A light reasoning infrastructure to enable context-aware mobile applications

Josué Iglesias, Ana M. Bernardos
Universidad Politécnica de Madrid
Madrid, Spain
{josue, abernardos}@grpss.ssr.upm.es

Alejandro Álvarez, Marcos Sacristán
Treelogic
Asturias, Spain
{alejandro.alvarez, marcos.sacristan}@treelogic.com

Abstract — **This paper presents a light inference system for context management, ready to perform reasoning tasks in resource-constrained devices. The inference system is part of a service-oriented mobile software framework which runs on Java-enabled handheld devices. This framework serves to facilitate the creation of context-aware applications, as it decouples sensor's data acquisition and context processing from the application logic itself. The inference architecture is composed by different modules; some of them encapsulate existing tools (μJena and Bossam) that have been adapted to the mobile working environment. The (rule-based) reasoning is prepared to use a general ontology. Both applications and the framework's components may configure the query set in order to retrieve the information they need from the inference system. In the paper, a validation example shows how this process is done.**

Keywords - Context-awareness; data modeling; light ontology management; reasoning; service-oriented architectures

I. Introduction

Mobile applications are increasingly capable of adapting their performance to the changing situation of their users. This is mainly possible thanks to the data retrieved from different sensors embedded in the mobile devices, which may provide a lot of information about the user's context if adequately processed.

In order to manage context information, many frameworks have been proposed to date. Some of them make use of off-the-shelf toolkits designed to parse data models and perform automatic reasoning. Most of them are unable to be deployed in mobile devices, as the toolkits they rely on are not targeted at resource-constrained devices. There are some proposals which work in mobile environments, but they usually lack flexibility as they are adhoc solutions. In this paper, we propose a reasoning system to be integrated in the PIRAmIDE mobile framework (PIRAmIDE states for '*Personalizable Interactions with Resources in AmI-enabled mobile Dynamic Environments*'), a service-oriented architecture conceived to accelerate the development of context-aware applications. The objective is to provide both applications and modules in the framework with a light and multi-domain infrastructure to perform automatic reasoning, in order to alleviate them of accomplishing these tasks.

Section II reviews existing embedded toolkits for data model management and reasoning. Section III briefly presents the framework PIRAmIDE, in which we have integrated our reasoning system. The framework's reasoning needs, together with its general data model are detailed in Section IV. Our embedded reasoning system and an example for validation are presented in Sections V and VI. Section VII concludes the paper with an analysis of the drawbacks and key issues we have identified when designing and building the reasoning system.

II. State-of-the-Art: Mobile Reasoning Systems

Most of the existing proposals for context management and reasoning (see [1], [2] for a survey) rely on server-based architectures which minimize the need for mobile devices to have any internal reasoning capability (e.g. [3], [4]). Of course, this centralized approach avoids to handle the restrictions imposed by mobile devices (limited e.g. in terms of processing, storage and batteries), but require permanent connectivity, which may lead to increasing response times, power consumption, security problems or service unavailability. On the contrary, device-embedded strategies for reasoning avoid permanent exchange of context information, securizing and accelerating the inference process.

Although scarce when comparing with general context management systems, some light tools enabling reasoning in resource-constrained devices have already been described in literature. Crivellaro et al. [5] developed *μJena*, which serves to manage ontologies stored in mobile devices although does not face ontological reasoning. *LOnt* [6] is another custom implementation of Jena API for mobile devices. Its main features are its small size and low memory fingerprint, which make it suitable to be used in J2ME mobile devices. Kleeman et al. [7] integrate the mobile *Pocket KRHyper* reasoner [8] for user's profile management and decision-making tasks. *Pocket KRHyper* is a powerful reasoning system for Java-enabled mobile devices, but it does not support any of the standard ontology languages (OWL, SWRL, etc.).

In [9], a mobile framework to support ontology processing and reasoning is proposed. The reasoning engine contains only a forward chaining rule-based inference engine which can be used to trigger the desired actions based on the rules that are explicitly defined, but it only supports a subset of OWL ontology inference rules. In this framework, a lightweight RDQL query engine supporting a subset of RDQL syntax is also developed. The *μOR* reasoner [10] is introduced as a part of a framework for developing AmI-based medical devices. At the moment it only reasons over a subset of OWL-Lite entailments. Vázquez [11] implements a 'MiniOwl and MiniRule' embedded reasoner, powered with ontologies and domain rules that can successfully interpret situations that were not previously solved without reasoning. This proposal only implements a subset of OWL Lite, too. Finally, *Bossam* [12] has native support for reasoning over OWL/SWRL ontologies and RuleML rules. Its runtime size is about 750Kb, running on J2ME CDC/PP platforms as well as J2SE platform of JDK 1.3 or later.

A transitional solution between server-based approaches and device-oriented ones are hybrid architectures. In MobileOntoDB [13], a real-time evaluation strategy is proposed: any query performed in the device is initially analyzed and, if it exceeds the device capabilities, it is then sent to a central reasoning server. A distributed case-based reasoning mechanism is used in AmbieSense [14], an agent-based infrastructure for context-based information delivery for mobile users, in which on-line reasoning resides on the user's mobile device, while off-line reasoning is done in the user's backbone system.

As previously stated, most of the proposals above are adhoc solutions to specific problems, hardly scalable or configurable. Up to our experience, the most feasible and versatile tools for data model management and reasoning are *µJena*, *Pocket KRHyper* and *Bossam* (see Section VII).

III. PIRAmIDE FRAMEWORK: FUNCTIONAL DESCRIPTION AND ARCHITECTURE

The light framework 'PIRAmIDE' aims at providing a set of standard features to build context-aware mobile applications, in order to support and accelerate their design and development life cycle.

PIRAmIDE provides easy access to the information acquired from a number of sensors, both embedded in the mobile device (accelerometers, gyroscopes, RFID readers, camera, etc.) or deployed within the user's environment (ambient sensors, QR codes, etc.). Additionally, the framework is also prepared to integrate context information from third parties (coming from virtual sensors, e.g. in-the-cloud calendars). Built on these *sensing enablers*, PIRAmIDE offers a set of application-independent services which may be used by the applications deployed on top of the framework to capture context data. Horizontal *services* are, for example, related to the physical detection of points of interest (through wireless technologies), the management of position estimation (based on GPS when outdoors and on WiFi, ZigBee and Bluetooth systems when indoors), image-based decoding of bidimensional codes and automatic reasoning.

PIRAmIDE is based on a service-oriented software architecture [15] composed by three main building blocks: *Sensing Subsystem*, *Context Management Subsystem* and *Core Subsystem* (Figure 1). The *Sensing Subsystem* decouples the access to embedded and external sensors from upper processing levels by wrapping sensor specific characteristics inside software units, which deals with low-level hardware information retrieval. The *Context Management Subsystem* is composed by a number of modules that process data coming from sensors (or from other modules), fuse them, and infer complex context parameters. Finally, the *Core Subsystem* provides several features to integrate software modules into the middleware, such as discovery and registry management of new elements and some common utility libraries. Applications consume context information provided by enablers using the features provided by PIRAmIDE *Core Subsystem*. Enablers and applications use an (asynchronous) event-based communication strategy, also leaded by the *Core Subsystem*, in order to exchange context data.

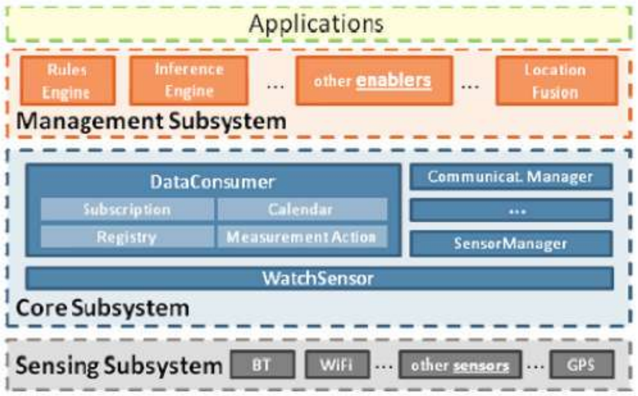

Figure 1. PIRAmIDE: mobile service-oriented framework architecture

PIRAmIDE has been developed using mobile OSGi (mOSGi). mOSGi is a Dynamic Module System for Java, handling modules referred as bundles (PIRAmIDE's *enablers*): cohesive, self-contained units, which explicitly define their dependencies to other modules/services and their external APIs. mOSGi improves encapsulation and reusability, simplifying the implementation of a modular system. Additionally, it is dynamically configurable in real time operation. Regarding its core functionalities, mOSGi enables a set of software tools useful for general service management and, particularly, for a fast and dynamic development of new services (automatic service registration, event management, logging service, etc.). PIRAmIDE middleware is currently implemented in Java, running on a mOSGi platform based on J9 Virtual Machine inside a Windows Mobile device. However, several implementations of mOSGi framework are available for other mobile operative systems such as Symbian or Android, so migration should be possible at a reasonable coding cost.

At present, the set of PIRAmIDE's sensors and horizontal enablers are being extended. Particularly, this paper refers to the design, development and integration of a new enabler providing reasoning services: a flexible and configurable tool to discharge applications of periodic reasoning tasks which may otherwise imply a higher cost both in terms of resources and quality of service.

IV. REASONING NEEDS IN PIRAMIDE

A. *Requirements for a embeddable reasoning infrastructure*

PIRAmIDE's reasoning system will initially rely on a generic data model which will offer support to the reasoning procedures of the horizontal services. The reasoning functionality is initially thought to be accessible from every PIRAmIDE component, although applications will be able to directly access the reasoning service too.

In order to build the reasoning system, PIRAmIDE needs a common data model, which needs to be:

- *Scalable*, allowing dynamic updates of the knowledgebase to include new context sources.
- *Flexible*, allowing standard access from different components in order to associate context information at different levels of abstraction.
- *Syntactic and semantically explicit and formal*, facilitating consistency checking when including new entities and concepts.
- *Sharable and reusable* among different types of systems, and prepared to support future distributed reasoning processes.
- *Light*, to be easily managed by resource-constrained mobile devices, ensuring limited reasoning response times.
- *Particularizable/extensible for different knowledge domains*, as PIRAmIDE is intended to support a wide range of heterogeneous applications. In particular, the *eHealth*/*eWellbeing* and the *eInclusion* reference domains have been chosen to bind the system validation.

On this data model, to be implemented by using an ontological approach (Section IV.B), a light reasoner will offer its functionalities to PIRAmIDE's components. The reasoner is required to offer:

- *Ontology model support*. The reasoner must be able to infer new context information from the constructions and expressions defined according the common ontological data model.
- *Rule-based reasoning support*. The reasoner must allow managing different sets of rules and apply them to the knowledgebase.
- *Query management*. Different sets of queries, to be answered by the reasoner about the inferred information, should be managed.
- *Standard-oriented*. The ontology model, the rule set and the queries need to be defined when possible by using standard formats in order to ensure 'shareability' and reusability.
- In the same way as the data model, the reasoner must be *light enough* to be deployed in resource-constrained mobile devices, ensuring bounded response times.

B. *General overview of PIRAmIDE's data model*

Taking into account the requirements above, PIRAmIDE's data model is being implemented by means of ontologies, as this approach is assumed to result in a versatile structure in terms of distribution, validation, formalization and completeness [16]. First of all, a general ontology considering common concepts has been defined. This ontology is to be extended with a number of sub-ontologies in order to cope with the particular aspects for each domain of application that PIRAmIDE could consider.

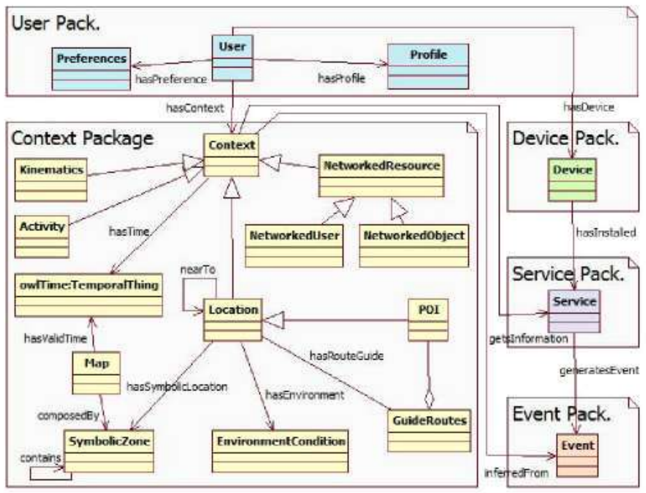

Figure 2. Classes and relations in *Context Package*

The general ontology aims at defining its main concepts to support context inference and persistence processes. Initially, five packages of classes are defined: *User*, *Device*, *Context*, *Service* and *Event*. Figure 2 highlights the classes and relationships modeled in the *Context Package* and the relationships among some of the most important concepts in other packages:

1) The *user package* considers explicit and non-dynamic characteristics of the user. For example, personal data (*userName*, *userBirthDate*, *userGender*, etc.), profile information (e.g. including disabilities) and preferences are included in this package. Generally, these data are manually entered into the system, directly through the user or a system administrator.

2) The *context package* models different features defining the situation of the user. This information is extracted from in-device or environment sensors, and offered to the applications over PIRAmIDE through different services. This package includes concepts such as *Location*, *Environmental Conditions*, user *Activity* or near *Networked Resources*.

3) The *device package* specifies particular features describing the user's mobile device. They include both, software (*operatingSystem*, *audioPlayerFormat*, etc.) and hardware features (*totalMemory*, *keyboardType*, etc.). The list of available services and device sensors is also modeled here.

4) The *service package* mainly defines the attributes characterizing the structure of PIRAmIDE's services, that is: the context information they offer and how to access this information.

5) The information offered by PIRAmIDE's enablers is usually modeled as events. Different types of events are modeled in the *Event package* (e.g. calendar appointments, points of interest, networked resources, etc.).

C. *Adapting PIRAmIDE's data model to a lightweight infrastructure*

OWL –a W3C standard language widely used in data modeling– has been chosen to implement the ontology. It has three increasing expressive sublanguages: OWL-Lite, OWL-DL, and OWL-Full. Following the analysis in [17], we have opted for OWL-Lite as the language to develop PIRAmIDE's ontology. OWL-Lite supports a classification hierarchy and simple constraints; classes and properties can be defined as equivalent, making possible schema-matching and ontology alignment. In addition, OWL-Lite allows properties to be made optional or required. Obviously, OWL-Lite is less complex than OWL-DL, a fact that can have a positive impact on the efficiency of reasoners [17].

However, in the initial stages of the ontology (and sub-ontologies) design, we aimed at fulfilling OWL-DL expressiveness, as the data model could also be used in a centralized infrastructure environment. OWL-DL is an extension to OWL-Lite, a subset of OWL that has computational completeness and decidability (which means that all computations are guaranteed to be computable within finite time). In PIRAmIDE, the initial OWL-DL models were transformed to accomplish OWL-Lite features. To the best of our knowledge, no automatic software tool exists that automatically performs this OWL-DL to OWL-Lite conversion, so it had to be done manually. In this process, some logical assertions, those available in OWL-DL but not in OWL-Lite, had to be removed (e.g. multiple cardinality restrictions), but not the main relations between concepts (and their attributes and data types constraints). Protegé 3.4.4 utility was used to develop every ontology, and also to verify that the resulting ontologies accomplished OWL-Lite expressiveness level.

Although OWL-Lite does support importing third parties' ontologies, external models available in the OWL-DL developments were removed in the OWL-Lite version, once more to free the mobile device from extra processing tasks. This issue will be revisited in future extensions of the 'lite ontology', after individually assessing if it is reasonable to process each of the external ontologies in the mobile infrastructure (regarding its size, expressiveness, etc.).

V. An Embeddable Infrastructure to Enable Dynamic Reasoning

This Section describes our proposal for a reasoning architecture (an ontological manager and an ontological and rule-based reasoner) ready to be integrated into the framework PIRAmIDE. The architecture needs two integration elements to be implemented: 1) first, it is necessary to develop a new enabler, which will encapsulate the reasoner itself (*Inference Enabler* in Figure 3); 2) next, each new enabler willing to use the reasoning services needs to include an *Ontology Manager* module in its own structure in order to be able to communicate with the *Inference Enabler* (Figure 3 shows a *Generic Enabler* integrating this module).

Our particular development uses *Bossam* [12], a rule engine extended to support OWL and SWRL reasoning capabilities, as the ontology (and rule-based) reasoner wrapped inside *Inference Enabler*. *µJena* [5], an ontology management API for mobile clients, has been used in order to manage PIRAmIDE's ontologies inside those enablers needing to use reasoning services.

PIRAmIDE's light data model is used as the common language for PIRAmIDE's components to exchange information. In this sense, the *Ontology Manager* needs to exchange three documents with the *Inference Enabler* in order to feed the reasoning system: 1) an OWL document containing the knowledgebase subset to reason over, 2) the set of rules to apply in the reasoning process and 3) a specification of the queries the reasoner needs to answer.

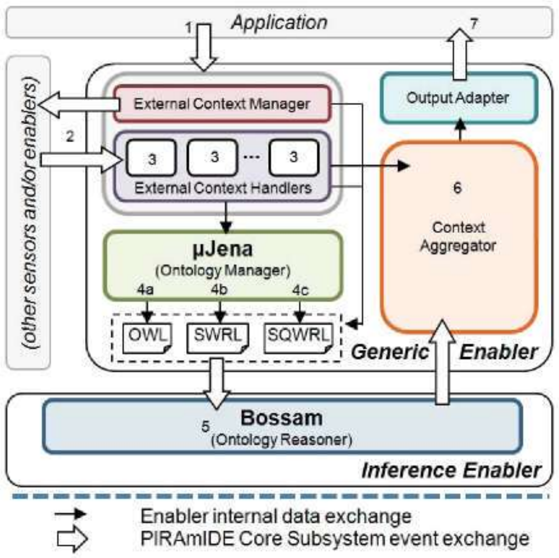

Figure 3. Software structures and information flow for building intelligent services based on ontological reasoning over PIRAmIDE

The use of the reasoning system is explained throughout the following data flow, which starts when an application asks the framework PIRAmIDE for some context information to be provided by a Generic Enabler through the use of sensors and other services in the framework:

1) An application demands information on a context parameter: the framework PIRAmIDE dynamically discovers the enabler providing this information (*Generic Enabler* in Figure 3) and starts and configures it to fulfill the application requirements.

2) Obtaining external context data: the *Generic Enabler* may need information coming from other enablers or sensors; it can then subscribe or query other PIRAmIDE components in order to get this information (by using its *External Context Manager*).

3) Handling different types of context parameters: each type of context data received from sensors and/or enablers will be managed by different *External Context Handlers*. Context data may be independently pre-processed here.

4) Preparing context information for reasoning processes: information handled by the *Generic Enabler* needs to be shaped in an ontological way in order to be understood by the *Inference Enabler*. This is performed by the *Ontology Manager*, leading to:

a) Model instantiation: context data acquired from other enablers or sensors need to be instantiated according to the ontology model. *μJena* will generate an OWL-Lite document (in N-TRIPLE serialization format) as a temporal knowledgebase (KB) to feed the reasoner.

b) Rules configuration: if the reasoner needs to be fed with any rule, a new rule ontology (based on SWRL standard) will be also generated at this point. Currently, just static SWRL files are used, as *μJena* does not support this standard yet.

c) Queries definition: finally, it is necessary to state the specific queries the reasoner needs to solve. A popular ontological query language, as SQWRL (or SPARQL, RDQL, etc.) is supposed to be used at this point, but currently a proprietary one, just managed by the reasoner, is actually used.

5) Reasoning: *Bossam* is used as ontology and rule-based reasoner. It reasons about the set of facts obtained from merging both, OWL and SWRL ontologies, returning the answers to the (SQWRL) queries.

6) Context aggregation: the *Generic Enabler* may also need to fuse different context data (some from the reasoning process, some directly coming from other enablers/sensors).

7) Service output adjustment: finally, the *Generic Enabler* output is adapted to meet the application requirements.

To clarify the process previously described, an example of reasoning service is following reviewed.

VI. Validation

Some location-based applications usually need to translate the Cartesian position information (coordinates <*x*,*y*>), provided by localization systems, into symbolic information (e.g. "living room"), describing the zone or area where the user is moving. This process, usually referred as reverse geocoding, may be configured and automated into PIRAmIDE by using its reasoning capabilities, preventing the internal services or applications from independently and redundantly compute the information once and again.

In this case, three different enablers have been implemented (Figure 4): 1) the *Inference Enabler*, as described in previous Section; 2) a *Location Enabler*, that obtains an accurate estimation of the location of a mobile device, by fusing information acquired from different positioning technologies (GPS, WiFi and Bluetooth); and 3) a *Location Mapping Enabler*, which encapsulates the necessary logic to provide the mapping service: subscription to the needed sensors/enablers, ontology model instantiation, rules and queries management, etc. This last enabler follows the architecture defined for the *Generic Enabler* in the previous Section.

Figure 4 shows these three components and the exchanged information flow: 1) the application requests subscription to the *symbolicZone* measurements and PIRAmIDE addresses this subscription towards the service in charge of that kind of measure: the *Location Mapping Enabler*; 2a) this enabler knows it needs to settle a subscription to the *Location Enabler*; 2b) at the same time, the *Inference Enabler* is initially configured (in this particular case, the *Location Mapping Enabler* already knows the SWRL rules and –SQWRL– queries to apply); 3) *Location Enabler* starts generating events, that PIRAmIDE will address to the subscribed components; 4) *Location Mapping Enabler* updates the OWL KB according to the information received from the *Location Enabler;* 5) the updated KB is used, together with the set of rules and queries, in order to determine the symbolic position of the user; and 6) the *Location Mapping Enabler* finally offers the user's symbolic location to the application.

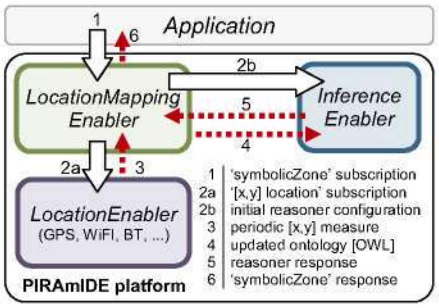
Application
1
6
2b
LocationMapping Enabler
Inference Enabler
5
4
2a
3
LocationEnabler (GPS, WiFI, BT, ...)
PIRAmIDE platform
1 'symbolicZone' subscription
2a '[x,y] location' subscription
2b initial reasoner configuration
3 periodic [x,y] measure
4 updated ontology [OWL]
5 reasoner response
6 'symbolicZone' response

Figure 4. Cartesian to symbolic location mapping service overview.

Figure 5 partially shows the data handled by the reasoner (and those provided by the mapping service). In particular it shows: the model instantiation of a number of symbolic zones (rectangles defined by opposite corners; Figure 5.1) and the Cartesian position of the user (Figure 5.2), the rule applied to infer the symbolic location (Figure 5.3) and the final result of the inference process (Figure 5.4).

```
    <SymbolicZone rdf:ID="zoneA">
        <x1 rdf:datatype="&xsd;int">23</x1>
        <y1 rdf:datatype="&xsd;int">27</y1>
1)      <x2 rdf:datatype="&xsd;int">43</x2>
        <y2 rdf:datatype="&xsd;int">54</y2>
    </SymbolicZone>
    <SymbolicZone rdf:ID="zoneB">[...]</SymbolicZone>
    <SymbolicZone rdf:ID="zoneC">[...]</SymbolicZone>
    [...]
                                 +
    <Location rdf:ID="userLoc">
2)      <x rdf:datatype="&xsd;int">31</x>
        <y rdf:datatype="&xsd;int">42</y>
    </Location>
                                 +
    Location(?p),SymbolicZone(?z),x(?p,?px),x1(?z,?zx1),x2(
3)  ?z,?zx2),y(?p,?py),y1(?z,?zy1),y2(?z,?zy2),greaterThan(
    ?px,?zx1),greaterThan(?py,?zy1),lessThan(?px,?zx2),less
    Than(?py,?zy2) -> hasSymbolicLocation(?p, ?z)
                                 ⇩
    <Location rdf:ID="userLoc">
4)      <hasSymbolicLocation rdf:resource="#zoneA"/>
    </Location>
```

Figure 5. Excerpt of the ontology data exchanged between the *Location Mapping Enabler* and the *Inference Enabler.*

VII. Discussion and Future Work

This paper presents a proposal for a mobile reasoning system, ready to be developed and integrated in a service-oriented framework for context management. The system is built on available tools for data model management and ontology rule-based reasoning, which have been adapted to work properly.

Following, we comment on some relevant aspects of the design and development experience. We aimed at defining a reasoning system: 1) capable of dealing with ontology data, 2) using standard formats and 3) working in resource-constrained mobile devices. On the one hand, nowadays it is easy to find implementations just satisfying requirements #1 and #2 (Jena, OWLAPI, Pellet, HermiT, etc.) but they are not really developed to be deployed in any kind of mobile device (we even tried to do so with most of them, without success). On the other hand, state-of-the-art analysis reveals a (limited) number of developments just meeting #3 prerequisite. So, as far as we are concerned, *μJena* (as ontology manager) and *Bossam* (as ontology and rule-based reasoner) are the only ones satisfying all three key requirements. Nevertheless, their integration inside the PIRAmIDE framework has not been smooth.

It is important to note that both *μJena* and *Bossam* are research developments and cannot be taken as fully-operative systems. One of their main drawbacks is their lack of integration: most of the infrastructure ontology reasoners (Pellet, HermiT, etc.) are directly coupled with Jena and/or OWLAPI in order to manage the models they reason over. However, *Bossam* just can load data models from the file system. This limitation breaks the process of updating the knowledgebase before inferring new information: in our particular implementation, *μJena* and *Bossam* communicates between them by exchanging documents stored in the mobile file system (see Figure 3.4), an error prone practice that definitely increases time responses. This is, from our point of view, *Bossam*'s main drawback.

Moreover, *μJena* lacks flexibility regarding the supported standard formats. Although future extensions are planned, at present it just can read OWL ontologies coded in N-TRIPLE syntax. No report about the performance of this syntax in mobile devices has been found, but updated references regarding this issue ([6], [7]) point out to Manchester or KRSS syntax as some of the most appropriate for resource-constrained devices. In addition, *μJena* lacks support to any rule or query language standard; besides, it does not allow external ontologies to be imported.

There still exists few developments capable of dealing with ontology management and reasoning in mobile devices, and the existing ones are still far from maturity. From our point of view there is a need of a common ontology management strategy (as Jena or OWLAPI for infrastructure environments) to be integrated in mobile reasoners. It is also worth mentioning that, as far as we are concerned, no one of the 'well-known' ontological reasoners (Pellet, FaCT++, RacerPro, etc.) are planned to be migrated to mobile environments.

Regarding the development of our PIRAmIDE light framework for context management, we are currently testing the feasibility of using the *Inference Enabler* simultaneously from different *Service Enablers*. Performance tests to evaluate the computational load of the reasoning solution will be done. We are also starting to face some of the main limitations of the employed tools, mainly the file system dependency and the dynamic SWRL and SQWRL management support.

ACKNOWLEDGEMENTS

This work has been supported by the Spanish Ministry of Industry, Tourism and Commerce and the European Fund for Regional Development through the Avanza I+D subprogramme, under grant TSI020301-2008-2. The authors also acknowledge enriching discussions with other partners of the PIRAmIDE consortium.

REFERENCES

[1] M. Baldauf, S. Dustdar, F. Rosenberg, "A Survey on Context-Aware systems", International Journal of Ad Hoc and Ubiquitous Computing, 2007, Vol. 2, Is. 4, pp. 263-277.

[2] M. Perttunen, J. Riekki, O. Lassila, "Context representation and reasoning in pervasive computing: a review", International Journal of Multimedia and Ubiquitous Engineering, 2009, 4(4):1-28.

[3] T. Gu, H.K. Pung, D.Q. Zhang, "Toward an OSGi-based infrastructure for context-aware applications", IEEE Pervasive Computing, 2004, Vol. 3, pp. 66-74.

[4] P. Fahy, S. Clarke "CASS – a middleware for mobile context-aware applications", 2004, Procs. of the Workshop on Context Awareness, Mobisys, 2004.

[5] F. Crivellaro, "μJena: Gestione di ontologie sui dispositivi mobili", MsC Thesis, 2007, Politecnico di Milano.

[6] M. Koziuk, J. Domaszewicz, R.O. Schoeneich, M. Jablonowski, P. Boetzel, "Mobile Context-Addressable Messaging with DL-Lite Domain Model", LCNS Smart Sensing and Context, 2008, Vol. 5279/2008, pp. 168-181.

[7] T. Kleemann, A. Sinner, "Description logic based matchmaking on mobile devices". Procs. 1st Workshop on Knowledge Engineering and Software Engineering, 2005, pp. 37-48.

[8] A. Sinner, T. Kleeman, "KRHyper - In Your Pocket", LCNS Automated Deduction - CADE-20, 2005, Vol. 3632/2005, pp. 452-457.

[9] T. Gu, Z. Kwok, K.K. Koh, H.K. Pung, "A Mobile Framework Supporting Ontology Processing and Reasoning", Proc. of the 2nd Workshop on Requirements and Solutions for Pervasive Software Infrastructures, 2007.

[10] S. Ali, S. Kiefer, "μOR – A micro owl description logic reasoner for ambient intelligent devices", LCNS - Advances in Grid and Pervasive Computing, 2009, Volume 5529/2009, pp. 305-316.

[11] J.I. Vázquez. "A reactive behavioural model for context-aware semantic devices. PhD Thesis, 2007, Univesidad de Deusto, Bilbao, Spain.

[12] M. Jang, J-C Sohn, "Bossam: An Extended Rule Engine for OWL Inferencing", LCNS Rules and Rule Markup Languages for the Semantic Web, 2004, Vol. 3323/2004, pp. 128-138.

[13] G. Specht, T. Weithoner, "Context-Aware Processing of Ontologies in Mobile Environments", Procs. of the IEEE Int. Conf. on Mobile Data Management, 2006, IEEE Computer Society, pp. 86.

[14] H. Myrhaug, N. Whitehead, A. Goker, T. E. Faegri, T.C. Lech "AmbieSense – A System and Reference Architecture for Personalised Context-Sensitive Information Services for Mobile Users", LCNS Ambient Intelligence, 2006, Vol. 3295/2004, pp. 327-338.

[15] OASIS Standard, "Reference Model for Service Oriented Architecture 1.0. 2006.

[16] A-C. Boury-Brisset, "Ontology-based Approach for Information Fusion", Proc. of the 6th Int. Conference on Information Fusion, 2003, 522 – 529.

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