# Migration of rule inference engine to mobile platform. Challenges and case study.

Mateusz Ślażyński, Szymon Bobek, Grzegorz J. Nalepa

AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Krakow, Poland {mateusz.slazynski,szymon.bobek,gjn}@agh.edu.pl

Abstract Mobile devices are valuable sources of information about their user location, physical and social activity, profiles and habits. Such an information can be used to build context-aware applications, that are able to adapt their functionality to user needs and preferences. A lot of research have been done in this field of science, providing multiple context-modelling approaches like rules, ontologies, probabilistic graphical models, etc. There were also several solutions developed that allow for efficient context-based reasoning. However, there is still lack of tools and research done in the area of context-awareness with respect to mobile environments. The solutions that were constructed for the desktop platforms cannot be directly migrated to the mobile systems, as the requirements and characteristics of this two environments are disjoint. In this paper we focus on migrating a lightweight rule-based inference engine to a mobile platform. We define the requirements that have to be fulfilled by the mobile reasoning system in order to be efficient and universal with respect to the variety of mobile operating systems available nowadays.

# 1 Introduction

Mobile devices such as smart phones or tablets have been becoming better in terms of hardware capabilities, including speed and storage. Moreover, they are equipped with number of sensors gathering a lot of environmental data about device and user context. This triggered an apparent need for complex application running on mobile platforms. In number of situations such platforms have become viable alternatives for classic ones, e.g. PC. Therefore, engineering of software on mobile platforms uses similar tools and techniques to regular software engineering. This also includes methods from the domain of knowledge engineering, since real-time processing of large numbers of data needed to program complex mobile applications requires intelligent techniques.

Rules, are one of the most important techniques in knowledge engineering. Rulebased tools, including rule engines are a first class citizen in number of intelligent software systems developed on PC platforms, server and cloud. Therefore, there has been a growing demand to use rule-based tools on mobile platforms. This turns out to be not a trivial issue. In fact classic rule engines are very hard to be ported, due to either of the source code base (e.g. CLIPS/Jess) or runtime requirements (e.g. Drools). This gives motivation to investigate opportunities to port or develop a mobile rule engine. In this paper we present the results of such consideration and discuss a specific case study of the HeaRT [1] rule engine developed by us.

The rest of the paper is organized as follows: In Sect. 2 related works is discussed along with the detailed motivation for our research. Then in Sect. 3 we specify requirements for porting an inference engine to a mobile environment. Based on them we discuss possible scenarios in Sect. 4. In Sec. 5 we focus on porting HeaRT to a mobile environment. The paper ends with the summary in Sect. 6.

## 2 Related works and motivation

In recent years, a lot of development was devoted to build applications that use mobile devices to monitor and analyze various user contexts.

The SocialCircuits platform [2] uses mobile phones to measure social ties between individuals, and uses long- and short-term surveys to measure the shifts in individual habits, opinions, health, and friendships influenced by these ties. Jung [3] focused on discovering social relationships between people. He proposed an interactive approach to build meaningful social networks by interacting with human experts, and applied the proposed system to discover the social networks between mobile users by collecting a dataset from about two millions of users. Given a certain social relation (e.g., isFatherOf), the system can evaluate a set of conditions (which are represented as propositional axioms) asserted from the human experts, and show them a social network resulted from data mining tools. Sociometric badge [4] has been designed to identify human activity patterns, analyze conversational prosody features and wirelessly communicate with radio base-stations and mobile phones. Sensor data from the badges has been used in various organizational contexts to automatically predict employee's selfassessment of job satisfaction and quality of interactions. Eagle and Pentland [5] used mobile phone Bluetooth transceivers, phone communication logs and cellular tower identifiers to identify the social network structure, recognize social patterns in daily user activity, infer relationships, identify socially significant locations and model organizational rhythms.

Besides research projects, there exist also a variety of application that are used for gathering information about context from mobile devices, like SDCF [6], AWARE <sup>1</sup>, JCAF [7], SCOUT [8], ContextDriod [9], Gimbal <sup>2</sup>. These are mostly concerned with low-level context data acquisition from sensors, suitable fur further context identification. On the other hand, they do not provide support nor methodology for creating complex and customizable context-aware systems.

Although there is a lot of frameworks and middlewares developed for context-aware systems, they are usually limited to a specific domain and designed without taking into consideration mobile platforms. Examples include CoBrA [10] and SOUPA [11] for building smart meeting rooms, GAIA [12] for active spaces, Context Toolkit [13].

There is still space for research in a field of lightweight context modeling and context reasoning targeted at mobile devices. Some attempts were made to develop such

<sup>&</sup>lt;sup>1</sup> http://www.awareframework.com/

<sup>&</sup>lt;sup>2</sup> https://www.gimbal.com/

frameworks, like SOCAM [14], or Context Torrent [15]. There were also attempts to migrate existing rule engines to the mobile platforms. In our preliminary approach we investigated several candidates for the mobile reasoning engines, including: Jess<sup>3</sup>, Context Toolkit [13], ContextDroid [16] and Context Engine [17]. Although some of them were successfully migrated and launched on the Android device, none of these solution fully supported the requirements that we believe are crucial for mobile computing with respect to the context-based reasoning (see Section 3).

Taking all the above into consideration, our primary motivation for the work presented in this paper was to:

- investigate possible scenarios for migrating existing rule-based engines to mobile platforms,
- define requirements that all the inference engines should meet in order to provide efficient, lightweight context-based reasoning on mobile platforms.

In our research we focused on the rule-based reasoners, because they are characterized by the high intelligibility [18] capabilities, which is a crucial factor in designing user-centric applications. Rule-based approach provides also very efficient reasoning, and intuitive modeling languages that help the end-user to understand the system and cooperate with it. The work presented in this paper is a continuation of our previous attempt of migration HeaRT rule-based engine to Android platform, which was successful only in a small fraction [19]. We managed to migrate rule-based reasoning engine to mobile platform, however the efficiency of this migrated version was far worse than the desktop one. Therefore, we decided to perform an extensive case study on the possible migration scenarios, which will allow us to define universal requirements for the mobile reasoning engine. The following section describes in details these, and shows which of them can be fulfilled by the existing solutions.

## **3** Requirements for mobile inference engine

Mobile devices are now dominated by Android and iOS operating systems. However to design a fully portable reasoning engines other systems should also be taken into consideration. The portability of the software between different mobile platforms is however not a trivial task. Therefore, to support the process of efficient migration of the inference engines to mobile environments, the following requirements were identified [20]:

- Portability (R1). The inference engine should be portable and as independent of the operating system as possible.
- Responsiveness (R2). The inference layer has to work under soft real-time constraints. Mobile environment is highly dynamic, and the inference layer should follow rapid changes of context in such an environment.
- Resource limitation (R3). It should consume as least resources as possible to work transparently in the background.

<sup>&</sup>lt;sup>3</sup> http://www.jessrules.com

- Privacy (R4). The reasoning service should not send any confidential data to the external servers, but perform all the reasoning locally.
- Robustness (R5). It should work properly when the contextual data is incomplete or uncertain. This requirement is beyond of the scope of this article and is just briefly described in the following paragraphs.
- Intelligibility (R6). It should be able to explain its decision to the user, and thus improve intelligibility of the system.

Due to the development of programming languages which execution is performed by the virtual machines, it became possible to successfully migrate existing rule-based inference engines directly onto different mobile platforms. For example Java bytecode can be executed not only on Android operating system, which supports it natively, but also on iOS and other platforms that provides appropriate Java Virtual Machine implementations. This may give a misleading impression that the requirement (R1) could be easily fulfilled. Our research shown that this is not true, as the quality of the virtual machines is not always satisfactory and hence, although portability is possible it may indirectly affect fulfillment of other requirements. For instance, if the rule-based engine is written in a programming language that is not natively supported by the mobile platform, but depends on the virtual machine that allows executing it, the responsiveness (R2) of such system may be affected by the inefficient implementation of the virtual machine. What is more, depending on the complexity of the inference engine, such migration may not always fully succeed, especially if the inference engine is based on the technology that is not supported by the mobile implementation of the virtual machine (i.e. Hibernate<sup>4</sup> for knowledge management in Drools and Context Toolkit). Some technologies even though possible to migrate to mobile platform, loses their efficiency, due to limited memory or CPU resources on the mobile device. Thus, the (R1) requirement should be achieved by choosing a programming language that is equally and efficiently supported by all the available mobile platforms. Currently to our knowledge there is such solution available.

Responsiveness (R2) of the system is usually affected by the efficiency of its implementation. Hence, this requirement is very tightly bound with (R3), which states that the mobile inference engine should consume as low CPU and memory resources as possible . Complex rule-based solutions like Drools or ContextToolkit although successfully launched on the Android mobile devices, were very inefficient as they depend on the parts of the JVM that is not supported by Dalvik Virtual Machine present in Android. The very important issue that tackles the problem of resource limitation is connected with energy consumption. Most of the sensors which are primary information providers about the environment, when turned on all the time, decrease the mobile device battery level very fast. This reflects on usability of the system and ecological aspects regarding energy saving. Currently most of the solutions (including these that were designed for mobile platforms like ContextEngine) rely on the built in mechanisms for sensor energy consumption.

Having in mind requirement (R3), the reasoning engines that make use of this information should be designed to best manage the tradeoff between high quality, accurate

<sup>&</sup>lt;sup>4</sup> http://hibernate.org/orm/

information form the sensors and energy efficiency [21]. Such a design of a reasoning engine requires on the other hand a mechanism that could cope with uncertain and incomplete knowledge. This can be guaranteed by fulfilling the requirement (R5). It is strictly connected with the nature of the mobile environment. Although the mobile devices are currently equipped with a variety of sensors, the contextual data provided by them is not always complete nor certain. For instance the location provider will not work properly underground, where there is no access to the GPS or the Internet. Currently only a ContextDroid (former SWAN) tackles this issue by introducing expiration time and temporal reasoning.

Another requirement that indirectly is connected with energy efficiency and therefore tackles the requirement (R3) is privacy requirement (R4). To improve energy efficiency of the intelligent mobile system, the most time and energy consuming tasks could be performed externally by the reasoning server. This however requires from the users to agree to share their private contextual information with the service provider, which in some countries like European Union is regulated by the law and not easily accessible. What is more, most of the users do not want to send information about their location, activities, and other private data to external servers. Hence, the context reasoning should be performed locally by the mobile device. This requirement was also one of the first motivations for our attempts to migrating reasoning engines to a mobile devices so they could work as a local inference services for other applications. Although not explicitly stated, all of the existing engines rely on the local knowledge bases. Even Drools and ContextToolkit that allows storing knowledge in remote database can be adapted to use only local storage.

Requirement (R6) on the other hand is the main reason why we decided to use rule-based engines. Besides very efficient reasoning mechanisms, rule-based systems in most cases fulfill the (R6) requirement by the definition. Most of the existing solutions have an explanation mechanism, which allows to trace back the inference chain and therefore explain the user what was the causes, conclusions and actions that was taken based on the previous both. The Context Toolkit s the only solution that provides intelligibility support with a special module designed for this. However, our attempts at migrating it to the mobile environment failed due to the problems in parsing module of Context Toolkit.

# 4 Possible scenarios

One of the biggest challenges for the mobile software creators is the large variety of technologies used in the mobile operating systems. Due to the more tightly coupled runtime environment than the one met in the traditional desktop operating system, a programmer has a very narrow choice of programming tools and languages. To make things even worse, every one of the leading mobile platforms promotes its own ecosystem incompatible with any other. Therefore it is not surprising that the industry is constantly looking for the cross-platform solutions. This section will concern possible scenarios of the mobile inference engine development, taking into account issues mentioned in this paragraph.

In the rest of the article we will focus on the three most popular platforms, namely: Android, iOS and Windows Phone. Despite this, most of the article applies also to Firefox OS which supports only the web technologies. There exist also a few Linux based operating systems, supporting software written for the desktop platforms — many of them, like Sailfish OS, are trying to reuse the Android ecosystem.

**Table 1.** Technologies supported by the most popular mobile platforms..

Technology / OS	Android	iOS	Windows Phone
JVM based		One-way interface only	Ŋ
	Native.	through non-standard	None.
		Java Virtual Machines.	
C/C++	Two-way interface	Native.	Two-way interface
	using Android NDK.		starting from WP 8.
Objective-C	None.	Native.	None.
CLR based	One-way interface	One-way interface	Native.
	through Mono.	through Mono.	
		JavaScriptCore <sup>6</sup> support	
JavaScript	New devices include	starting from iOS 7.	Includes proprietary
	V8 JavaScript engine. <sup>5</sup>	Many independent	JavaScript engine.
		solutions.	
Lua	Many implementations, including LuaJIT.	Supported without JIT.	Supported without JIT.

## 4.1 Supported technologies

Before presenting the proposed approaches, we will briefly summarize the mobile programming toolset available for the developers. Table 1. contains short summary of supported languages on different platforms. However, it needs some explanation carried out in the next three paragraphs.

Firstly, we should explain our choice of the technologies presented in the table. JVM, CLR and C-family were obvious choices as the native technologies on some platform — their use would simplify implementation at least on the one family of devices. On the other hand, JavaScript and Lua<sup>7</sup> are presented because of their small and easily accessible runtimes, what makes them perfect candidates for the cross-platform embedded solutions. Lack of the JIT support for Lua means that we cannot use the LuaJIT<sup>8</sup> —

<sup>&</sup>lt;sup>5</sup> https://code.google.com/p/v8/

<sup>&</sup>lt;sup>6</sup> http://trac.webkit.org/wiki/JavaScriptCore

<sup>&</sup>lt;sup>7</sup> http://www.lua.org

<sup>8</sup> http://luajit.org

a very fast Lua implementation. There are of course other technologies missing from the table, but we were looking only for the tested and production ready implementations.

Secondly, "one-way interface support" means that we can build an application in the selected language, but the code written in it cannot be directly called within a native application. For example: programmer can write an entire application using Mono<sup>9</sup> on iOS, but can not use library written in C# within native application.<sup>10</sup> These inconveniences render Mono as not suitable for our purposes. JVM based languages share the same problem — necessity of running JVM on the system dramatically reduces its possibilities to reuse on the other platforms.

Finally, despite the shared support for the C-family languages, we cannot freely share code written in C/C++ between all these platforms. In fact there are many differences mainly due to different compilation toolchains and standard libraries included in the system — for example Android uses its own Bionic C standard library. Furthermore, while on iOS the C/C++ code can be freely mixed with Objective-C, both Android and Windows Phone need a foreign function interface to call the C functions from Dalvik/CLR virtual machines. Lately it became possible to compile Objective-C code into Android assembler, but it is a very immature project. To conclude, shared C codebase is possible, but we should not expect it will work on the all systems without any platform-dependent fixes.

#### 4.2 Proposed solutions

Leaving aside for the moment technical differences between the mobile platforms we can distinguish different approaches to the cross-platform software development on the basis of the code base architecture, whether it is fully or only partly shared between supported platforms. Particularly there can be listed three border cases:

- Separated code bases (S1) every platform has its own independent code base.
- Shared code base (S2) all platforms share the same code base.
- Hybrid approach (S3) there exist elements of project implemented especially (particularly API interface) for the different platforms, but we can also distinguish shared parts of code.

In this section we will confront all these approaches with technical characteristic of mobile operating systems presented in the previous section.

Foremost, according the the S1 approach, we can simply have separate code bases for every platform. In spite of the obvious disadvantages of this approach, which include larger support and development costs, this is a perfectly sane solution for the projects aiming mainly for the high quality implementation for the one selected platform and less supported ports for the other systems. Moreover, this way we can take advantage of the features characteristic for the different runtimes; in the context of the mobile libraries it has a great impact on the quality of API available for the programmers. Therefore this

<sup>9</sup> http://www.mono-project.com/Main\_Page

<sup>&</sup>lt;sup>10</sup> It is not exactly true. Linking Mono code to the native iOS application is possible but troublesome and not production ready.

is a very popular approach used for so called Software Development Kits provided by such companies like Google or Facebook.

On the other hand, when we are interested in supporting possible many (maybe even not known in the beginning of the project) environments, the costs related to the code maintenance can be overwhelming. S2 seems to be an universal solution for these problems, however we must remember that there is always a trade-off between universality and expressiveness of the resulting code — programmer is forced to abstract from the platform dependent features. From the technical point of view there are two distinct methods to achieve the shared code base on mobile devices. In the first scenario we could use a technology support on every platform, we have interest in. Again, in the context of table 1 we could distinguish two possible cases presented below.

*Low level approach (S2LL).* Most modern operating systems support some kind of the low level interface to the C-family languages. C/C++ code base has many advantages — first of all it has a great performance and, secondly, it is very popular among programmers, so there should not be a problem with the reuse of the existing solutions and libraries. Unfortunately, as it was indicated before, there are also significant differences between the low level characteristics of the operating systems — they differ in used standard libraries and methods of the code calling from the native platform language. These differences eventually lead to the corner cases maintained by platform specific code contained in the conditionally compiled sections of project. Second, the more controversial disadvantage of the low level code is that is harder to maintain than its higher level counterparts. Due to the efficiency reasons, the low level approach is widely adapted by the game development industry. The most popular examples include cocos2d-x<sup>11</sup> and Marmalade<sup>12</sup> libraries.

High level approach (S2HL). Nowadays there can be observed the rapid growth of dynamic languages; particularly web technologies like JavaScript are regarded as a new kind of a portable assembler language. Thanks to the efficient, highly optimized and widely available runtime it recently became the most popular platform for the crossplatform applications. Lua is an other, less known language with similar features lower popularity of this platform is compensated with more sane language characteristic and even smaller runtime, ported already to the most exotic hardware platforms. Unlike in C-family languages, there should be no need for the platform-dependent sections of code in project written using high level languages. Moreover, distribution of the dynamic code is much easier - it does not need the compilation step and can be distributed as plain text through the Internet. The main concern about high level approach regards its efficiency; dynamic code tends to be slower, however due to the rapid growth of the technology stack, the gap between low level and high level code successfully decreases. Given these facts, it should be not surprising that currently there appear more and more high level frameworks, which could be used to create the mobile applications from scratch (for example PhoneGap<sup>13</sup> or Sencha Touch<sup>14</sup>) including even more de-

<sup>&</sup>lt;sup>11</sup> http://www.cocos2d-x.org/

<sup>&</sup>lt;sup>12</sup> https://www.madewithmarmalade.com/

<sup>&</sup>lt;sup>13</sup> http://phonegap.com

<sup>&</sup>lt;sup>14</sup> http://www.sencha.com/products/touch

manding products like game engines (for example LÖVE<sup>15</sup> and Loom<sup>16</sup>, both written in Lua).

The second scenario concerns the more complicated **compiling toolchain (S2CT)** — precisely, before the deployment of the project, we could translate it into the native platform technology. This way we could work on the shared code base without worrying about differences between the platforms. Recently this approach receives a lot of attention — there are many technologies treating C or JavaScript as the portable assembler languages. Especially the latter, given its easy deployment on the web, is currently the compile target of the almost every popular programming language, including even the low level technologies. While making C or JavaScript our compile target does not differ much from using them as the project main language depending on the target platform. This way we could maintain portability of the project and also benefit from native API on every supported platform at the same time. Remarkably, the Haxe<sup>17</sup> language can be compiled into JavaScript, C++, C# and Java, trying to make this optimistic scenario possible.

Given the advantages of both S1 and S2 approaches, the hybrid approach simply combines them through the greater modularization of project. While the core black box part of the project can be written in the portable way, possibly using S2HL or S2LL methods, the interface layer should be interchangeable and written in regard to each supported platform separately. Consequently core of the project can truly abstract from platform dependent code (as in the optimistic cases of S2) and it is possible to make a high quality API for the end-users (as in S1). This approach is widely used by the so called wrappers — libraries written in the high level technology, which only wrap the low level library written already in C or other efficient technology. The proposed architecture of the inference engine implemented using the hybrid approach is presented in the figure 1, where the Platform Independent Runtime can mean low level machine programmable using C-family language or some kind of virtual machine destined to run the high level code, for example Lua interpreter or V8 JavaScript engine. The API part of the architecture should contain an query interface and methods to load rules from their plain representation.

#### 5 HeaRT inference engine on mobile platforms

This section will focus on the case of porting the HeaRT inference engine [1] to the mobile platforms. HeaRT is a lightweight rule-based inference engine that uses XTT2 [22] notation for knowledge representation [23].

The first part of the section will briefly describe our efforts to port existing code to the Android ecosystem, which was believed to be a good platform for the "proof of concept" implementation. The second part will contain plans for the future work, based on the experience gained from the unsuccessful attempts and research results presented earlier in section 4.

<sup>&</sup>lt;sup>15</sup> https://love2d.org/

<sup>&</sup>lt;sup>16</sup> https://www.loomsdk.com/

<sup>&</sup>lt;sup>17</sup> http://haxe.org

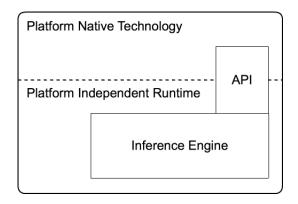


Figure 1. Proposed architecture of the portable interface engine using the hybrid approach.

#### 5.1 Previous attempts

The original implementation of HeaRT was written in the Prolog language and was executed by SWI–Prolog, an open source ISO-compliant Prolog implementation<sup>18</sup>. Thanks to the Prolog metaprogramming and reasoning features it was easy to represent rules as simple logical clauses and querying could be expressed with standard Prolog queries. Therefore the first attempts to port the engine were focused mainly on possible ways of executing Prolog code on the mobile platforms, particularly Android as it was stated in previous paragraph. The detailed description of these efforts can be found in [19]. In the terminology of Section 4.2 all attempts presented below can be assigned to the S2 category.

First and conceptually the simplest attempt can be categorized as S2LL and consisted of porting the SWI–Prolog environment to the Android platform. Due to the source code written almost entirely in C it was hoped to succeed without breaking changes in the environment. Unfortunately, it could not be carried successfully mainly because of the two major issues:

- SWI-Prolog contains large parts of platform dependent code, dealing with the low level facilities like threading support, console interface, etc. Due to the lack of the libraries and different standard library on Android, migration of the SWI-Prolog must involve further refactoring and new conditionally compiled lines of code.
- The interface between Java and SWI–Prolog was found to be not satisfactory, particularly it was not clear, whether it would work with the Dalvik Java virtual machine.

Next attempt was supposed to leave aside issues connected with the low level character of the popular Prolog environments and was oriented on their pure Java counterparts (therefore it can be regarded as a S2HL type). Following the carefully carried research four possible candidates were proposed: tuProlog<sup>19</sup>, jekejeke Prolog<sup>20</sup>, Gnu

- <sup>19</sup> http://apice.unibo.it/xwiki/bin/view/Tuprolog/
- <sup>20</sup> http://www.jekejeke.ch/idatab/doclet/intr/en/docs/package.jsp

<sup>&</sup>lt;sup>18</sup> http://www.swi-prolog.org/

Prolog for Java<sup>21</sup> and jinniprolog<sup>22</sup>. Despite their supposed ISO-compliance, none of them could be really regarded as a fully working Prolog environment. Nevertheless, after large refactoring of HeaRT source code we were finally able to run it using the tuProlog environment. Unfortunately, performance of this solution was unacceptable, clearly not satisfying the R2 and R3 requirements presented in Section 3.

The last used technique concerned translation of Prolog code into Java (therefore it was representative of the S2CT approach). Unfortunately, evaluated Prolog Café translator<sup>23</sup>, in spite of efficiency, lacked many advanced features used broadly in the HeaRT code, explicitly it dose not support mixing of the dynamic and static predicates. Furthermore, the text representation of HeaRT rules being itself a valid Prolog code, was also needed to be compiled to Java class, making the dynamic loading of the rules very inflexible.

To sum up the foregoing, Prolog proved to be too unpopular language to have adequate running environments on the mobile platforms. Consequently, all approaches of S2 type have to be preceded by the creation (or adaption) of efficient and portable Prolog environment. Due to related development cost, we have currently resigned from this approach.

#### 5.2 Future plans

The current plans for the mobile HeaRT development were created according to the S3 approach. The inference elements of HeaRT will be rewritten in the Lua language — the main reasons of this choice are low development cost and large variety of compact and efficient runtimes on even very exotic hardware. The architecture of so called LuaHeaRT matches the one presented in Figure 1. There were identified three major drawbacks of the selected approach:

- 1. The source code must be completely rewritten into language with different paradigm and capabilities. The related costs are not negligible, but inevitable.
- 2. Text representation of the rules in HeaRT is also a Prolog code, therefore in original implementation the task of processing it was performed entirely by SWI–Prolog. In the new architecture this process will be divided into two parts: parsing realized by parser included in the API module and semantic analysis performed entirely by the inference engine. Thanks to the formal grammar of the rule format it will be possible to semi-automatically generate parsers for the different platforms.
- 3. In the desktop version of HeaRT queries are defined as the standard Prolog queries. For the sake of the new implementation we must specify new, possibly portable method of creating queries. The related code should be automatically generated (similarly to the grammar parser) and included in the API module. On the other hand the inference engine must contain some kind of low level query API, which could be the target of the human friendly representation of queries.

<sup>&</sup>lt;sup>21</sup> http://www.gnu.org/software/gnuprologjava/

<sup>22</sup> http://code.google.com/p/jinniprolog/

<sup>&</sup>lt;sup>23</sup> https://code.google.com/p/prolog-cafe/

For the aforementioned reasons, there is no doubt that the portable implementation of HeaRT according to the S2 approach will involve the overall redesign of the desktop version. However, it is not entirely bad phenomenon, the end effects can be highly beneficial for the future development of the engine.

#### 6 Summary

In recent years, a lot of development was devoted to build applications that make use of contextual information to behave in an intelligent way. Due to the evolution of mobile devices which became omnipresent in every human life, the context-aware application became also one of the primary focus of mobile system developers. However, mobile and desktop environments are different, and therefore migrating solutions between them is not a trivial task. In this paper we focused on the problem of migration of a rule engine to mobile platform.

We defined requirements that every mobile reasoning engine should fulfill to provide efficient reasoning solution. We confronted these requirements with an attempt to migrate Prolog-based rule engine HeaRT onto the mobile platform. Finally, we presented a study of different migration scenarios, and propose a solution that provides both efficiency, portability and allows for effective source code maintenance.

## References

- Nalepa, G.J.: Architecture of the HeaRT hybrid rule engine. In Rutkowski, L., [et al.], eds.: Artificial Intelligence and Soft Computing: 10th International Conference, ICAISC 2010: Zakopane, Poland, June 13–17, 2010, Pt. II. Volume 6114 of Lecture Notes in Artificial Intelligence., Springer (2010) 598–605
- Chronis, I., Madan, A., Pentland, A.S.: Socialcircuits: the art of using mobile phones for modeling personal interactions. In: Proceedings of the ICMI-MLMI '09 Workshop on Multimodal Sensor-Based Systems and Mobile Phones for Social Computing. ICMI-MLMI '09, New York, NY, USA, ACM (2009) 1:1–1:4
- Jung, J.J.: Contextualized mobile recommendation service based on interactive social network discovered from mobile users. Expert Syst. Appl. 36 (2009) 11950–11956
- Olguin, D., Waber, B.N., Kim, T., Mohan, A., Ara, K., Pentland, A.: Sensible organizations: Technology and methodology for automatically measuring organizational behavior. IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B: CYBERNET-ICS (2009) 43–55
- 5. Eagle, N., (Sandy) Pentland, A.: Reality mining: sensing complex social systems. Personal Ubiquitous Comput. **10** (2006) 255–268
- Atzmueller, M., Hilgenberg, K.: Towards capturing social interactions with sdcf: An extensible framework for mobile sensing and ubiquitous data collection. In: Proc. 4th International Workshop on Modeling Social Media, ACM Press (2013)
- Bardram, J.E.: The java context awareness framework (JCAF) a service infrastructure and programming framework for context-aware applications. In Gellersen, H.W., Want, R., Schmidt, A., eds.: Pervasive Computing. Volume 3468 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2005) 98–115
- 8. Woensel, W.V., Casteleyn, S., Troyer, O.D.: A Framework for Decentralized, Context-Aware Mobile Applications Using Semantic Web Technology. (2009)
- 9. van Wissen, B., Palmer, N., Kemp, R., Kielmann, T., Bal, H.: ContextDroid: an expressionbased context framework for Android. In: Proceedings of PhoneSense 2010. (2010)
- Chen, H., Finin, T.W., Joshi, A.: Semantic web in the context broker architecture. In: Per-Com, IEEE Computer Society (2004) 277–286

- Chen, H., Perich, F., Finin, T.W., Joshi, A.: Soupa: Standard ontology for ubiquitous and pervasive applications. In: 1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2004), Networking and Services, 22-25 August 2004, Cambridge, MA, USA, IEEE Computer Society (2004) 258–267
- 12. Ranganathan, A., McGrath, R.E., Campbell, R.H., Mickunas, M.D.: Use of ontologies in a pervasive computing environment. Knowl. Eng. Rev. **18** (2003) 209–220
- 13. Dey, A.K.: Understanding and using context. Personal Ubiquitous Comput. 5 (2001) 4-7
- 14. Gu, T., Pung, H.K., Zhang, D.Q., Wang, X.H.: A middleware for building context-aware mobile services. In: In Proceedings of IEEE Vehicular Technology Conference (VTC. (2004)
- 15. Hu, H., of Hong Kong, U.: ContextTorrent: A Context Provisioning Framewrok for Pervasive Applications. University of Hong Kong (2011)
- Palmer, N., Kemp, R., Kielmann, T., Bal, H.: Swan-song: A flexible context expression language for smartphones. In: Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones. PhoneSense '12, New York, NY, USA, ACM (2012) 12:1– 12:5
- Kramer, D., Kocurova, A., Oussena, S., Clark, T., Komisarczuk, P.: An extensible, self contained, layered approach to context acquisition. In: Proceedings of the Third International Workshop on Middleware for Pervasive Mobile and Embedded Computing. M-MPAC '11, New York, NY, USA, ACM (2011) 6:1–6:7
- Dey, A.K.: Modeling and intelligibility in ambient environments. J. Ambient Intell. Smart Environ. 1 (2009) 57–62
- Szymon Bobek, Grzegorz J. Nalepa, M.S.: Challenges for migration of rule-based reasoning engine to a mobile platform. In Dziech, A., Czyżewski, A., eds.: Multimedia Communications, Services and Security. Volume XX of Communications in Computer and Information Science., Springer Berlin Heidelberg (2014) accepted.
- Nalepa, G.J., Bobek, S.: Rule-based solution for context-aware reasoning on mobile devices. Computer Science and Information Systems 11 (2014) 171–193
- Bobek, S., Porzycki, K., Nalepa, G.J.: Learning sensors usage patterns in mobile contextaware systems. In: Proceedings of the FedCSIS 2013 conference, Krakow, IEEE (2013) 993–998
- 22. Nalepa, G.J., Ligeza, A., Kaczor, K.: Formalization and modeling of rules using the XTT2 method. International Journal on Artificial Intelligence Tools **20** (2011) 1107–1125
- Ligeza, A., Nalepa, G.J.: A study of methodological issues in design and development of rule-based systems: proposal of a new approach. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 1 (2011) 117–137