# Android goes Semantic: DL Reasoners on Smartphones

Roberto Yus, Carlos Bobed, Guillermo Esteban, Fernando Bobillo, and Eduardo Mena

University of Zaragoza
Maria de Luna 1, Zaragoza, Spain
{ryus,cbobed,gesteban,fbobillo,emena}@unizar.es

**Abstract.** The massive spread of mobile computing in our daily lives has attracted a huge community of mobile apps developers. These developers can take advantage of the benefits of semantic technologies (such as knowledge sharing and reusing, knowledge decoupling, etc.) to enhance their applications. Moreover, the use of semantic reasoners would enable them to create more intelligent applications capable of inferring logical consequences from the knowledge considered. However, using semantic APIs and reasoners on current Android-based devices is not problem-free and, currently, there are no remarkable efforts to enable mobile devices with semantic reasoning capabilities.
In this paper, we analyze whether the most popular current available DL reasoners can be used on Android-based devices. We evaluate the efforts needed to port them to the Android platform, taking into account its limitations, and present some tests to show the performance of these reasoners on current smartphones/tablets.

## 1 Introduction

In the last few years, we have witnessed a massive spread of mobile computing in our daily lives. The progress and popularity of the different mobile devices (smartphones, tablets, etc.) has attracted a huge community of developers that are continually releasing new applications via the different app stores. For example, 136 millions of Android-based devices were sold in third quarter of 2012 (which represented 75% of the market)[1] and the Google Play[2] market contained almost 700,000 available applications in April 2013.

Due to the current device capabilities, we think that we could also start to enhance all these applications with semantic technologies [16]. Using these techniques, developers can enhance their applications based on the principles of knowledge sharing and reusing [3], making explicit domain assumptions and thus decoupling the knowledge from the application, etc. For example, the use

[1] IDC studies, http://www.idc.com/getdoc.jsp?containerId=prUS23771812, last accessed 18th April 2013
[2] http://play.google.com

of ontologies and semantic reasoners would enable them to create more intelligent applications capable of inferring logical consequences from the knowledge considered [6]. However, the use of semantic technologies on mobile apps has not (yet) spread due, in part, to the fact that there are currently no remarkable efforts to enable mobile devices with semantic reasoning capabilities. Moreover, having local reasoners on the devices of the users would enable developers and mobile apps to manage knowledge even when network disconnections make impossible to rely on third-party devices/computers to carry out the reasoning. While authors in [11] have implemented a mobile reasoner from scratch which supports the Description Logic (DL) $\mathcal{ALCN}$ (they state that "... current Semantic Web reasoners cannot be ported without a significant re-write effort."); we have adopted another approach and we have started a survey to analyze if this re-writing effort is worth enough and how we can handle semantics in mobile applications by adapting existing DL reasoners.

In this paper, we evaluate whether the most popular among the current DL reasoners can be used directly in mobile devices, the efforts needed if they do not, and their performance with five well-know ontologies once we were able to make them work. We have focused on Android-based devices due to their spread, their openness, and the fact that it has a native virtual machine that is really close to Java (Dalvik). The existence of this Java-like virtual machine is really appealing in order to have important APIs working on the mobile devices, and thus allows reusing a lot of already developed code in new applications.

## 2 Reasoning on Android

Most of current popular semantic reasoners are implemented using Java (e.g., Pellet and HermiT) and are usually used along with semantic APIs (e.g., OWL API and Jena). Android, which is a Linux-based operating system whose middleware, libraries, and APIs are written in C, supports Java code as it uses a Java-like virtual machine called Dalvik [9]. In fact, Dalvik runs "dex-code" (Dalvik Executable), and Java bytecodes can be converted to Dalvik-compatible .dex files to be executed on Android. However, Dalvik does not align to Java SE and so it does not support Java ME classes, AWT or Swing. Thus, running semantic APIs and reasoners on Android could require some rewriting efforts.

### 2.1 Running Semantic APIs on Android

In the following we explain how to use two useful semantic APIs on Android.

**OWL API** [5] is an ontology API to manage OWL 2 ontologies in Java applications and provides a high-level way to interact with DL reasoners. It can be considered as a de facto standard, as the most recent versions of the DL reasoners use OWL API to load and process the ontologies.

We considered the last available version of the OWL API 3.4.3[3] which could be converted to Dalvik without any modifications and so, imported into an

[3] http://owlapi.sourceforge.net

Android project directly. However, we tried the OWL API 2.2.0 (automatically imported by Pellet along with the OWL API 3.2.4) but it uses Java classes that are not supported by Dalvik. Nevertheless, as new developers are expected to use the OWL API 3 they would not find any problems when importing the library.

**Jena** is an ontology API to manage OWL ontologies and handle RDF data in Java applications, but support for OWL 2 is not available yet. Jena reasoners are based on answering queries over RDF graphs.

Although Jena cannot be directly imported into an Android project, there exist a project called Androjena[4] to port it to the Android platform. The last version of Androjena 0.5, which was used in our tests, contains all the original code of Jena 2.6.2.

### 2.2 Running Reasoners on Android

In the following we present how to run some popular reasoners on Android (ordered incrementally according to the effort needed).

**JFact** is a port of FaCT++ to Java. FaCT++ reasoner [15], successor of Fact reasoner, is implemented in C++ and supports full OWL 2 with reasoning based on a tableaux algorithm.

We used JFact 0.9.1[5], which does not import any external libraries (except for the OWLAPI), and its code can be converted directly to Dalvik. In this way, we can develop an Android app that uses this reasoner by simply importing the JFact 0.9.1 and OWLAPI 3.4.3 .jar files in our Android project.

**CB** [7] reasoner is implemented in OCaml and supports a fragment of OWL 2 (Horn-$\mathcal{SHIF}$). Reasoning is based on a consequence-based procedure.

Android does not support the OCaml language natively but there exist some projects to develop OCaml interpreters for the platform. However, to run the reasoner on Android, we chose another approach: compiling CB build 6[6] to native Android code. The resulting native code can be executed on Android using the command line tool Android Debug Bridge (adb). To import this native code into an Android project we could use the Java Native Interface (JNI) and Android NDK (however, for the purpose of this paper we tested the native code directly).

**HermiT** [12] reasoner is implemented in Java and supports full OWL 2 and DL safe rules with reasoning based on a hypertableaux algorithm. It was the first DL reasoner that could classify some large ontologies thanks to a novel and efficient classification algorithm.

HermiT 1.3.6[7] cannot be converted directly to Dalvik as it references unsupported Java classes (both in its source code and in the external library JAutomata). Specifically, all the references to *java.awt.point* from both HermiT and

[4] https://code.google.com/p/androjena
[5] http://jfact.sourceforge.net
[6] https://code.google.com/p/cb-reasoner
[7] http://www.hermit-reasoner.com

JAutomata need to be changed or eliminated (they are used mainly for debugging, and Android does not support Java AWT as it has its own graphical libraries). In this way, we firstly eliminated the debug package of HermiT and its references, and reimplemented the methods of JAutomata that used these classes. However, in runtime, that version threw an error when loading ontologies with data properties due to a failure of Dalvik when unmarshalling the objects that the external library *dk.brics.automaton* serializes. To solve this problem, we reimplemented the marshalling/unmarshalling methods of these objects.

**Pellet** [13] reasoner is implemented in Java and supports full OWL 2 and DL safe rules with reasoning based on a tableaux algorithm. It was the first DL reasoner fully supporting OWL DL.

Pellet 2.3.0[8] (the last version available at the moment) cannot be converted directly to Dalvik. In this case, the reasoner uses three libraries that reference unsupported Java classes: Jena (which can be replaced by Androjena), OWL API 2.2.0 (which can be removed from the final compiled version), and JAXB (which uses the *javax.xml.bind* and the *Xerces* parser libraries not contained on Android). This problem can be solved by removing the JAXB .jar file and adding the source code of both *javax.xml.bind* and *Xerces* to our Android project. However, Dalvik has a limit of 65536 methods references per .dex file and it gets exceeded when applying this solution. To solve this, we removed the JAXB library and copied only the nine classes that Pellet needs from both the *java.xml.bind* package and the Xerces library to our Android project.

**Other Reasoners.** We also tried to load other reasoners on Android without success, namely RacerPro [4], KAON2 [8], QUEST [10], TrOWL [14], and fuzzyDL [2]. None of them can be directly converted to Dalvik because of their references to unsupported Java classes (similarly to what happened in HermiT and Pellet), and their source code was not publicly available. We identified that these reasoners use some problematic libraries that cannot be run on Android: Jena (QUEST, TrOWL), Java RMI (KAON2), Xerces (QUEST), Gurobi (fuzzyDL), etc. Both Jena and RMI could be replaced by projects that port these libraries to the Android platform (such as LipeRMI[9] or the aforementioned Androjena), and Xerces could be addressed as explained for Pellet. Finally, we have not found a replacement for Gurobi.

## 3 Experimental Evaluation

The main goal of this paper is to analyze whether current available DL reasoners can be used on Android. Hence, we considered also interesting to test their behavior on current devices. In this way, we tested the analyzed reasoners with five well-known ontologies (see Table 1): $Pizza$[10] and $Wine$[11], which are two

[8] http://clarkparsia.com/pellet
[9] http://lipermi.sourceforge.net
[10] http://www.co-ode.org/ontologies/pizza/pizza.owl
[11] http://www.w3.org/TR/owl-guide/wine.rdf

expressive ontologies; *DBpedia* 3.8[12] (T-BOX), which can be useful for mobile apps developers to access the structured content of DBPedia (a semantic entry point to Wikipedia) [1]; and the Gene Ontology (*GO*) and the US National Cancer Institute (*NCI*), which contain a high number of concepts.

**Table 1.** Selected ontologies for the tests.

| | DL Expressivity | Horn | $\|N_{LA}\|$ | $\|N_R\|$ | $\|N_C\|$ | $\|N_I\|$ |
|---|---|---|---|---|---|---|
| Pizza | $\mathcal{SHOIN}$ | × | 714 | 8 | 100 | 5 |
| Wine | $\mathcal{SHOIN}(\mathcal{D})$ | × | 950 | 17 | 138 | 206 |
| DBpedia | $\mathcal{ALF}(\mathcal{D})$ | ✓ | 3542 | 1894 | 436 | 0 |
| GO | $\mathcal{ALE}+$ | ✓ | 28897 | 1 | 20465 | 0 |
| NCI | $\mathcal{ALE}$ | ✓ | 46940 | 70 | 27652 | 0 |

Horn (✓): the ontology does not contain "non-deterministic" constructors.
$|N_{LA}|$: number of logical axioms; $|N_R|$: number of roles; $|N_C|$: number of concepts; $|N_I|$: number of individuals

For each ontology we tested the classification performance (i.e., time needed to compute the class subsumption hierarchy) of each reasoner on two devices with different Android versions. We also performed the tests on a PC to determine how slow is reasoning on Android compared to this baseline (taking into account that the PC hardware overperforms Android devices and their virtual machines are optimized for different purposes). The results obtained after performing 10 tests for each reasoner and device are shown in Table 2.

First, we want to highlight that the Galaxy Nexus with Android 4.2.1 (Android2) overperformed the Samsung Galaxy Tab with Android 2.3.3 (Android1) by 30% in all the tests. All the reasoners running on the Android 4.2.1 device were able to classify all the ontologies except *DBpedia* (which contains unsupported temporal data type properties for JFact –*gYear*–), and *NCI* (where JFact and Pellet ran out of memory). Notice that most of the reasoners running on the Android 2.3.3 device (JFact, HermiT, and Pellet) were unable to classify *GO* and *NCI* because of a memory constraint. Android 2.3 and earlier versions usually provide a maximum heap size limit per application of 64MB while later versions usually provide a maximum heap size of 256MB by using the *android:largeHeap="true"* attribute for the manifest of the application (the actual maximum size limit depends on the specific device). CB was able to classify all the ontologies but the reasoning for *Pizza* and *Wine* was incomplete because they are not Horn. The reasoning of CB on *DBpedia* was complete even when the profile of the ontology is not fully supported by the reasoner. In addition, CB does not face the same memory restriction than other reasoners as it was compiled from C code and runs outside Dalvik.

In summary, the major issue that reasoning on Android faces currently is the limited memory of smartphones/tablets (especially for apps running on Dalvik). This limitation affects especially when using large ontologies. Finally, as ex-

[12] http://dbpedia.org/Ontology

**Table 2.** Comparison of classification time for PC and Android in seconds.

| | | JFact | CB | HermiT | Pellet |
|---|---|---|---|---|---|
| Pizza | PC | 0.37 | $0^{\Diamond}$ | 0.57 | 0.97 |
| | Android1 | 4.90 | $0^{\Diamond}$ | 14.88 | 33.22 |
| | Android2 | 3.42 | $0^{\Diamond}$ | 10.43 | 20.77 |
| Wine | PC | 10.39 | $0^{\Diamond}$ | 6.54 | 2.22 |
| | Android1 | 2196.05 | $0^{\Diamond}$ | 511.97 | 194.12 |
| | Android2 | 1609.32 | $0^{\Diamond}$ | 361.38 | 131.80 |
| DBpedia | PC | UDT! | 0 | 0.10 | 1.39 |
| | Android1 | UDT! | 0 | 8.87 | 115.30 |
| | Android2 | UDT! | 0 | 5.13 | 63.15 |
| GO | PC | 7.77 | 0.11 | 1.56 | 1.96 |
| | Android1 | OOM! | 1.95 | OOM! | OOM! |
| | Android2 | 435.60 | 1.47 | 487.98 | 83.97 |
| NCI | PC | 2.61 | 0.24 | 2.23 | 4.24 |
| | Android1 | OOM! | 3.31 | OOM! | OOM! |
| | Android2 | OOM! | 2.69 | 2020.48 | OOM! |

*PC*: Windows 64-bits, i5-2320 3.00GHz, 16GB RAM; *Android1*: Samsung Galaxy Tab, 1.0GHz, 512MB RAM, Android 2.3.3; *Android2*: Galaxy Nexus, 1.2GHz dual-core, 1GB RAM, Android 4.2.1
0: time below 0.005s; $^{\Diamond}$: incomplete reasoning; *OOM!*: Out of Memory; *UDT!*: Unsupported Data Type

pected, we observe that reasoning on Android is slower than reasoning on a PC but nevertheless the results show that this is feasible.

## 4 Conclusions and Future Work

The emergence of mobile computing in our daily lives allows to consider new applications where semantic technologies could be useful. However, some efforts are needed to enable developers to use ontologies and ontology reasoning in their mobile apps. In this paper, we shown that current Android devices could be able to use most of the semantic reasoners although they need some manual work due to unsupported Java libraries and classes for the virtual machine of Android (Dalvik). Once this issue has been addressed, the main limitation that reasoners will face on current smartphones/tablets concerns memory usage and processing requirements. However, we noticed an increment of 30% on the performance of the reasoners between two Android devices (a Samsung Galaxy Tab and a Google Galaxy Nexus –introduced on 2010 and 2011, respectively–) which could show the future trend. As future work we plan to further test current semantic reasoners on Android measuring other important aspects for mobile computing such as memory and battery usage.

*Acknowledgments*: This research work has been supported by the CICYT project TIN2010-21387-C02 and DGA-FSE.

## References


1. C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. DBpedia - A crystallization point for the Web of Data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):154–165, 2009.
2. F. Bobillo and U. Straccia. fuzzyDL: An expressive fuzzy description logic reasoner. In *Proceedings of the International Conference on Fuzzy Systems (FUZZ-IEEE 2008)*, 2008.
3. T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
4. V. Haarslev and R. Möller. RACER system description. In *Proceedings of the First International Joint Conference on Automated Reasoning (IJCAR 2001)*, 2001.
5. M. Horridge and S. Bechhofer. The OWL API: A Java API for OWL ontologies. *Semantic Web Journal*, 2(1):11–21, 2011.
6. S. Ilarri, A. lllarramendi, E. Mena, and A. Sheth. Semantics in location-based services – guest editors' introduction for special issue. *IEEE Internet Computing*, 15(6):10–14, 2011.
7. Y. Kazakov. Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In *Proceedings of the 21st International Joint Conference on Artificial intelligence (IJCAI 2009)*, 2009.
8. B. Motik and R. Studer. KAON2–a scalable reasoning tool for the semantic web. In *Proceedings of the 2nd European Semantic Web Conference (ESWC 2005)*, 2005.
9. H.-S. Oh, B.-J. Kim, H.-K. Choi, and S.-M. Moon. Evaluation of Android Dalvik virtual machine. In *Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES 2012)*, 2012.
10. M. Rodriguez-Muro and D. Calvanese. Quest, an OWL 2 QL reasoner for ontology-based data access. In *Proceedings of the 9th International Workshop on OWL: Experiences and Directions (OWLED 2012)*, 2012.
11. M. Ruta, F. Scioscia, G. Loseto, F. Gramegna, and E. D. Sciascio. A mobile reasoner for semantic-based matchmaking. In *Proceedings of the 6th International Conference on Web Reasoning and Rule Systems (RR 2012)*, 2012.
12. R. Shearer, B. Motik, and I. Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*, 2008.
13. E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):51–53, 2007.
14. E. Thomas, J. Z. Pan, and Y. Ren. TrOWL: Tractable OWL 2 reasoning infrastructure. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010)*, 2010.
15. D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: system description. In *Proceedings of the Third International Joint Conference on Automated Reasoning (IJCAR 2006)*, 2006.
16. R. Yus, E. Mena, S. Ilarri, and A. Illarramendi. SHERLOCK: A system for location-based services in wireless environments using semantics. In *22nd International World Wide Web Conference (WWW 2013)*, 2013.