGSC(X).

Function Explore4Mgsc(*C, X*)
 If *C is marked Then return({})*
 Else Mark *C*
  If *C does not subsume X*
   // No *C's ascendant is subsumed by X*
  Then Mark all the *C's ascendants; Return({})*
  Else mgsc = {}
   For every ascendant *A of C do*
    mgsc = mgsc ∪ Explore4Mgsc(*A, X*) *od*
   If mgsc = {} Then return(*C*) *Else return(mgsc)*

Let us now apply these concepts and algorithms to our concern, i.e. entities capability management.

4.3 Capability Management

Remember that we aim at retrieving a set of individuals or a set of objects whose skills meets a given set of capabilities: let $\{X_1, \cdots, X_n\}$ denote that set of capabilities (conceptually, $\{X_1, \cdots, X_n\}$ describes a function in the application domain). The request is viewed as a concept X having the given capabilities and the query evaluation consists in "situating" X in the classification hierarchy.

Let $C_X = \{X_1, \cdots, X_n\}$ be the capabilities required from X and let $I(X)$ denote the set of individuals that satisfy C_X. If $I(N_k)$ denotes the set of individuals associated with a node N_k then $I(X) = \cup_{k=1}^{r} I(N_k)$, r being the number of nodes N_k that refer to individuals that satisfy the query. Let $C_{N_k} = \{c_1^k, \cdots, c_{m_k}^k\}$ be the respective capabilities of the nodes N_k. From a procedural standpoint, $I(X)$ is obtained thanks to the computing of MSSC(X) and/or MGSC(X). Let us formally and informally detail the various situations.

Case 1: *The individuals in $I(X)$ exactly satisfy the request*, that means $\forall k \in [1..r] m_k = n \wedge \forall c_j^k, j \in [1..m_k] \wedge c_j^k \in C_{N_k}, \exists X_i \in C_X$ such that $c_j^k = X_i$.

This case arises when $MSSC(X) \equiv MGSC(X)$: in other words, a single node satisfies the query (r = 1). For example, considering figure 5, the node F_3 satisfies the request for individuals having skills in Java and Cobol programming languages.

Case 2: *The individuals in $I(X)$ have capabilities that are "wider" than those requested.* That means that $\forall k \in [1..r] m_k > n \wedge \forall X_i \in C_X, \exists c_j^k j \in [1..m_k] \wedge c_j^k \in C_{N_k}$ such that $c_j^k = X_i$.

$I(X)$ is the union of the individuals that are associated with the nodes in MGSC(X). As an example, considering $C_X = \{Java, Pascal\}$, $I(X)$ is the individuals associated with the node F_4 on figure 5 .

Case 3: *No single individual satisfies the preceding situations*, but when "put together", several individuals satisfy the requested capabilities, i.e. $H(D)$ being the graph hierarchy of an application domain D and $C_N = \{c_1, \cdots, c_N\}$ being the capabilities of the individuals associated with the node N: $\neg\exists N \in H(D)$, such that $\forall X_i \in C_X \exists c_j \in C_N \wedge X_i = c_j$.

In this situation, the required capabilities may be satisfied by groups of individuals, while every separate group only partly satisfies them. These groups are constituted by MSSC(X). More formally, that means $I(X) = \cup_{k=1}^{r} I(N_k)$ where N_k ($k \in [1..r]$) are nodes in $H(D)$ and where $\forall k \in [1..r] C_{N_k} \subset C_X \wedge \cup_{k=1}^{r} C_{N_k} = C_X$. One should notice that the groups capability sets may overlap or not (i.e. $\forall k, k\prime \in [1..r]\, C_{N_k} \cap C_{N_{k\prime}} = \emptyset \vee C_{N_k} \cap C_{N_{k\prime}} \neq \emptyset$). For example, referring again to figure 5, $\{F_0, F_1\}$ is the answer to the request for Java and Ada skilled programmers.

Case 4: *No single individual nor group of individuals totally satisfy the requested capabilities*, but individuals exist who partly possess the required skills. For example, only "incomplete" answers ($\{F_0, F_1\}$) can be returned when looking for individuals having Java, Ada and C programming skills.

Case 5: *None of the requested capabilities is satisfied by any individual or group of individual* (e.g. C++ and C programmers): this situation means that MSSC(X) and MGSC(X) are empty sets.

4.4 Status of the Implementation

The algorithms that have been introduced have been implemented in Java considering the conceptual abstraction of the system as depicted in figure 4. Only the *and* constructor has been considered for the moment: that means that the current implementation only covers the cases 1, 2, 3 and 5. The introduction of constructors other than the only *and* is required to deal with the fourth case. Moreover, one should notice that the fourth and the fifth cases lead to a failure when only one mediator is implied in the request evaluation. But if we assume a federation of mediators, these are typical situations where cooperation between mediators is required. That cooperation will proceed as follows. In situation 5, the whole request is transmitted to a next mediator who will execute the same process as the one that has been sketched in the previous section. In situation 4, the reason of the failure has first to be identified: mainly the unsatisfied part of the request has to be identified. The identification is founded on *concept complement*, defined as: if A is subsumed by B then *comp(B,C) = A and (B, C)* $\equiv A$[6]. Intuitively, the concept complement designates the knowledge that is missing in a concept B to enable B to be an instance of A.

5 Concluding Remarks

In this paper, we presented an approach for modelling, representing and exploiting properties of "objects" called object capabilities. The approach is founded on description logics to provide a semantic description of an application domain. This foundation has the merit to use a single mechanism, the classification process, for at the same time building the domain representation and exploiting it. A "classical" approach, like an object-oriented modelling and a database implementation, has several drawbacks, like the multiplicity of representation and exploitation mechanisms, and it lacks extendibility [17].

The current state of this work does not treat, from the implementation point of view, the problem of the cooperation between mediators. This is part of on-going work. Indeed, when there exists no individual or group of individuals who satisfies requested capabilities, a mediator may initiate a cooperation with another mediator in order to attempt to satisfy the request. The cooperation may be governed by defined policies and constraints (for example, to constraint the size of the answer set to at least or at most n groups of individuals). The extensions to this work requires additional constructors in the description logics language. It also requires a kind of query decomposition mechanism to determinate which parts of an initial request have to be addressed to the cooperating mediators. It is clear that an additional level of knowledge is required: this level will inform on the capabilities of the mediators themselves (i.e. what domain knowledge does every mediator manage). Finally, in order to integrate our proposal into realistic platforms, we think about integrating the current system with "traditional" databases to enable a more detailed description of an application domain and its individuals.

[6] Two concepts are equivalent if a mutual subsumption relationship holds among them, i.e. $A \equiv B$ if $A \sqsubseteq B \wedge B \sqsubseteq A$.

References

[1] Ikujiro Nonaka and Hirotaka Takeuchi. *The Knowledge Creating Company; How Japanese Companies Create the Dynamics of Innovation.* Oxford University Press, 1995.

[2] UDDI. Universal Description, Discovery and Integration. Technical White Paper, Sept. 2000. (http://uddi.org).

[3] T. Dyck. UDDI 2.0 Provides Ties That Bind, April 2002. (http://www.eweek.com/).

[4] T-D. Han, S. Purao, and V. Storey. A Methodology for Building a Repository of Object-Oriented Design Fragments. In *18th Int. Conf. on Conceptual Modelling, ER'99*, 203–217, Paris, Nov. 1999. Springer Verlag. LNCS 1728.

[5] A. Borgida and P. Devanhu. Adding more "DL" to IDL: Towards more Knowledgeable Component Interoperability. In *21rst Int. Conf. on Software Engineering, ICSE'99*, pages 378–387, Los Angeles, CA, May 1999. ACM Press.

[6] DL-org. Description logics. http//dl.kr.org/.

[7] I. Horrocks. Description Logic: Axioms and Rules. Talk given at Dagstuhl "Rule Markup Technique" Workshop, February 2002. http://www.cs.man.ac.uk/ horrocks/Slides/.

[8] M. Bouchikhi and N. Boudjlida. Using Larch to Specify the Behavior of Objects in Open Distributed Environments. In *Proc. Maghrebian Conf. on Software Engineering and Artificial Intelligence*, 275–287, Tunis, Tunisia, Dec. 1998.

[9] OMG. The Object Model Architecture Guide. http://www.omg.org/

[10] A. Napoli. Une introduction aux logiques de description. Technical Report No 314, INRIA-LORIA, Nancy, 1997.

[11] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*. LNAI 422, Springer Verlag, 1990.

[12] A. Borgida. Description Logics in Data Management. *IEEE TKDE*, 7(5):671–682, 1995.

[13] M. Schmidt-Schauss and G. Smolka. Attribute Concepts Description with Complements. *Artificial Intelligence Journal*, 48(1):1–26, 1991.

[14] N. Boudjlida. Knowledge in Interoperable and Evolutionary Systems. In L. Dreschler-Fischer and S. Pribbenow, editors, *KRDB'95, Workshop on "Reasoning about Structured Objets: Knowledge Representation Meets Databases"*, pages 25–26, Bielefeld, Germany, Sept. 1995.

[15] D. Calvanese and al. Information Integration: Conceptual Modeling and Reasoning Support. In *6th Int. Conf. on Cooperative Information Systems, CoopIS'98*, 280–291, 1998.

[16] C. Beeri, A. Levy, and M-C. Rousset. Rewriting Queries Using Views in Description Logics. In *ACM Symp. on Principles Of Database Systems*, pages 99–108, Tucson, Arizona, 1997.

[17] M.A. Bouneffa and N. Boudjlida. Managing Schema Changes in Object-Relationship Databases. *Proc. 14th Int. Conf. on Conceptual Modelling, OO-ER'95*, 113–122, Gold Coast, Australia, Dec. 1995. Springer-Verlag, LNCS 1021.