Non-standard inference services for mobile computing: concept abduction via m-OODBMS

Michele Ruta, Floriano Scioscia, and Eugenio Di Sciascio

Politecnico di Bari, Via Re David 200, I-70125, Bari, ITALY
[m.ruta,f.scioscia,disciascio]@poliba.it

Abstract. Though increased potentialities of handheld devices allow to apply discovery approaches designed for the Semantic Web, care has to be paid in re-designing original frameworks and algorithms. The paper presents a revision of basic inference services leveraging object-oriented Database Management System to perform semantic matchmaking and provide logical explanations. OWL-DL Knowledge Bases have been properly migrated towards an object oriented version to enable reasoning by addressing proper queries to a mobile DB. The proposed framework has been implemented and tested: preliminary results have been reported.

1 Introduction

Most common matchmaking approaches have been shown to be inadequate in unpredictable and volatile resource discovery scenarios (*e.g.*, wireless sensor networks, disaster recovery or mobile commerce applications) [1]. Solutions devised for wired and stable scenarios are impractical to be transferred to mobile and ubiquitous computing. Particularly, logic-based matchmaking –borrowed from approaches tailored for the Semantic Web– provides useful means to accomplish advanced discovery tasks in ad-hoc contexts, but it is very hard to be successfully applied there.

Considering that a generic semantic-based matchmaking and the Object Oriented Programming (OOP) paradigm share some basic notions such as class, object (*i.e.*, class instance) and inheritance ones, and furthermore that Object Oriented Data Base Management Systems (OODBMSs) have a good flexibility in managing changes occurred to their elements and to relationships among them, it could be an interesting issue trying to devise basic inference services exploiting object DBMS. This paper introduces a possible approach to reasoning in mobility, implementing two basic inferences for retrieving compatible resources w.r.t. a request and computing concept abduction. The proposed approach translates into an object oriented fashion the reference Knowledge Bases (KBs) allowing to execute reasoning tasks by issuing queries to a mobile database.

Experiments to prove the feasibility of the proposal have been conducted on a Java-based open source OODBMS, *i.e.*, db4o[1], and evidence the expressiveness of adopted formalisms plays a critical role: we selected the $\mathcal{ALN}$ Description

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

Logic (DL) and the 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 the next Section we report on most relevant related work about reasoning (particularly in pervasive environments); afterward, in Section 3 we move on to the presentation of the proposed framework and approach. Relevant experiments we carried out are described in Section 4 and finally conclusions close the paper.

2 Related work

In a Knowledge Representation System (KRS), domain knowledge can be declared explicitly –in a suitable logic-based language– and manipulated by general-purpose algorithms. An interface of *reasoning services* is provided to end-user applications. Hence KRSs are also called *reasoners*. From an architectural standpoint, the role of a KRS has always been similar to a DBMS. Both act as back-end repositories where knowledge of the problem domain can be inserted (forming a KB) and upon which automated reasoning tasks can be performed so as to extract implicit knowledge. Nevertheless, this approach is effective only as long as relatively large computing resources and a stable network infrastructure are available. The Semantic Web vision posed new technological challenges to standard DL-based KRSs. In a semantic-enabled WWW, available resources should be annotated w.r.t. OWL-DL ontologies. This allows the use of DL-based reasoners to infer new information from the one available in annotations [2]. The large scale of managed knowledge and the complexity of reasoning procedures were the main issues. Research has made significant progress in devising optimization techniques to achieve acceptable performance for reasoning services applied to adequately expressive languages [3, 4].

A still different approach is needed to adapt KR technologies and tools to mobile and pervasive computing applications. Embedding reasoning capabilities into mobile computing devices appears as a needed condition for fully decentralized semantic-based applications in pervasive contexts. Pocket KRHyper [5] was the first reasoning engine for mobile devices. It was built as a Java ME (Micro Edition) library. It supports the Description Logic $\mathcal{ALCHIR}+$. Pocket KRHyper provides standard satisfiability and subsumption inference services. They were exploited by the authors in a DL-based matchmaking framework between user profiles and descriptions of mobile resources/services [6]. That solution has some points in common with the one proposed in the present work, particularly the use of two queries for match evaluation. That scheme, however, can only distinguish among full, potential and partial match types (adopting the terminology of [7]). This is due to the fact that satisfiability and subsumption provide only binary "true/false" answers. Our abductive matchmaking approach enables a more fine-grained semantic ranking as well as explanation of outcomes. Müller

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

et al. [8] reported their early work on the development of a mobile DL reasoner for the Java ME platform, supporting the language $\mathcal{ALCN}$: no further result on their effort was published, though. Both the above systems are based on adapting tableaux algorithms –used in most "wired" DL reasoners– to mobile computing platforms. This could allow more expressive languages to be used, but efficient implementation of useful non-standard reasoning services is still an open problem due to limitations in running time and memory.

In [9], a different approach was proposed to adapt logic-based reasoning services to pervasive computing contexts. It was based on simplifying the underlying logical languages and admitted axioms in a KB, so that standard and non-standard reasoning services could be reduced to set-based operations. KB management and reasoning services were then executed through a data storage layer, based on a mobile RDBMS (Relational DBMS).

3 Concept Abduction with OODBMS

In what follows, preliminary notions are introduced about the adopted language and reasoning tasks[3] and the framework implementation is described.

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. *Concept descriptions* can include only: (i) concept names: A, B; (ii) conjunctions: $A \sqcap B$; (iii) universal role quantifications: $\forall R.C$; (iv) unqualified number restrictions: $(\geq\ n\ R), (\leq\ n\ R)$, where R is a role (a.k.a. property), C is a concept description and n is a non-negative integer. Note that, for any role R, $(\geq\ 1\ R)$ is semantically equivalent to the unqualified existential restriction $\exists R$.

The following ontology axioms are allowed: (i) definition: $A \equiv B$; (ii) inclusion: $A \sqsubseteq B$; (iii) disjoint group: $disj(A_1, \ldots, A_k)$ where $A, A_1, \ldots, A_k$ are *concept names*. This language is known as $\mathcal{ALN}$ (Attributive Language with unqualified Number restrictions) [10].

$\mathcal{ALN}$ provides a minimal set of constructs that allow one to represent a concept taxonomy, disjoint groups, role restrictions ($\mathcal{AL}$), and restrictions on the number of fillers of a role ($\mathcal{N}$). Furthermore, ontologies are bound by the *simple-TBox* constraints defined in Description Logics literature [11].

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 [12]. We observe that normalization of a concept preserves semantic equivalence w.r.t. models induced by the ontology; furthermore, CNF is unique (up to commutativity of conjunction operator). The normal form of an unsatisfiable concept is simply $\bot$. Every satisfiable concept C can be divided into four components as: $C_n \sqcap C_{\geq} \sqcap C_{\leq} \sqcap C_{\forall}$. 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

[3] The reader is supposed to be familiar with basics of Description Logics [10].

```
Algorithm: checkCompatibility(D, S)
Input: a request D ≡ D_n ⊓ D_≥ ⊓ D_≤ ⊓ D_∀ and a resource S ≡ S_n ⊓ S_≥ ⊓ S_≤ ⊓ S_∀ both satisfiable w.r.t.
       T and in CNF
Output: true if D ⊓ S is satisfiable w.r.t. T otherwise false;
1  foreach concept name A in D_n do
2      if ¬A is in S_n then
3          return false;
4      end
5  end
6  foreach number restriction (≥ n R) in D_≥ do
7      if (≤ m R) is in S_≤ and (m < n) then
8          return false;
9      end
10 end
11 foreach number restriction (≤ n R) in D_≤ do
12     if (≥ m R) is in S_≥ and (m > n) then
13         return false;
14     end
15 end
16 foreach universal quantifier ∀R.E in D_∀ do
17     if ∀R.F is in S_∀ and checkCompatibility(E, F) == false then
18         return false;
19     end
20 end
21 return true;
```

Algorithm 1: Compatibility check

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 $\forall R.\bot$. We define the *Quantification Nesting* (QN) of a concept as the following positive integer:

$QN(C) = 0$;

$QN(\forall R.C) = QN(C) + 1$;

$QN(C_1 \sqcap C_2) = max(QN(C_1), QN(C_2))$.

In the simple-TBox hypothesis, the ontology can be "embedded" into the concepts through a processing step, known as *unfolding* [10].

This formalization allows polynomial-time algorithms for standard and non-standard reasoning tasks that build a non-monotonic semantic matchmaking framework [7]. Let us consider a request D and a supplied resource S, both referred to a TBox $\mathcal{T}$ in $\mathcal{ALN}$ language. The proposed system currently implements the following reasoning services.

1. Retrieval of compatible supplies. Each available resource is checked against the request for semantic compatibility, which occurs if their conjunction is satisfiable w.r.t. $\mathcal{T}$, $D \sqcap S \not\sqsubseteq_{\mathcal{T}} \bot$. Compatibility check is a sub-case of standard *satisfiability* reasoning service and corresponds to request and resource having something in common and no conflicting features. Algorithm 1 is adopted. In the current implementation, compatible resources are retrieved from the KB, whereas incompatible ones are not processed any further by the matchmaker. It is known that useful outcomes could still be provided by means of *concept contraction* inference service [7]. That feature, though, was planned for subsequent versions of our mobile matchmaker.

2. Concept Abduction. Given a compatible resource S, the evaluation of the semantic degree of correspondence with the request is a **Concept Abduction Problem** (CAP) [7]. Algorithm 2 provides a solution to the CAP $H = solveCAP(S, D)$ indicating what has to be hypothesized and added to S in order to completely satisfy the request D. If and only if $S \sqsubseteq D$ (S subsumes

Algorithm: solveCAP(D, S)
Input: a request $D \equiv D_n \sqcap D_{\geq} \sqcap D_{\leq} \sqcap D_{\forall}$ and a resource $S \equiv S_n \sqcap S_{\geq} \sqcap S_{\leq} \sqcap S_{\forall}$ both in CNF, with $D \sqcap S$ satisfiable w.r.t. $\mathcal{T}$
Output: a solution H to the Concept Abduction Problem

```
1  H_A ← ⊤, H_≥ ← ⊤, H_≤ ← ⊤, H_∀ ← ⊤;
2  foreach concept name A in D_n do
3      if A is not in S_A then
4          H_A ← H_A ⊓ A;
5      end
6  end
7  foreach number restriction (≥ n R) in D_≥ do
8      if (≥ m R) is not in S_≥ or (m < n) then
9          H_≥ ← H_≥ ⊓ (≥ n R);
10     end
11 end
12 foreach number restriction (≤ n R) in D_≤ do
13     if (≤ m R) is not in S_≤ or (m > n) then
14         H_≤ ← H_≤ ⊓ (≤ n R);
15     end
16 end
17 foreach universal quantifier ∀R.E in D_∀ do
18     if ∀R.F is in S_∀ then
19         H' ← solveCAP(E, F);
20         H_∀ ← H_∀ ⊓ ∀R.H';
21     else
22         H_∀ ← H_∀ ⊓ ∀R.E;
23     end
24 end
25 H ← H_A ⊓ H_≥ ⊓ H_≤ ⊓ H_∀;
26 return H;
```

Algorithm 2: Concept Abduction

D), then Algorithm 2 returns $H = \top$, meaning that the resource satisfies the request completely, *i.e.*, a full match [7] occurs and no hypothesis is required. Thus, Abduction can be seen as an extension of the standard *classification* (a.k.a. *subsumption*) inference service, also providing an explanation for (missed) subsumption.

Example. A small example can help understand the usefulness of abductive matchmaking. Let us consider a toy TBox $\mathcal{T}$ for the tourism domain, containing just the axioms (i) $Palace \sqsubseteq \exists hasAge \sqcap \exists hasRoof$ and (ii) $Palace \sqsubseteq \neg Church$. Then let us take a request and two resources:

- Request: medieval palace with a courtyard. $R \equiv Palace \sqcap \forall hasAge.MiddleAge \sqcap \exists hasCourtyard$
- Resource 1: medieval church with a courtyard. $S_1 \equiv Church \sqcap \forall hasAge.MiddleAge \sqcap \exists hasCourtyard$
- Resource 2: medieval palace with frescoed roofs. $S_2 \equiv Palace \sqcap \forall hasAge.MiddleAge \sqcap \forall hasRoof.FrescoedRoof$

Compatibility check allows to determine that S_1 is not compatible with R while S_2 is. Furthermore, $solveCAP(S_2, R)$ returns $H = \exists hasCourtyard$, thus pointing out the requirement that is not explicitly satisfied. On the contrary, using standard Subsumption check (like in works mentioned in Section 2) the requester would obtain just a *false* outcome, so she would not be aware that S_2 matches her request by a significant extent (it is indeed a medieval palace).

Implementation. In order to perform the above inferences via proper queries, both ABox and TBox have to be modeled in the OODB. As shown in Figure 1, managed data are structured in three different layers, each one providing a specific view of the KB.

1. Physical Layer. It refers to the OWL KB, further translated to a proper db4o database file. In current implementation, the database is built upon an XML file which describes an $\mathcal{ALN}$ KB using a simplified syntax w.r.t. OWL-DL.

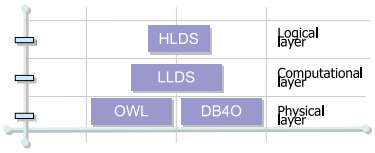

Fig. 1. Data organization layers

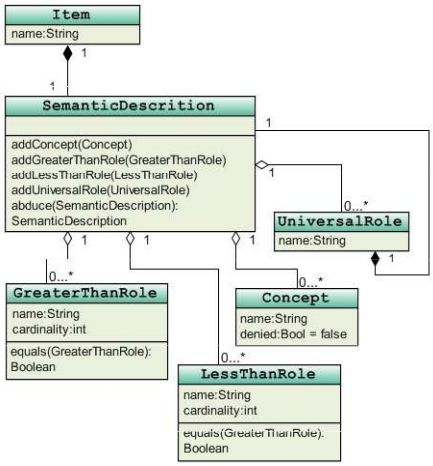

Fig. 2. HLDSs UML class diagram

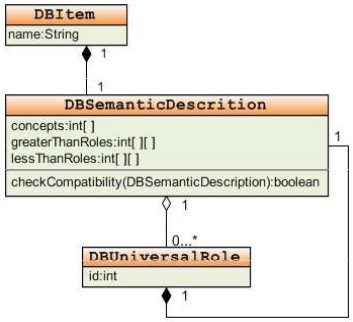

Fig. 3. LLDSs UML class diagram

In a preprocessing step, OWL files of thr KB are translated to this intermediate XML format.

2. High Level Data Structures. They correspond to the logical layer of the stack. Component classes (see Figure 2) are:

- *Item*: each KB individual is an instance of this class. The individual name is reported in the `name` attribute.
- *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. Among methods exposed by this class, the `abduce (SemanticDescription, request)` implements the abduction algorithm (see later on).
- *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 $\neg A_i$.
- *GreaterThanRole* (respectively *LessThanRole*): model number restrictions belonging to $C_{\geq}$ and $C_{\leq}$ sets. Role name and cardinality are stored in the homonym variables.

– *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.

3. Low Level Data Structures. In order to make more compact previous information and to optimize them for the inference algorithms, the HLDSs are translated into corresponding LLDSs, being manipulated by the DBMS. The parsing of an HLDS to produce a LLDS involves: (i) the translation of Java Collections into arrays; (ii) the parsing of role and concept strings into integer data (this *lexical normalization* [4] 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 through a specific index. It is used to get the concept (or role) corresponding to a given integer value (*i.e.*, 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 string-integer translations. Figure 3 shows the UML diagram of LLDSs. In what follows each class is detailed.

– *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.
– *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:
  • *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.
  • *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 further value associated to each restriction.
– *DBUniversalRole*: it maps the correspondent HLDS *UniversalRole*; the integer `id` attribute replaces the `name` string following the correspondence maps.

Concept Abduction Before running the abduction algorithm an early compatibility check has to be made aiming at excluding incompatible resources. Note that the *Transparent Activation* [13] in db40 allows to load in 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 by means of the following *native query* [13]:

```
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 (which implements Algorithm 1). The query returns a set of compatible resource instances in LLDS form, to be translated in the corresponding HLDSs. Noteworthily, for each atomic concept in the request, condition in line 2 of Algorithm 1 simply corresponds to searching for the negated integer in the concept array of the resource. Similarly, restrictions on the same property in line 6, 11, 16 in both request and resource are detected through elementary integer comparisons.

Abduction is executed by the `abduce()` method of the *SemanticDescription* class. When applied to a *SemanticDescription* object, it accepts in input the request description and returns not covered request features. They are modeled in a new *SemanticDescription* instance called `uncovered`. In the first loop of Algorithm 2 the following instruction is executed:

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

Notice that the `contains()` method of the Java Collection interface uses the `equals()` one 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.

4 Experimental Evaluation

Experiments covered all steps of system usage, namely Knowledge Base creation, retrieval of compatible supplies and abduction. Running time and memory usage were measured by means of profiling tools of NetBeans 5.5[4] development environment. All tests were performed on a Toshiba SA50-432 notebook PC, equipped with 1.5 GHz Intel Centrino CPU, 512 MB of RAM and Microsoft Windows XP operating system. Three KBs of different size and complexity were used for the tests: their features are summarized in Table 1.

[4] NetBeans IDE, http://www.netbeans.org/

Object Database creation. In addition to the three realistic KBs, 48 ones were randomly generated by varying several parameters as reported in Table 2.

Domain	Toys	Clothing	Electronics
XML input size (KB)	4	81	1823
♯ of concepts	19	86	280
♯ of roles	8	18	69
♯ of instances	7	35	42
avg ♯ of concept names per CNF expression	2.8	4.1	6.6
avg ♯ of number restrictions per CNF exp.	1.2	4.8	1.7
avg ♯ of universal quantifications per CNF exp.	1.4	6.0	2.1
maximum quantification nesting (QN)	2	2	12

Table 1. Summary of realistic Knowledge Bases used for tests

Parameter	Values
♯ of instances	10, 100, 1000
♯ of atomic concepts per CNF expression	2, 8
♯ of number restrictions per CNF expression	2, 6
♯ of universal quantifications per CNF expression	2, 6
maximum quantification nesting (QN)	4, 12

Table 2. Selected parameters for random Knowledge Base generation

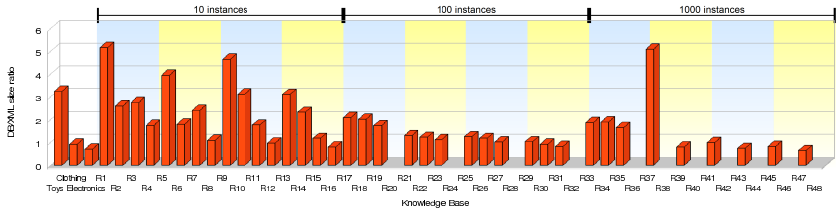

Fig. 4. Ratio between DB output and XML input size

Evaluated performance metrics include: output DB size (both absolute value and ratio over XML input size), processing time, throughput (amount of input data processed per time unit) and main memory peak during DB creation. Time and memory results are the average of 5 consecutive runs of the test suite. We must point out that some test cases were not completed due to an out of memory error raised by the DOM (Document Object Model) XML parsing library. In particular, this happened for random KBs having simultaneously: ≥ 100 instances; 6 universal quantifications for each level of a CNF expression; maximum QN set to 12. That corresponds to a worst case of $6^{12} = 2,176,782,336$ universal quantifications for each instance, which is far above typical cases.

Figure 4 shows the size ratio between XML input and DB output. Random KBs are grouped by number of instances, then in 4-tuples characterized by the same average of atomic concepts and number restrictions. Chart inspection and correlation values evidence that the ratio mainly depends on two KB parameters, namely the average number of universal quantifications and the quantification nesting. For simpler KBs the ratio is quite high, whereas for more complex KBs the object database has a proportionally lower memory footprint: this can be deemed as an effect of lexical normalization.

Figure 5 illustrates results for DB creation time. Time strongly depends on input KB size; average times for random KBs with 10, 100 and 1000 instances

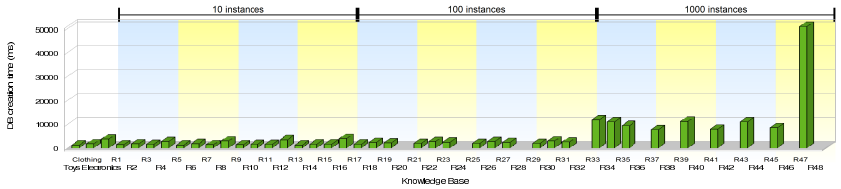

Fig. 5. DB creation time

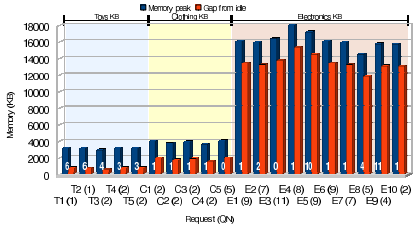

Fig. 6. Main memory usage peak for retrieval of compatible instances

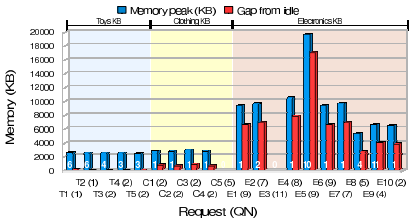

Fig. 7. Main memory usage peak for abduction

are 1.94, 2.33 and 14.44 s respectively, hence denoting a less-than-linear growth. Number of universal quantifications and QN are the structural properties that affect processing time the most. Throughput is highly correlated with input size (*i.e.*, larger Knowledge Bases are processed relatively faster), varying from few kB/s for smaller Knowledge Bases to hundreds of kB/s for larger ones. Finally, the main memory peak during processing depends fundamentally on QN and the number of roles in concept descriptions. The maximum value is 35 MB. Collected results, however, are not able to provide a complete picture of memory usage, due to the above issue with the DOM parser. A reimplementation of XML parsing module is underway using a more efficient SAX (Simple API for XML) library, that will allegedly shed more light over factors influencing memory cost.

Reasoning services. Retrieval of compatible instances and concept abduction were tested on the three real KBs. For Toys and Clothing KBs, having a maximum QN of 2, a set of 5 requests each was prepared. For the Electronics 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 the request expression and those 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 6 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 seems to be dependent on the size and complexity of the KB, regardless of both the QN of the request and the number of retrieved instances;

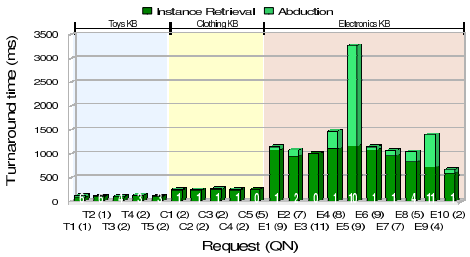

Fig. 8. Turnaround time

this is especially clear for the Electronics KB. Figure also shows that memory usage in idle state –*i.e.*, after DB creation and before a request is submitted– exhibits little variation among the three Knowledge Bases (being 2364 kB, 2035 kB and 2664 kB respectively), which suggests that the adoption of the object database does not introduce noticeable overhead.

Figure 7 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 value of 6.1 MB and the worst-case value of 19.6 MB are congruent with memory amounts of typical current mobile devices. The dependency of memory usage upon size and complexity of the Knowledge Base is even more manifest here, since the memory allocation gap from the idle state has different orders of magnitude in the three request groups (tens, hundreds and thousands of kB respectively).

Figure 8 reports results for aggregate turnaround time, also showing its two component tasks. It shows that time mostly depends on the complexity of the given KB. Overall performance can be deemed satisfactory, since all but one query on the largest KB require less than 1.5s, and time is much lower for smaller KBs. The duration of compatible retrieval task shows a mild correlation with the QN of the request, whereas no significant relation can be determined with the number of retrieved instances. This is likely due to the OODBMS approach, which translates instance retrieval and compatibility check into a set of queries on the database. On the other hand, the duration of the abduction process is highly dependent both on the complexity of the KB and on the number of abduction tests that are executed w.r.t. a given request. This may be explained by the fact that the algorithm acts upon in-memory data structures with no involvement of the DB. This could also justify the strong correlation between time and memory usage at abduction stage, which can be noticed comparing Figure 7 and the abduction time bars in Figure 8.

5 Conclusion

We proposed algorithms and a lightweight implementation of basic inference services for mobile devices. Proper reasoning tasks are performed issuing structured queries over an object oriented database modeling a given KB. The intrinsic

structure of an object oriented model of a DB lets relationships emerge among domain entities built as structured objects. Hence, we used it for information elicitation starting from what is explicit in the model instance. Proposed algorithms and approach have been tested on handheld devices using a Java-based open source OODBMS obtaining an early verification of their effectiveness. Future work will aim at implementing contraction algorithm whereas further optimizations are in progress to improve system performances.

References

1. Chen, H., Joshi, A., Finin, T.: Dynamic Service Discovery for Mobile Computing: Intelligent Agents MeetJini in the Aether. Cluster Computing **4**(4) (2001) 343–354
2. Baader, F., Horrocks, I., Sattler, U.: Description Logics as Ontology Languages for the Semantic Web. Festschrift in honor of Jorg Siekmann, Lecture Notes in Artificial Intelligence. Springer-Verlag (2003)
3. Baader, F., Hollunder, B., Nebel, B., Profitlich, H., Franconi, E.: Am empirical analysis of optimization techniques for terminological representation systems. Applied Intelligence **4**(2) (1994) 109–132
4. Horrocks, I., Patel-Schneider, P.: Optimizing description logic subsumption. Journal of Logic and Computation **9**(3) (1999) 267–293
5. Sinner, A., Kleemann, T.: KRHyper - In Your Pocket. In: Proc. of 20th International Conference on Automated Deduction (CADE-20), Tallinn, Estonia (July 2005) 452–457
6. Kleemann, T., Sinner, A.: User Profiles and Matchmaking on Mobile Phones. In Bartenstein, O., ed.: Proc. of 16th International Conference on Applications of Declarative Programming and Knowledge Management INAP2005, Fukuoka. (2005)
7. Colucci, S., Di Noia, T., Pinto, A., Ragone, A., Ruta, M., Tinelli, E.: A non-monotonic approach to semantic matchmaking and request refinement in e-marketplaces. International Journal of Electronic Commerce **12**(2) (2007) 127–154
8. Müller, F., Hanselmann, M., Liebig, T., Noppens, O.: A Tableaux-bades Mobile DL Reasoner - An Experience Report. In: Proceedings of the 2006 International Workshop on Description Logics (DL 06), Lake District, UK (May 2006)
9. Ruta, M., Di Noia, T., Di Sciascio, E., Piscitelli, G., Scioscia, F.: Semantic-based mobile registry for dynamic RFID-based logistics support. In: 10th International Conference on Electronic Commerce, ICEC 08, ACM Press (2008) ISBN 978-1-60558-075-3.
10. Baader, F., Calvanese, D., Mc Guinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook. Cambridge University Press (2002)
11. Donini, F., Lenzerini, M., Nardi, D., Schaerf, A.: Reasoning in Description Logics. In Brewka, G., ed.: Principles of Knowledge Representation: Studies in Logic, Language and Information. CSLI Publications (1996) 191–236
12. Di Noia, T., Di Sciascio, E., Donini, F.: Semantic matchmaking as non-monotonic reasoning: A description logic approach. Journal of Artificial Intelligence Research (JAIR) **29** (2007) 269–307
13. S. Edlich and, H.H., Hörning, R., Paterson, J.: The Definitive Guide to db4o. Apress edn. (June 2006)