Mini-ME: the Mini Matchmaking Engine

M. Ruta, F. Scioscia, E. Di Sciascio, F. Gramegna, and G. Loseto

Politecnico di Bari, via E. Orabona 4, I-70125 Bari, Italy
E-mail: m.ruta@poliba.it, f.scioscia@poliba.it, disciascio@poliba.it,
gramegna@deemail.poliba.it, loseto@deemail.poliba.it

Abstract. The Semantic Web of Things (SWoT) is a novel paradigm, blending the Semantic Web and the Internet of Things visions. Due to architectural and performance issues, it is currently impractical to use available reasoners for processing semantic-based information and perform resource discovery in pervasive computing scenarios. This paper presents a prototypical mobile reasoner for the SWoT, supporting Semantic Web technologies and implementing both standard (subsumption, satisfiability, classification) and non-standard (abduction, contraction) inference tasks for moderately expressive knowledge bases. Architectural and functional features are described and an experimental performance evaluation is provided both on a PC testbed (w.r.t. other popular Semantic Web reasoners) and on a smartphone.

1 Introduction

The Semantic Web of Things (SWoT) is an emerging paradigm in Information and Communication Technology, joining the Semantic Web and the Internet of Things. The Semantic Web initiative [5] envisions software agents to share, reuse and combine data available in the World Wide Web, by means of machine-understandable annotation languages such as RDF[1] and OWL[2], grounded on Description Logics (DLs) formalisms. The Internet of Things vision [11] promotes pervasive computing on a global scale, aiming to give intelligence to ordinary objects and physical locations by means of a large number of heterogeneous micro-devices, each conveying a small amount of information. Consequently, the goal of the SWoT is to embed semantically rich and easily accessible metadata into the physical world, by enabling storage and retrieval of annotations from tiny smart objects. Such a vision requires an increased autonomy and efficiency of knowledge-based systems for what concerns information memorization, management, dissemination and discovery. Particularly, reasoning and query answering aimed to resource discovery is critical in mobile computing platforms (*e.g.*, smartphones, tablets) which –albeit increasingly effective and powerful– are still affected by hardware/software limitations. They have to be taken into account

[1] Resource Description Framework, W3C Recommendation 10 February 2004, http://www.w3.org/TR/rdf-primer/

[2] OWL 2 Web Ontology Language, W3C Recommendation 27 October 2009, http://www.w3.org/TR/owl-overview/

when designing systems and applications: particularly, to use more expressive languages increases the computational complexity of inferences and significant architectural and performance issues affect porting current OWL-based reasoners, designed for the Semantic Web, to handheld devices. This paper presents *Mini-ME (the Mini Matchmaking Engine)*, a prototypical mobile reasoner for moderately expressive DLs, created to support *semantic-based matchmaking* [6], [16]. It complies with standard Semantic Web technologies through the OWL API [9] and implements both standard reasoning tasks for Knowledge Base (KB) management (subsumption, classification, satisfiability) and non-standard inference services for semantic-based resource discovery and ranking (abduction and contraction [6]). Mini-ME is developed in Java, adopting Android as target computing platform.

The remaining of the paper is organized as follows. Section 2 reports on related work, providing perspective and motivation for the proposal. Mini-ME is presented in Section 3, where details are given about reasoning algorithms, software architecture, data structures and supported logic languages. Section 4 relates to performance evaluation on the venue reference datasets[3] and a comparison with other popular Semantic Web reasoners is proposed. Finally conclusion and future work in Section 5 close the paper.

2 Related Work

When processing semantic-based information to infer novel and implicit knowledge, careful optimization is needed to achieve acceptable reasoning performance for adequately expressive languages [3, 10]. This is specifically true in case of logic-based matchmaking for mobile computing, which is characterized by severe resource limitations (not only affecting processing, memory and storage, but also energy consumption). Most mobile engines currently provide only rule processing for entailments materialization in a KB [14, 27, 12, 18], so basically, available features are not suitable to support applications requiring non-standard inference tasks and extensive reasoning over ontologies [18]. More expressive languages could be used by adapting tableaux algorithms –usually featuring reasoners running on PCs– to mobile computing platforms, but an efficient implementation of reasoning services is still an open problem. Several techniques [10] allow to increase expressiveness or decrease running time at the expense of main memory usage, which is the most constrained resource in mobile systems. *Pocket KRHyper* [24] was the first reasoning engine specifically designed for mobile devices. It supported the $\mathcal{ALCHIR}+$ DL and was built as a Java ME (Micro Edition) library. Pocket KRHyper was exploited in a DL-based matchmaking framework between user profiles and descriptions of mobile resources/services [13]. However, its limitation in size and complexity of managed logic expressions was very heavy due to frequent "out of memory" errors. To overcome those constraints, tableaux optimizations to reduce memory consumption were introduced in [26] and implemented in *mTableau*, a modified version of Java SE *Pellet* reasoner

[3] http://www.cs.ox.ac.uk/isg/conferences/ORE2012/

[25]. Comparative performance tests were performed on a PC, showing faster turnaround times than both unmodified Pellet and *Racer* [8] reasoner. Nevertheless, the Java SE technology is not expressly tailored to the current generation of handheld devices. In fact, other relevant reasoners, such as *FaCT++* [28] and *HermiT* [23], cannot run on common mobile platforms. Porting would require a significant re-write or re-design effort, since they rely on Java class libraries incompatible with mosto widespread mobile OS (*e.g.*, Android). Moreover, the above systems only support standard inference services such as satisfiability and subsumption, which provide only binary "yes/no" answers. Consequently, they can only distinguish among *full* (*subsume*), *potential* (*intersection-satisfiable*) and *partial* (*disjoint*) match types (adopting the terminology in [6] and [16], respectively). Non-standard inferences, as Concept Abduction and Concept Contraction, are needed to enable a more fine-grained semantic ranking as well as explanations of outcomes [6]. In latest years, a different approach to implement reasoning tools arose. It was based on simplifying both the underlying logic languages and admitted KB axioms, so that structural algorithms could be adopted, but maintaining expressiveness enough for broad application areas. In [1], the basic $\mathcal{EL}$ DL was extended to $\mathcal{EL}^{++}$, a language deemed suitable for various applications, characterized by very large ontologies with moderate expressiveness. A structural classification algorithm was also devised, which allowed high-performance $\mathcal{EL}^{++}$ ontology classifiers such as CEL [4] and Snorocket [15]. OWL 2 profiles definition complies with this perspective, focusing on language subsets of practical interest for important application areas rather than on fragments with significant theoretical properties. In a parallel effort motivated by similar principles, in [22] an early approach was proposed to adapt non-standard logic-based inferences to pervasive computing contexts. By limiting expressiveness to $\mathcal{AL}$ language, acyclic, structural algorithms were adopted reducing standard (*e.g.*, subsumption) and non-standard (*e.g.*, abduction and contraction) inference tasks to set-based operations [7]. KB management and reasoning were then executed through a data storage layer, based on a mobile RDBMS (Relational DBMS). Such an approach was further investigated in [20] and [19], by increasing the expressiveness to $\mathcal{ALN}$ DL and allowing larger ontologies and more complex descriptions, through the adoption of both mobile OODBMS (Object-Oriented DBMS) and performance-optimized data structures. Finally, in [21] expressiveness was extended to $\mathcal{ALN}$(D) DL with fuzzy operators. The above tools were designed to run on Java ME PDAs and were adopted in several case studies employing semantic matchmaking over moderately expressive KBs. The reasoning engine presented here recalls lessons learned in those previous efforts, and aims to provide a standards-compliant implementation of most common inferences (both standard and non-standard) for widespread mobile platforms.

3 System Description

The architecture of the proposed reasoning engine is sketched as UML diagram in Figure 1. Components are outlined hereafter:

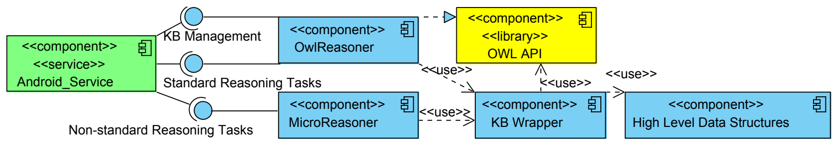

<<component>> <<service>> Android_Service
KB Management
Standard Reasoning Tasks
Non-standard Reasoning Tasks
<<component>> OwlReasoner
<<component>> MicroReasoner
<<component>> <<library>> OWL API
<<component>> KB Wrapper
<<component>> High Level Data Structures
<<use>>
<<use>>
<<use>>
<<use>>

Fig. 1: Component UML diagram

- **Android Service**: implements a service (*i.e.*, a background daemon) any Android application can invoke to use the engine;
- **OwlReasoner**: OWL API [9] implementation exposing fundamental KB operations (load, parse) and standard reasoning tasks (subsumption, classification, satisfiability); it is endorsed by the OWL API open source library;
- **MicroReasoner**: interface for non-standard reasoning tasks (concept abduction, contraction);
- **KB Wrapper**: implements KB management functions (creation of internal data structures, normalization, unfolding) and basic reasoning tasks on ontologies (classification and coherence check);
- **High Level Data Structures**: in-memory data structures for concept manipulation and reasoning; they refer to reasoning tasks on concept expressions (concept satisfiability, subsumption, abduction, contraction).

Mini-ME was developed using Android SDK Tools[4], Revision 12, corresponding to Android Platform version 2.1 (API level 7), therefore it is compatible with all devices running Android 2.1 or later. Mini-ME can be used either through the *Android Service* by Android applications, or as a library by calling public methods of the *OwlReasoner* and *MicroReasoner* components directly. In the latter form, it runs unmodified on Java Standard Edition runtime environment, version 6 or later. The system supports OWL 2 ontology language, in all syntaxes accepted by the OWL API parser. Supported logic constructors are detailed in Section 3.1. Implementation details for both standard and non-standard reasoning services are given in Section 3.2. Data structures for internal representation and manipulation of concept expressions are outlined in Section 3.3.

3.1 Supported Language

In DL-based reasoners, an ontology $\mathcal{T}$ (a.k.a. Terminological Box or TBox) is composed by a set of axioms in the form: $A \sqsubseteq D$ or $A \equiv D$ where A and D are concept expressions. Particularly, a *simple-TBox* is an acyclic TBox such that: (i) A is always an atomic concept; (ii) if A appears in the left hand side (lhs) of a concept equivalence axiom, then it cannot appear also in the lhs of any concept inclusion axiom. Mini-ME supports the $\mathcal{ALN}$ (Attributive Language with unqualified Number restrictions) DL, which has polynomial computational complexity for standard and non-standard inferences in simple-TBoxes, whose depth of concept taxonomy is bounded by the logarithm of the number of axioms in it (see [7] for further explanation). Actually, such DL fragment has been

[4] http://developer.android.com/sdk/tools-notes.html

Table 1: Syntax and semantics of $\mathcal{ALN}$ constructs and simple-TBoxes

Name	Syntax	Semantics
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Atomic negation	$\neg A$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$
Universal quantification	$\forall R.C$	$\{d_1 \mid \forall d_2 : (d_1, d_2) \in R^{\mathcal{I}} \rightarrow d_2 \in C^{\mathcal{I}}\}$
Number restriction	$\geq nR$	$\{d_1 \mid \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \geq n\}$
	$\leq nR$	$\{d_1 \mid \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \leq n\}$
Inclusion	$A \sqsubseteq D$	$A^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Equivalence	$A \equiv D$	$A^{\mathcal{I}} = D^{\mathcal{I}}$

selected for the first release of Mini-ME as it grants low complexity and memory efficiency of non-standard inference algorithms for semantic matchmaking. $\mathcal{ALN}$ DL constructs are summarized in Table 1.

3.2 Reasoning Services

Mini-ME exploits structural algorithms for standard and non-standard reasoning and then, when a knowledge base is loaded, it has to be preprocessed performing *unfolding* and *Conjunctive Normal Form* (CNF) *normalization*. Particularly, given a TBox $\mathcal{T}$ and a concept C, the **unfolding** procedure recursively expands references to axioms in $\mathcal{T}$ within the concept expression itself. In this way, $\mathcal{T}$ is not needed any more when executing subsequent inferences. **Normalization** transforms the unfolded concept expression in CNF by applying a set of predefined substitutions. Any concept expression C can be reduced in CNF as: $C \equiv C_{CN} \sqcap C_{LT} \sqcap C_{GT} \sqcap C_{\forall}$, where C_{CN} is the conjunction of (possibly negated) atomic concept names, C_{LT} (respectively C_{GT}) is the conjunction of $\leq$ (resp. $\geq$) number restrictions (no more than one per role), and $C_{\forall}$ is the conjunction of universal quantifiers (no more than one per role; fillers are recursively in CNF). Normalization preserves semantic equivalence w.r.t. models induced by the TBox; furthermore, CNF is unique (up to commutativity of conjunction operator) [7]. The normal form of an unsatisfiable concept is simply $\bot$. The following standard reasoning services on (unfolded and normalized) concept expressions are currently supported:
- **Concept Satisfiability** (a.k.a. consistency). Due to CNF properties, satisfiability check is trivially performed during normalization.
- **Subsumption test**. The classic structural subsumption algorithm is exploited, reducing the procedure to a set containment test [2].

In Mini-ME, two non-standard inference services were also implemented, allowing to (i) provide explanation of outcomes beyond the trivial "yes/no" answer of satisfiability and subsumption tests and (ii) enable a logic-based relevance ranking of a set of available resources w.r.t. a specific query [19]:
- **Concept Contraction**: given a request D and a supplied resource S, if they are not compatible with each other, Contraction determines which part of D is conflicting with S. If one retracts conflicting requirements in D, G (for *Give up*), a concept K (for *Keep*) is obtained, representing a contracted version of

the original request, such that $K \sqcap S$ is satisfiable w.r.t. $\mathcal{T}$. The solution G to Contraction represents "why" $D \sqcap S$ are not compatible.
- **Concept Abduction**: whenever D and S are compatible, but S does not imply D, Abduction allows to determine what should be hypothesized in S in order to completely satisfy D. The solution H (for *Hypothesis*) to Abduction represents "why" the subsumption relation $\mathcal{T} \models S \sqsubseteq D$ does not hold. H can be interpreted as *what is requested in* D *and not specified in* S.

In order to use Mini-ME in more general knowledge-based applications, the following reasoning services over ontologies were also implemented:
- **Ontology Satisfiability**: since Mini-ME does not currently process the ABox, it performs an ontology *coherence* check rather than satisfiability check (difference is discussed *e.g.*, in [17]). During ontology parsing, the *KB Wrapper* module creates a hash table to store all concepts in the TBox $\mathcal{T}$. Since CNF normalization allows to identify unsatisfiable concepts, it is sufficient to normalize every table item to locate unsatisfiability in the ontology.
- **Classification**: ontology classification computes the overall concept taxonomy induced by the subsumption relation, from $\top$ to $\bot$ concept. In order to reduce the subsumption tests, the following optimizations introduced in [3] were implemented: *enhanced traversal top search*, *enhanced traversal bottom search*, exploitation of *told subsumers*. The reader is referred to [3] for further details.

3.3 Data Structures

The UML diagram in Figure 2 depicts classes in the *High Level Data Structures* package (mentioned before) and their relationships. Standard Java Collection Framework classes are used as low-level data structures:
- **Item**: each concept in the ontology is an instance of this class. Attributes are the name and the corresponding concept expression. When parsing an ontology, the *KB Wrapper* component builds a Java *HashMap* object containing all concepts in the TBox as *String*-*Item* pairs. Each concept is unfolded, normalized and stored in the HashMap with its name as key and *Item* instance as value.
- **SemanticDescription**: models a concept expression in CNF as aggregation of $C_{CN}, C_{GT}, C_{LT}, C_{\forall}$ components, each one stored in a different Java *ArrayList*. Methods implement inference services: `abduce` returns the hypothesis H expression; `contract` returns a two-element array with G and K expressions; `checkCompatibility` checks consistency of the conjunction between the object *SemanticDescription* and the one acting as input parameter; similarly, `isSubsumed` performs subsumption test with the input *SemanticDescription*.
- **Concept**: models an atomic concept A_i in C_{CN}; `name` contains the concept name, while `denied`, if set to *true*, allows to express $\neg A_i$.
- **GreaterThanRole** (respectively **LessThanRole**): models number restrictions in C_{GT} and C_{LT}. Role name and cardinality are stored in the homonym variables.
- **UniversalRole**: a universal restriction $\forall R.D$ belonging to $C_{\forall}$; R is stored in `name`, while D is a *SemanticDescription* instance.

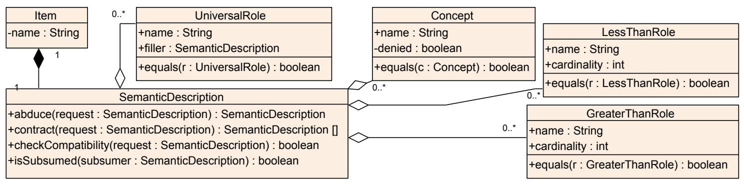

Item
-name : String
1
1
0..*
UniversalRole
+name : String
+filler : SemanticDescription
+equals(r : UniversalRole) : boolean
Concept
+name : String
-denied : boolean
+equals(c : Concept) : boolean
0..*
LessThanRole
+name : String
+cardinality : int
+equals(r : LessThanRole) : boolean
0..*
SemanticDescription
+abduce(request : SemanticDescription) : SemanticDescription
+contract(request : SemanticDescription) : SemanticDescription []
+checkCompatibility(request : SemanticDescription) : boolean
+isSubsumed(subsumer : SemanticDescription) : boolean
0..*
GreaterThanRole
+name : String
+cardinality : int
+equals(r : GreaterThanRole) : boolean

Fig. 2: Class diagram of *High Level Data Structures* package

In the last classes, the `equals` method, inherited from *java.lang.Object*, has been overridden in order to properly implement logic-based comparison.

4 Experimental Evaluation

Performance evaluation was carried out for classification, class satisfiability and ontology satisfiability, including both a comparison with other popular Semantic Web reasoners on a PC testbed[5] and results obtained on an Android smartphone[6]. The reference dataset is composed of 214 OWL ontologies with different complexity, expressiveness and syntax. Full results are reported on the project home page[7], while main highlights are summarized hereafter. Mini-ME was compared on PC with FaCT++[8], HermiT[9] and Pellet[10]. All reasoners were used via the OWL API [9]. For each reasoning task, two tests were performed: (i) correctness of results and turnaround time; (ii) memory usage peak. For turnaround time, each test was repeated four times and the average of the last three runs was taken. For memory tests, the final result was the average of three runs. Performance evaluation for non-standard inferences is not provided here.

4.1 PC Tests

Classification. The input of this task was the overall ontology dataset. For each test, one of the following possible outcomes was recorded: (i) *Correct*, the computed taxonomy corresponds with the reference classification –if it is included into the dataset– or results of all the reasoners are the same; in this case the total time taken to load and classify the ontology is also reported; (ii) *Parsing Error*, the ontology cannot be parsed by the OWL API due to syntax errors; (iii) *Failure*, the classification task fails because the ontology contains unsupported logic language constructors; (iv) *Out of Memory*, the reasoner generates an exception

[5] Intel Core i7 CPU 860 at 2.80 GHz (4 cores/8 threads), 8 GB DDR3-SDRAM (1333 MHz) memory, 1 TB SATA (7200 RPM) hard disk, 64-bit Microsoft Windows 7 Professional and 64-bit Java 7 SE Runtime Environment (build 1.7.0_03-b05).

[6] Samsung i9000 Galaxy S with ARM Cortex A8 CPU at 1 GHz, 512 MB RAM, 8 GB internal storage memory, and Android version 2.3.3.

[7] Mini-ME Home Page, http://sisinflab.poliba.it/swottools/minime/

[8] FaCT++, version 1.5.3 with OWL API 3.2, http://owl.man.ac.uk/factplusplus/

[9] HermiT OWL Reasoner, version 1.3.6, http://hermit-reasoner.com/

[10] Pellet, version 1.3, http://clarkparsia.com/pellet/

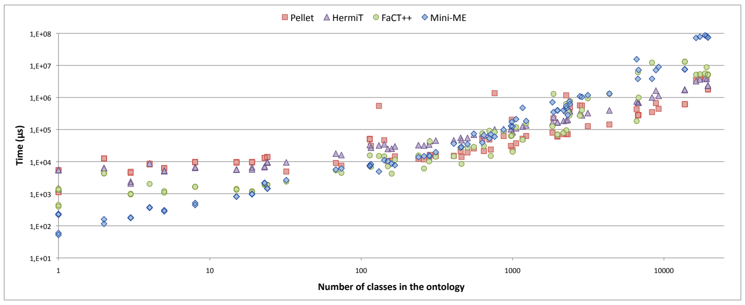


Fig. 3: Classification test on PC

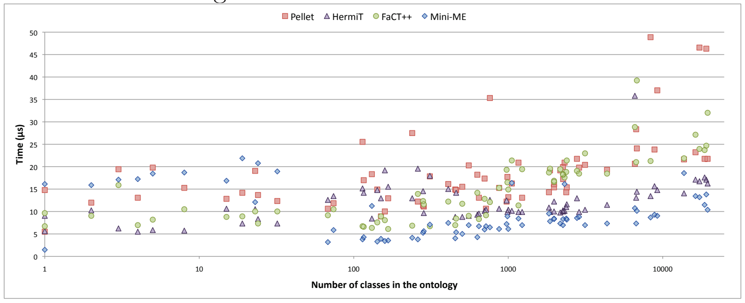


Fig. 4: Class Satisfiability on PC

due to memory constraints; (v) *Timeout*, the task did not complete within the timeout threshold (set to 60 minutes). Mini-ME correctly classified 83 of 214 ontologies; 71 were discarded due to parsing errors, 58 presented unsupported language constructors, the timeout was reached in 2 cases. Pellet classified correctly 124 ontologies, HermiT 127, FaCT++ 118. The lower "score" of Mini-ME is due to the presence of General Concept Inclusions, cyclic TBoxes or unsupported logic constructors, since parsing errors occur in the OWL API library and are therefore common to all reasoners. Figure 3 compares the classification times of each reference reasoner w.r.t. the number of classes in every ontology. Pellet, HermiT and FaCT++ present a similar trend (with FaCT++ slightly faster than the other engines), while Mini-ME is very competitive for small-medium ontologies (up to 1200 classes) but less for large ones. This can be considered as an effect of the Mini-ME design, which is optimized to manage elementary TBoxes.

Class satisfiability. The reference test dataset consists of 107 ontologies and, for each of them, one or more classes to check. However, we tested only the 69 ontologies that Mini-ME correctly classified in the previous proof. Figure 4 shows that performances are basically similar, with times differing only for few microseconds and no reasoner consistently faster or slower. Moreover, the chart suggests no correlation between the time and the number of classes in the ontology.

Ontology satisfiability. Figure 5 is similar to Figure 3, because this test implies loading, classifying and checking consistency of all concepts in the ontology; the

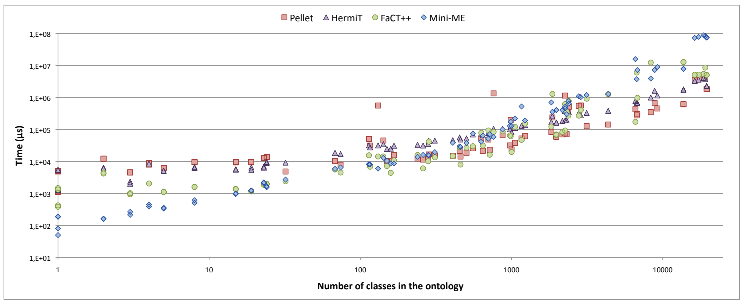
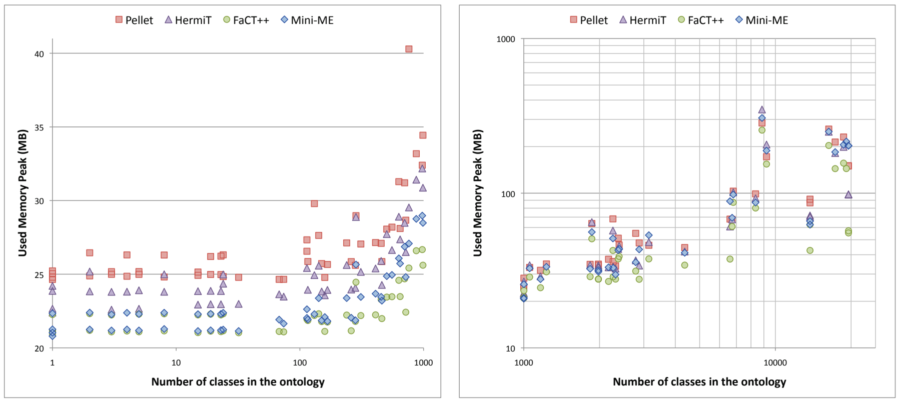


Fig. 5: Ontology Satisfiability on PC



(a) Small ontologies (b) Large ontologies

Fig. 6: Memory usage test on PC

first two steps require the larger part of the time. Results of all reasoners are the same, except for ontologies with IDs 199, 200, 202, 203. In contrast to Pellet, HermiT and FaCT++, Mini-ME checks ontology coherence regardless of the ABox. The above ontologies include an unsatisfiable class (`GO_0075043`) with no instances, therefore the ontology is reported as incoherent by Mini-ME but as satisfiable by the other reasoners.

Memory Usage. Figure 6 reports on memory usage peak during classification, which was verified as the most memory-intensive task. For small ontologies, used memory is roughly similar for all reasoners; Mini-ME provides good results, with lower memory usage than Pellet and HermiT and on par with FaCT++. Also for large ontologies, Mini-ME results are comparable with the other reasoners, although FaCT++ has slightly better overall performance.

4.2 Mobile Tests

Results for mobile tests have been referred to the above outcomes for PC tests in order to put in evidence Mini-ME exhibits similar trends (so offering predictable memory and time consumption behaviors). Anyway, figures clearly evidence the performance gap, but they highlight the reasoner acceptably works also on mobile platforms. When out-of-memory errors did not occur, results computed by Mini-ME on the Android smartphone were in all cases the same as on the PC. 73 ontologies over 214 were correctly classified on the mobile device, 53 were discarded due to parsing errors, 56 had unsupported language constructors, 30 generated out-of-memory exceptions and 2 reached the timeout. Figure 7 shows the classification turnaround time –only for the correct outcomes– compared with the PC test results. Times are roughly an order of magnitude higher on the Android device. Absolute values for ontologies with 1000 classes or less are under 1 second, so they can be deemed as acceptable in mobile contexts. Furthermore, it can be noticed that the turnaround time increases linearly w.r.t. number of classes both on PC and on smartphone, thus confirming that Mini-ME has predictable behavior regardless of the reference platform. Similar considerations apply to class and ontology satisfiability tests (which were run for the 60 ontologies that were correctly classified): the turnaround time comparisons are reported in Figure 8 and Figure 9. Figure 10 reports on the memory allocation peak for each ontology during the classification task. Under 1000 classes, the required memory is roughly fixed in both cases. Instead, for bigger ontologies the used memory increases according to the total number of classes. Moreover, in every test memory usage on Android is significantly lower than on PC. This is due to the harder memory constraints on smartphones, imposing to have as much free memory as possible at any time. Consequently, Android Dalvik virtual machine performs more frequent and aggressive garbage collection w.r.t. Java SE virtual machine. This reduces memory usage, but on the other hand can be responsible for a significant portion of the PC-smartphone turnaround time gap that was found.

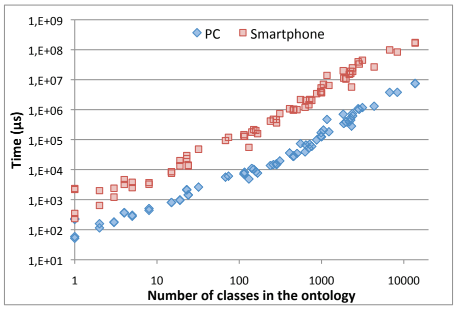


Fig. 7: Classification, PC vs mobile

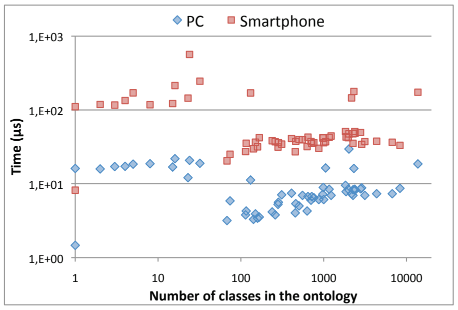


Fig. 8: Class Satisfiability, PC vs mobile

5 Conclusion and Future Work

The paper presented a prototypical reasoner devised for mobile computing. It supports Semantic Web technologies through the OWL API and implements

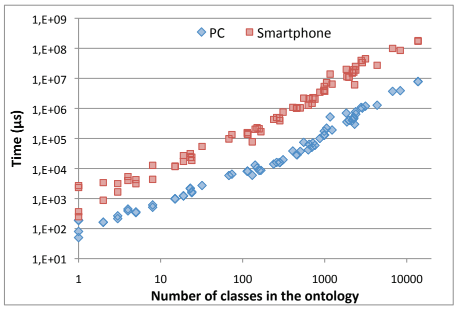


Fig. 9: Ont. Satisfiability, PC vs mobile

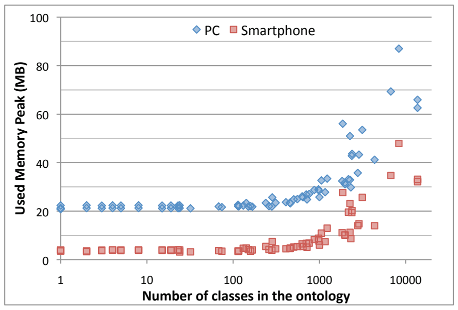


Fig. 10: Memory usage, PC vs mobile

both standard and non-standard reasoning tasks. Developed in Java, it targets the Android platform but also runs on Java SE. Early experiments were made both on PCs and smartphones and evidenced correctness of implementation and competitiveness with state-of-the-art reasoners in standard inferences, and acceptable performance on target mobile devices. Besides further performance optimization leveraging Android Dalvik peculiarities, future work includes: support for ABox management and OWLlink protocol[11]; implementation of further reasoning tasks; $\mathcal{EL}^{++}$extension of abduction and contraction algorithms.

References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the EL envelope. In: Int. Joint Conf. on Artificial Intelligence. vol. 19, p. 364. Lawrence Erlbaum Associates LTD (2005)
2. Baader, F., Calvanese, D., Mc Guinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook. Cambridge University Press (2002)
3. Baader, F., Hollunder, B., Nebel, B., Profitlich, H., Franconi, E.: An empirical analysis of optimization techniques for terminological representation systems. Applied Intelligence 4(2), 109–132 (1994)
4. Baader, F., Lutz, C., Suntisrivaraporn, B.: CEL – a polynomial-time reasoner for life science ontologies. Automated Reasoning pp. 287–291 (2006)
5. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic Web. Scientific American 284(5), 28–37 (2001)
6. 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. Int. Jour. of Electronic Commerce 12(2), 127–154 (2007)
7. Di Noia, T., Di Sciascio, E., Donini, F.: Semantic matchmaking as non-monotonic reasoning: A description logic approach. Jour. of Artificial Intelligence Research (JAIR) 29, 269–307 (2007)
8. Haarslev, V., Müller, R.: Racer system description. Automated Reasoning pp. 701–705 (2001)
9. Horridge, M., Bechhofer, S.: The OWL API: a Java API for working with OWL 2 ontologies. Proc. of OWL Experiences and Directions 2009 (2009)

[11] OWLlink Structural Specification, W3C Member Submission, http://www.w3.org/Submission/owllink-structural-specification/

10. Horrocks, I., Patel-Schneider, P.: Optimizing description logic subsumption. Jour. of Logic and Computation 9(3), 267–293 (1999)
11. ITU: Internet Reports 2005: The Internet of Things (November 2005)
12. Kim, T., Park, I., Hyun, S., Lee, D.: MiRE4OWL: Mobile Rule Engine for OWL. In: Computer Software and Applications Conf. Workshops (COMPSACW), 2010 IEEE 34th Annual. pp. 317–322. IEEE (2010)
13. Kleemann, T., Sinner, A.: User Profiles and Matchmaking on Mobile Phones. In: Bartenstein, O. (ed.) Proc. of 16th Int. Conf. on Applications of Declarative Programming and Knowledge Management INAP2005, Fukuoka (2005)
14. Koch, F.: 3APL-M platform for deliberative agents in mobile devices. In: Proc. of the fourth international joint conference on Autonomous agents and multiagent systems. p. 154. ACM (2005)
15. Lawley, M., Bousquet, C.: Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner. In: Proc. 6th Australasian Ontology Workshop (IAOA10). Conf.s in Research and Practice in Information Technology. vol. 122, pp. 45–49 (2010)
16. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. Int. Jour. of Electronic Commerce 8(4), 39–60 (2004)
17. Moguillansky, M., Wassermann, R., Falappa, M.: An argumentation machinery to reason over inconsistent ontologies. Advances in Artificial Intelligence–IBERAMIA 2010 pp. 100–109 (2010)
18. Motik, B., Horrocks, I., Kim, S.: Delta-Reasoner: a Semantic Web Reasoner for an Intelligent Mobile Platform. In: Twentyfirst Int. World Wide Web Conf. (WWW 2012). ACM (2012), to appear
19. Ruta, M., Di Sciascio, E., Scioscia, F.: Concept abduction and contraction in semantic-based P2P environments. Web Intelligence and Agent Systems 9(3), 179–207 (2011)
20. Ruta, M., Scioscia, F., Di Noia, T., Di Sciascio, E.: Reasoning in Pervasive Environments: an Implementation of Concept Abduction with Mobile OODBMS. In: 2009 IEEE/WIC/ACM Int. Conf. on Web Intelligence. pp. 145–148. IEEE (2009)
21. Ruta, M., Scioscia, F., Di Sciascio, E.: Mobile Semantic-based Matchmaking: a fuzzy DL approach. In: The Semantic Web: Research and Applications. Proceedings of 7th Extended Semantic Web Conference (ESWC 2010). Lecture Notes in Computer Science, vol. 6088, pp. 16–30. Springer (2010)
22. Ruta, M., Di Noia, T., Di Sciascio, E., Piscitelli, G., Scioscia, F.: A semantic-based mobile registry for dynamic RFID-based logistics support. In: ICEC '08: Proc. of the 10th Int. Conf. on Electronic commerce. pp. 1–9. ACM, New York, USA (2008)
23. Shearer, R., Motik, B., Horrocks, I.: Hermit: A highly-efficient owl reasoner. In: Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008). pp. 26–27 (2008)
24. Sinner, A., Kleemann, T.: KRHyper - In Your Pocket. In: Proc. of 20th Int. Conf. on Automated Deduction (CADE-20). pp. 452–457. Tallinn, Estonia (July 2005)
25. Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Web Semantics: science, services and agents on the World Wide Web 5(2), 51–53 (2007)
26. Steller, L., Krishnaswamy, S.: Pervasive Service Discovery: mTableaux Mobile Reasoning. In: Int. Conf. on Semantic Systems (I-Semantics). Graz, Austria (2008)
27. Tai, W., Keeney, J., O'Sullivan, D.: COROR: a composable rule-entailment owl reasoner for resource-constrained devices. Rule-Based Reasoning, Programming, and Applications pp. 212–226 (2011)
28. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. Automated Reasoning pp. 292–297 (2006)