Reasoning in Pervasive Environments: an Implementation of Concept Abduction with Mobile OODBMS

Michele Ruta, Floriano Scioscia, Tommaso Di Noia, Eugenio Di Sciascio
Politecnico di Bari, via Re David 200, I-70125 Bari, Italy
{m.ruta, f.scioscia, t.dinoia, disciascio}@poliba.it

Abstract

The paper focuses on an implementation of concept abduction with an Object-oriented Database Management System (OODBMS). OWL-DL Knowledge Bases have been translated to an OO version to enable standard and non-standard inference services as queries over a DB suitable for handheld devices. The framework has been implemented and tested: early experiments are reported.

1 Introduction and Motivation

Canonical matching approaches available in most common mobile standard protocols have been shown to be inadequate in articulated resource discovery scenarios (*e.g.*, wireless sensor networks, disaster recovery or m-commerce applications) [2]. Ranked matchmaking of resource instances according to their semantic descriptions overlapping w.r.t. a request is –on the other hand– an extremely useful feature, allowing to delegate to software agents tasks originally performed only by human users. Furthermore, match explanations can provide useful information to modify or refine the early discovery attempt in a principled way. Nevertheless, required software and hardware facilities restrain the diffusion in mobile contexts of those approaches which can be successfully applied to the Web.

Considering that a logic-based approach to matchmaking and the Object Oriented Programming (OOP) paradigm share some basic notions as class, object (*i.e.*, class instance) and inheritance; and furthermore OODBMS have a good flexibility in managing changes to their elements and to relationships among them, it could be an interesting issue trying to implement inference services exploiting object databases. This paper introduces a possible approach to reasoning in mobility, by presenting a solution to implement a concept abduction algorithm with a mobile OODBMS. It adopts an object-oriented translation of reference Knowledge Bases (KBs) allowing to execute standard and non-standard inference services [4] as queries issued to a mobile database, dealing with non-exact matches (computing approximate results).

Experiments to prove feasibility of the framework have been conducted on db4o[1], which has been employed as reference architecture. Analysis of computational costs points out the expressiveness of adopted formalisms plays a critical role: we selected a sublanguage deriving from OWL DL[2] to model ontologies, requests and resource annotations in the *simple-TBox* (for Terminological Box) hypothesis.

The remaining of the paper is structured as follows: in Section 2 the theoretical framework is presented. Experimental findings are described in Section 3. Conclusions close the paper.

2 OODBMS-based Concept Abduction

Here, preliminary notions are introduced about the adopted language and reasoning tasks. Afterwards we move onto framework description. The reader is supposed to be familiar with basics of Description Logics [1].

Language. The logic language has been selected to be as expressive as possible, while still allowing polynomial-time inferences for "bushy but not deep" ontologies. The adopted logic is known as $\mathcal{ALN}$ (Attributive Language with unqualified Number restrictions) [1]. Furthermore, ontologies are bound by the *simple-TBox* constraints defined in Description Logics literature [5]. Simple-TBoxes can be "embedded" into the concepts through a processing step, known as *unfolding* [1]. Such process transforms every concept into an equivalent one, where references to the ontology are not needed any more. Every concept description can be rewritten in a normal form –called *Conjunctive Normal Form* (CNF)– by applying well-known normalization rules, reported *e.g.*, in [4]. The CNF of concepts can also take unfolding into account. In this way, every satisfiable concept C can be divided into four components as:

[1]db4o: Native Java and .NET Open Source Object Database, http://www.db4o.com/

[2]OWL Web Ontology Language, W3C Recommendation 10 February 2004, http://www.w3.org/TR/owl-features/

978-0-7695-3801-3/09 $26.00 © 2009 IEEE
DOI 10.1109/WI-IAT.2009.29

$C_n \sqcap C_{\geq} \sqcap C_{\leq} \sqcap C_{\forall}$. The component C_n is the conjunction of all concept names $A_1, \ldots, A_h$. The component $C_{\forall}$ conjoins all concepts of the form $\forall R.D$, one for each role R, where D is recursively in normal form. The components $C_{\geq}$ and $C_{\leq}$ are the conjunction of all at-least and at-most number restrictions respectively, no more than one for every role in each component (the maximum at-least and the minimum at-most for each role), including $(\leq\ 0\ R)$ in $C_{\leq}$ for every conjunct of $C_{\forall}$ of the form $\forall R.\bot$. Finally, we define the *Quantification Nesting* (QN) of a concept as:
$QN(C) = 0$;
$QN(\forall R.C) = QN(C) + 1$;
$QN(C_1 \sqcap C_2) = max(QN(C_1), QN(C_2))$.

This formalization allows polynomial-time algorithms for standard and non-standard reasoning tasks that form a non-monotonic semantic matchmaking framework [3]. Let us consider a request D and a supplied resource S, both represented as concept descriptions w.r.t. a common TBox $\mathcal{T}$ in $\mathcal{ALN}$ language. The proposed system currently implements the following services for semantic matchmaking.
1. Retrieval of compatible supplies. Firstly, each available resource is checked against the request for semantic compatibility, which occurs if their conjunction is satisfiable w.r.t. $\mathcal{T}$, $D \sqcap S \not\sqsubseteq_{\mathcal{T}} \bot$. Compatible resources are retrieved from the Knowledge Base, whereas incompatible ones are not processed any further by the matchmaker.
2. Concept Abduction. Given a compatible resource S, the evaluation of the semantic correspondence with the request is modeled as a **Concept Abduction Problem** (CAP) [3]. We write $H = solveCAP(S, D)$ to indicate that H is what has to be hypothesized and added to S in order to completely satisfy the request D.

KB modeling. The framework has been implemented with the aim to: (i) efficiently store a KB on the device secondary memory; (ii) enable inference services described above; (iii) be modular and scalable in order to further integrate new reasoning services. A central question is the KB modeling. In order to perform inferences via proper queries over the database, ABox instances and TBox are stored in the OODB. Managed data are structured in three different layers, each one providing a specific view of the KB.

Physical Layer. It maintains the OWL file of the knowledge base, further translated to a proper db4o database file. KB instances (individuals) are unfolded and expressed in CNF.

High Level Data Structures. They correspond to the logical layer. With reference to Figure 1, component classes are: *i) Item*: each KB individual is an instance of this class. The individual name is reported in the `name` attribute; *ii) SemanticDescription*: models the individual semantic annotation in CNF. Hence, it is expressed as aggregation of $C_n, C_{\geq}, C_{\leq}, C_{\forall}$ components, each one stored in a different Java Collection. The `abduce`

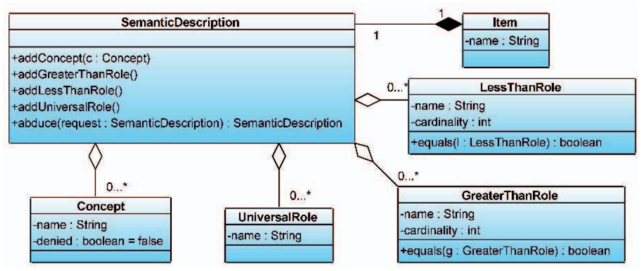

Figure 1. HLDSs UML class diagram

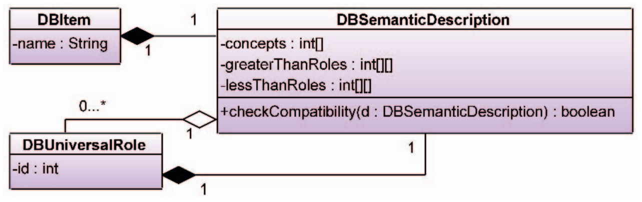

Figure 2. LLDSs UML class diagram

`(SemanticDescription request)` method implements the abduction algorithm; *iii) Concept*: models an atomic concept A_i of the C_n set; the `name` variable will contain the concept name and the `denied` one, if set to *true*, allows to express $\neg A_i$; *iv) GreaterThanRole* (respectively *LessThanRole*): model number restrictions belonging to $C_{\geq}$ and $C_{\leq}$ sets. The role name and cardinality are stored in the homonym variables; *v) UniversalRole*: each instance of this class refers to a universal restriction $\forall R.D$ belonging to the $C_{\forall}$ set where R will be stored in the `name` variable and D will be a *SemanticDescription* class instance.

Low Level Data Structures. In order to put previous information in a more compact form and to optimize them for the inference algorithms, the HLDSs are translated into corresponding LLDSs to be manipulated by the DBMS. The parsing of an HLDS to produce a LLDS involves the translation of Java Collections into arrays and the conversion of role and concept strings into integer data. The latter operation, called *lexical normalization* [6], is needed because integers require less memory and can be searched and compared much more cheaply than strings. Each role or atomic concept is mapped to a unique positive value; furthermore, if k_c maps an atomic concept C, then $-k_c$ will map $\neg C$. The correct correspondence between a string and the related integer value is maintained using two different Java structures, namely an *ArrayList* and a *SortedMap*. The former is a list of elements directly accessible via a specific index. It is used to get the concept (or role) corresponding to a given integer value which is exactly the access index for that element. Such a structure allows a very quick inverse translation from LLDSs to HLDSs. The *SortedMap* is a hash map exploited for the direct parsing from concept (or role) names to numerical values, *i.e.*, from HLDSs to LLDSs. The little redundancy given by having two twin structures is largely repaid by a consistent speed-up in translations string-integer. Figure 2 shows the UML diagram of LLDSs. In what follows each class is detailed: *i) DBItem*: each instance of this class corresponds to only one instance of the above *Item* HLDS. These two classes basically differ only for Java implementation details. Notice that, leaving out the correspondence maps, individual names are the only strings the database uses; *ii) DBSemanticDescription*: also in this case there is a one-to-one correspondence with the above *SemanticDescription*. The conversion from LLDSs to HLDSs is performed as follows: *ii.1 Concept*: each object is converted into an integer value and further inserted into an array. To this aim the correspondence maps are exploited after the lexical normalization described before; *ii.2 GreaterThanRole* (*LessThanRole* respectively): bi-dimensional integer arrays are used. As for concepts, the restricted role is converted into an integer value; the cardinality is expressed with a second value associated to each restriction; *iii) DBUniversalRole*: it maps the correspondent HLDS *UniversalRole*; the integer `id` attribute replaces the `name` string following the correspondence maps.

Abduction Implementation. Before running the abduction algorithm an early compatibility check has to be made aiming at the exclusion of incompatible resources. Note that the *Transparent Activation* [7] in db40 allows to load into main memory only the components to be compared with the request, so avoiding complex and unnecessary role fillers. The compatibility check is executed at the DB layer and only compatible objects will be actually loaded in memory for further inference operations. Retrieval of all compatible resources with a given request is trivially obtained with the following *native query* [7]:

```
List potentialDBItems = db.query(new Predicate() {
    public boolean match(DBItem dbItem) {
        return dbItem.checkCompatibility(demand,
                DBSemanticDescription);
    }
});
```

The argument of the `query()` method is a predicate defining a `match()` method. `match()` executes the comparison among request and individual descriptions exploiting `checkCompatibility()` of the *DBSemanticDescription* class. The query returns a set of compatible resource instances in LLDS form, to be translated in the corresponding HLDSs. The `abduce()` method of the *SemanticDescription* class, applied to a *SemanticDescription* object, accepts in input the request description and returns not covered request features. They are modeled in a new *SemanticDescription* instance called `uncovered`.

```
if (!this.concepts.contains(requestConcept))
    uncovered.addConcept(requestConcept);
```

Notice that the `contains()` method of the Java Collection interface uses the `equals()` method to select objects in the collection. To reach the algorithm purpose this method has been overridden in the *Concept* class: its new implementation returns *true* iff two concept names in comparison are exactly the same. In both second and third loops, numerical restrictions are considered. The `contains()` method is used as explained before: in the *GreaterThanRole* and *LessThanRole* classes, `equals()` has been redefined in order to return *true* only if role names coincide and related cardinality is less than (respectively greater than). The last loop refers to universal quantifiers. When a universal quantifier of the same role is in both the request and the supplied resource, the abduction algorithm is executed recursively on the respective fillers.

3 Experimental Results

Running time and memory usage tests were performed on a Toshiba SA50-432 notebook PC, equipped with 1.5 GHz Intel Centrino CPU, 512 MB RAM and Microsoft Windows XP O.S.. Three Knowledge Bases of different size and complexity were used (namely *Toys*, *Clothing* and *Electronics*). In addition, 48 artificial KBs were randomly generated.

Retrieval of compatible instances and concept abduction were tested on the three realistic KBs. For the first and the second one, having a maximum QN of 2, a set of 5 requests was prepared; for the third KB, having a maximum QN of 12, a set of 10 requests with varying complexity was built. For each request, every instance of the KB was checked for compatibility as a first step, then abduction was performed between request and resources which passed the first check. Turnaround time and main memory peak were measured at each stage. Each test was repeated 5 times and average values were taken. Figure 3-a shows the memory usage peak for the retrieval of compatible instances. The QN of each request is reported along the horizontal axis, while the number of retrieved instances is displayed in white upon each bar in the graph. Memory usage appears to be dependent on the size and complexity of the KB, regardless of both the request QN and the retrieved instances. In idle state –*i.e.*, after DB creation and before a request is submitted– there is little variation. Figure 3-b likewise reports the memory usage peak for the abduction stage. Values for requests C5 and E3 are blank because no compatible instance was found, so abduction could not be performed. The average (6.1 MB) and worst-case (19.6 MB) values are congruent with memory amounts of typical current mobile devices. Figure 4 reports results for aggregate turnaround time. Time mostly depends on the KB complexity. Overall performance can be

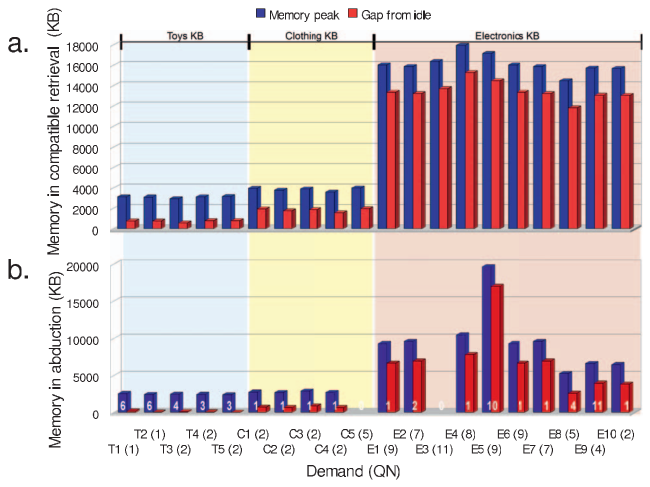

Figure 3. Main memory usage peak

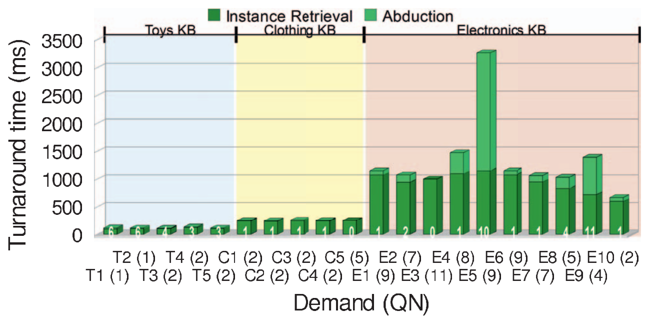

Figure 4. Turnaround time

deemed as satisfactory, since all but one query on the largest KB require less than 1.5s. The duration of retrieval of compatible instances shows a mild correlation with the QN of the request. No significant relation can be determined with retrieved instances. This is likely due to the OODBMS approach, which performs instance retrieval and compatibility check with a set of queries on the DB. On the other hand, the duration of the abduction process is highly dependent on both the KB complexity and the number of executed abduction tests. This can be explained because the algorithm acts upon in-memory HLDSs with no involvement of the DB. This could also justify the strong correlation between time and memory usage of the abduction stage.

4 Conclusion

We proposed an implementation of semantic-based matchmaking and logical explanation for mobile Web and ubiquitous computing: structured queries over a OODB properly modeling the reference KB have been exploited. Proposed approach has been implemented and simulated to provide early verification of its effectiveness.

References

[1] F. Baader, D. Calvanese, D. Mc Guinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2002.

[2] H. Chen, A. Joshi, and T. Finin. Dynamic Service Discovery for Mobile Computing: Intelligent Agents MeetJini in the Aether. *Cluster Computing*, 4(4):343–354, 2001.

[3] S. Colucci, T. Di Noia, A. Pinto, A. Ragone, M. Ruta, and E. Tinelli. A non-monotonic approach to semantic matchmaking and request refinement in e-marketplaces. *International Journal of Electronic Commerce*, 12(2):127–154, 2007.

[4] T. Di Noia, E. Di Sciascio, and F. Donini. Semantic matchmaking as non-monotonic reasoning: A description logic approach. *Journal of Artificial Intelligence Research (JAIR)*, 29:269–307, 2007.

[5] F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in Description Logics. In G. Brewka, editor, *Principles of Knowledge Representation: Studies in Logic, Language and Information*, pages 191–236. CSLI Publications, 1996.

[6] I. Horrocks and P. Patel-Schneider. Optimizing description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

[7] H. H. S. Edlich and, R. Hörning, and J. Paterson. *The Definitive Guide to db4o*, apress edition, June 2006.