Design and validation of a light inference system to support embedded context reasoning

Josué Iglesias · Ana M. Bernardos · Paula Tarrío · José R. Casar · Henar Martín

Abstract Embedded context management in resource-constrained devices (e.g. mobile phones, autonomous sensors or smart objects) imposes special requirements in terms of lightness for data modelling and reasoning. In this paper, we explore the state-of-the-art on data representation and reasoning tools for embedded mobile reasoning and propose a light inference system (LIS) aiming at simplifying embedded inference processes offering a set of functionalities to avoid redundancy in context management operations. The system is part of a service-oriented mobile software framework, conceived to facilitate the creation of context-aware applications—it decouples sensor data acquisition and context processing from the application logic. LIS, composed of several modules, encapsulates existing lightweight tools for ontology data management and rule-based reasoning, and it is ready to run on Java-enabled handheld devices. Data management and reasoning processes are designed to handle a general ontology that enables communication among framework components. Both the applications running on top of the framework and the framework components themselves can configure the rule and query sets in order to retrieve the information they need from LIS. In order to test LIS features in a real application scenario, an 'Activity Monitor' has been designed and implemented: a personal health-persuasive application that provides feedback on the user's lifestyle, combining data from physical and virtual sensors. In this case of use, LIS is used to timely evaluate the user's activity level, to decide on the convenience of triggering notifications and to determine the best interface or channel to deliver these context-aware alerts.

Keywords Context-aware application · Data modelling · Light ontology management · Embedded reasoning · Service-oriented architectures · Activity inference

This paper is an extension of the work entitled 'A light reasoning infrastructure to enable context-aware mobile applications' presented in the third International Workshop on Sensor Networks and Ambient Intelligence (SeNAmI 2010), Hong Kong, China, 2010.

J. Iglesias (✉) · A. M. Bernardos · P. Tarrío · J. R. Casar · H. Martín
Universidad Politécnica de Madrid, Av. Complutense 30, 28040 Madrid, Spain
e-mail: josue@grpss.ssr.upm.es

A. M. Bernardos
e-mail: abernardos@grpss.ssr.upm.es

P. Tarrío
e-mail: paula@grpss.ssr.upm.es

J. R. Casar
e-mail: jramon@grpss.ssr.upm.es

H. Martín
e-mail: hmartin@grpss.ssr.upm.es

1 Introduction

Acquiring, processing, merging and disseminating data coming from heterogeneous sensors in order to infer information about a target entity's situation and intent (usually referred as *context information* in scientific literature since mid-90s) may be a very costly and complex process. To alleviate the difficulty of dealing with context management tasks and facilitate the development of context-aware applications, a good number of frameworks have been proposed to date (e.g. see [1, 2] for a survey). Most of them rely on infrastructure-based centralized architectures (e.g. [3–6]), which allow powerful processing and reasoning. This centralized approach requires permanent communication between context-data sources and

context-data consumers. Data losses or delays in the information flow may cause distorting effects on the application side due to information misalignment. Moreover, this approach may require complex security techniques as context data are handled in external infrastructures, and security implementation becomes significantly important when managing context information (as it includes highly private data such as location, identity, activity or social network information [7]).

At the same time that new context-aware architectures and reasoning tools have evolved, mobile devices have dramatically improved their communications, processing and sensing capabilities. Mobile applications are nowadays increasingly adapted to user's profile and needs, as they can take advantage of the deep knowledge about the individual that it is possible to gather directly from the mobile device.

The process of extracting information to personalize a mobile service may be transferred from the application side to common 'core' device modules. Hence, a personal device may establish a univocal relationship with the user, performing an accurate and continuous analysis of the user's behaviour and context (from communication data exchange, sensor data, contacts, agenda, information requests or interactions), which it may internally use or expose for other applications to exploit. This embedded personalization capability may be used, for example, to adapt any kind of interface in the mobile device or to serve as a commodity layer to develop diverse behaviour-based applications (such as personal trainers, navigation tools taking into account the user's driving skills, family managers helping to schedule daily life or ambient care systems enhancing the social network of the elder).

Context-aware/behaviour-based embedded applications make an intensive use of sensor data, needing to accomplish in-device complex reasoning processes to efficiently analyse the collected information. Thus, embedded architectures to manage context become important to deal with acquisition, fusion and reasoning tasks. This device-centric approach for context management enables the user to control the personal information shared with external systems and services, at the same time that the application performance and the final user experience may benefit from the absence of communication delays or interruptions.

Nowadays, mobile operating systems are providing easily accessible interfaces to mobile devices embedded sensors, but context-aware reasoning-oriented solutions to deal with context data and avoid processing redundancy are still pending. Although there are some proposals coping with this issue, most of them are application oriented, not being suitable for scaling or changing the application domain.

In this paper, we address the design and integration of a light inference system (LIS). This system has been designed to be part of an embedded service-oriented mobile framework offering a complete set of context management features (acquisition, data fusion and context reasoning), ready to be directly consumed and customized from the developer side. LIS is built on existing tools for data modelling and reasoning, which have been adapted and integrated to work coordinately. The result is an embeddable software module offering domain-agnostic rule-based reasoning and data model managing capabilities, which may be configured and used either from the context-aware framework itself or directly by context-aware applications. As a case of use relying on the designed architecture, LIS is used in this work to facilitate the deployment of a context-aware behaviour-based application, an Activity Monitor, which aims at persuading the user to increase his/her physical activity all day long, by establishing well-defined goals related to the user's lifestyle.

The paper is structured as follows. Section 2 reviews and compares existing embedded toolkits for data model management and reasoning. Section 3 briefly presents the embedded context-aware framework supporting the light inference system, including the data model, which serves to support the communication between software modules. From a set of operational requirements, LIS is fully described in Sect. 4. Section 5 explains how the reasoning needs of the Activity Monitor are fulfilled by using LIS. Section 6 concludes the paper with an analysis of the drawbacks and key issues identified when designing, building and testing the reasoning system.

2 Related work on lightweight data representation and reasoning

Relying on a restricted use of communications, device-embedded strategies for context management may deliver relevant functional benefits, such as personal data securization and control and efficient data handling and processing, thus offering an acceptable performance from the user's point of view in terms of Quality of Service. Nevertheless, this approach implies redesigning context management processes to adapt them to adequately work in resource-constrained devices. In particular, centralized context-aware frameworks may integrate standard reasoners—e.g. Pellet, HermiT—together with data model management tools to automate the inference process of complex context information on a knowledgebase, an approach to consider also in mobile computing environments. To date, how to face this capability in resource-constrained devices has been explored in several proposals. Following, a state-of-the-art on lightweight tools to manage data models and a review on strategies to integrate automated reasoning on context instances are addressed.

From a functional point of view, machine-processable descriptions are necessary to organize, valorize and share the huge amount of context information nowadays generated by information systems [8]. Sensor data, personal digital information, linked open data, etc., may be jointly processed if they are organized under a common and expressive data structure. Information representation techniques differ depending on the area of knowledge and, mainly, on the practical requirements of the final application itself. How to optimally represent information in resource-constrained mobile devices is still an open challenge, and several approaches for light context information modelling have been addressed in the literature (e.g. [1, 9, 10]). For example, the simplicity of tuple-based models (also known as key value models) reduces management overhead and enables them to be applied to already existent mobile systems (e.g. [11, 12]), but they lack from validation and scalability capabilities and they are not suitable for handling the typical ambiguity of context information. The hierarchical structure and automatic validation are some of the main strong points of markup scheme modelling. These approaches are usually based on Standard Generic Markup Language (SGML), being XML the most popular. XML has a high semantic redundancy, and it is not adapted to the limited resources of embedded devices; according to [13] a better performance, saving computing resources and bandwidth can be obtained with other techniques as, e.g., JavaScript Object Notation (JSON) [14]. A lightweight XML-based approach for modelling context information and for encoding context management messages is presented in [15]; it enables fast processing on resource-constrained mobile devices but lacks semantic expressiveness.

Ontology-based modelling [16] combines the advantages of object-based and logic models [17], i.e., encapsulation, extendibility and reusability, and formalism and inference capabilities, respectively. Ontology-based data models facilitate information fusion from heterogeneous data and knowledge sources, also providing support for automated reasoning [18]. OWL (Web Ontology Language) [19], the standard ontology language endorsed by the W3C, enables different applications/tasks to share a common model [20], providing common shared domain vocabularies and a consistent mechanism for information representation. According to [9], these features are particularly important in mobile and pervasive environments, in which different heterogeneous and distributed entities must interact for exchanging users' context information. OWL needs a particular syntax in order to store and exchange ontologies among applications. RDF/XML [21] is widely used; it is a 'heavy-weight' representation but many methods and tools for lighten XML have been proposed (mainly based on compression methods or binary XML formats [22]). Besides RDF/XML, there exist several other syntaxes tailored to the features of resource-constrained devices. Manchester OWL [23] is a natural-language compact syntax for OWL ontologies, more readable and understandable by common human users; according to [24], ontology files encoded in Manchester OWL are easier to parse and approximately twice smaller than when encoded in the RDF/XML syntax. KRSS2 (Knowledge Representation System Specification 2), an extension of KRSS [25], is another easy-to-parse syntax for DL-based knowledge representation; it is used in [26] to formalize several ontologies used to perform embedded reasoning. N-Triples [27] is a line-based, plain text serialization format for ontologies; it was designed to be a fixed subset of N3 syntax (and hence of Turtle) and therefore easier for software to parse and generate; however, it lacks some of the shortcuts provided by other RDF serializations.

Formal information representation facilitates automated reasoning (e.g. model and knowledgebase consistency checking, concept and instance classification [28]). Although scarce compared with general context management systems, some light tools enabling reasoning in resource-constrained devices have already been described in the literature (see Table 1 for a comparison); most of them work on ontology-based data models. Crivellaro et al. [29] developed *μJena* to manage ontologies stored in mobile devices (this tool does not provide ontological reasoning but only an API to create, delete, edit, etc. ontology concepts, relations, etc.). *LOnt* [24] is another custom implementation of the Jena API [30] for mobile devices. Its main features are its small size and low-memory fingerprint, which make it suitable for use in J2ME mobile devices. Kleeman et al. [7] integrated the mobile *Pocket KRHyper* reasoner [31] for profile management and decision-making tasks. *Pocket KRHyper* is a reasoning system for Java-enabled mobile devices; it does not offer support for any of the standard ontology languages (OWL, SWRL [32]—Semantic Web Rule Language—etc.). In [33], a mobile framework providing ontology processing and reasoning was proposed. The reasoning engine contains 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 works on a subset of OWL ontology inference rules. In this framework, a lightweight RDQL (RDF Data Query Language) query engine supporting a subset of RDQL syntax is also developed. The *μOR* reasoner [34] is introduced as a part of a framework for developing AmI-based medical devices. It makes reasoning over a subset of OWL-Lite entailments. Vázquez [35] 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.

Table 1 Summary of embedded semantic tools

	Reasoning capabilities	Semantic expressivity	Syntax	Language for rules and queries	Platform	Licence
µJena [29]	No (just ontology management)	OWL-Full	N-Triples	–	J2ME CLDC/ MIDP	Proprietary licence (Jena) + GNU GPL (µJena extension)
LOnt [24]	DL (lite)	OWL-DL	Manchester OWL [23]	–	J2ME	–
Pocket KRHyper [31]	FOL (theorem prover)	FOL	KRSS [25]	–	J2ME CLDC/ MIDP	GNU LGPL
[33]	DL (subset)	Subset of OWL-DL	RDF/XML	RDQL	J2ME CLDC/ MIDP	–
µOR [34]	DL (lite subset)	Subset of OWL-Lite	Extended N-Triples	Subset of SPARQL [53]	J2ME CDC	–
MiniOWL-MiniRule [35]	DL (lite subset)	Subset of OWL-Lite	Proprietary	–	–	–
Bossam [36]	DL + rule-based	OWL-DL	RDF/XML, OWL/XML [54], N-Triples	SWRL, SQWRL [55], RuleML, Buchingae (proprietary)	J2ME CDC/ PP	Proprietary licence, closed source

DL, description logic; *FOL*, first-order logic

Finally, *Bossam* [36] has native support for reasoning over OWL/SWRL ontologies and RuleML rules [37]. Its runtime size is about 750 Kb, running on J2ME CDC/PP platforms as well as on J2SE platform of JDK 1.3 or later. At this point, it is worth mentioning *androjena,*[1] a new development (the first version was released on May 2010) based on a subset of the popular Jena framework, migrated to Android platforms. To the best of our knowledge, up to now, it has only been used in few works (e.g. [38, 39]) to support information management and reasoning in embedded privacy protection and mobile social network developments.

In addition, some hybrid architectures combining server-based and device-oriented approaches have also been explored. *MobileOntoDB* [40] evaluates each query performed in the device, and if it exceeds the device capabilities, it is sent to a central reasoning server. A distributed case-based reasoning mechanism is used in *AmbieSense* [41], an agent-based infrastructure for context-based information delivery for mobile users, in which online reasoning (most simple and common processes) lies on the user's mobile device while offline reasoning (processes requiring more resources but without response time constrains) is done in the backbone system.

The state-of-the-art analysis reveals two approaches when embedding ontology model management and reasoning capabilities. On the one hand, most of the current developments are focused on tailoring representation and reasoning techniques to solve specific problems (e.g. [42, 43, 44]). The work presented in [45] is particularly relevant at this point, as it presents an OWL ontology reasoner that is dynamically and automatically composed to provide only the level of reasoning required for the ontology in use. On the other hand, recent improvements in the performance of mobile devices (with higher processing capabilities, available memory, etc.) enable lightweight general-purpose ontological tools to be embedded with a promising success (e.g. [29, 36, 39]). However, there are still few developments in this direction, and the existing ones are still far away from maturity. Although some performance tests can be found in the literature (e.g. for µJena [29], Bossam [36] or, more recently, androjena [39]), there is a lack of experiments comparing their performances in a common scenario. Finally, it should be noted that much of these developments are discontinuous research projects so, not only the performance has to be taken into account when making a choice, but also its public availability, standard compatibility, maintenance or planned future extensions.

In this paper, we propose the design of a general-purpose light inference system (LIS) aiming at offering ontology model support, rule-based reasoning and multiple

[1] http://www.code.google.com/p/androjena/.

query management. For interoperability and reusability purposes, we target the design of a standard-oriented reasoning proposal, in terms of languages used for ontology modelling, rule set and queries. As none of the previously analysed tools fulfil all these requirements, we aim at adapting and combining a selection of them to design our own solution.

3 A service-oriented mobile framework to manage context data

The light inference system described in this paper is part of a light framework that 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. It is out of the scope of the paper to detail how the framework has been designed and built, but Sect. 3.1 gathers a general view of its software architecture in order to contextualize the following description of the inference system (Sect. 4). For the same reason, Sect. 3.2 describes the underlying data model handled by LIS to support the reasoning tasks.

3.1 General description of the framework software architecture

The framework is conceived to provide easy access to data coming from a number of sensors, either embedded in the mobile device (such as accelerometers, gyroscopes, RFID/NFC interfaces or cameras) or deployed within the user's environment (e.g. wireless nodes with sensors or bidimensional codes). 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 modules, the framework offers a set of general and application-independent facilities to deal with context processing, offered as *horizontal services* to be used by any application deployed on top of it. To date, horizontal services provide, for example, the following: (1) seamless position estimation (handling handovers from outdoor GPS to WiFi, ZigBee and Bluetooth indoor localization systems), (2) detection of tagged Points of Interest (which can be dynamically discovered by using wireless technologies or by previous registration of their location in the PoI database), (3) image-based decoding of bidimensional codes and (4) ontology rule-based reasoning. This last capability is the one addressed in this paper through the design of LIS. From its functional perspective, LIS is thought to be a flexible and configurable tool to outsource periodic rule-based reasoning tasks from applications or internal framework components, reducing the coding complexity and the computational cost in runtime.

In brief, the framework requirements include the following:

- Encapsulation of embedded and external sensors, offering common interfaces to access the data they offer.
- Offering different acquisition methods (synchronous, asynchronous or on-demand) in order to adapt sensor data retrieval to consumer applications needs.
- Managing a dynamic registry of available sensors, and its associated measurements, in order to support discovery and subscription processes.
- Providing a set of horizontal services (in charge of processing the data coming from sensors or from other services) that are encapsulated as a special type of sensor, sharing a common registration method and access mechanisms.

In order to clarify how LIS has been integrated into the framework, a brief description of its software architecture is presented next. The framework is implemented on a service-oriented software architecture [46] composed by three main building blocks: *Sensing Subsystem*, *Context Management Subsystem* and *Core Subsystem* (Fig. 1). The *Sensing Subsystem* decouples access to embedded and external sensors from upper processing levels by wrapping sensor-specific characteristics inside software units, which deal with low-level hardware information retrieval. The *Context Management Subsystem* is composed of a number of modules that process data coming from sensors (or from other modules), fuse them and infer complex context parameters (offered as horizontal services). Finally, the *Core Subsystem* provides several features to integrate software modules into the framework, such as discovery and registry management of new elements and some common utility libraries. Internal modules and applications use an (asynchronous) event-based communication strategy, leaded by the *Core Subsystem*, in order to exchange context data. Applications consume context information provided by modules using the features provided by the *Core Subsystem*.

The resulting approach has been developed using the service-oriented mobile OSGi programming platform. mOSGi handles modules referred as bundles (*enablers*): cohesive, self-contained units, which explicitly define their dependencies to other modules/services and their external APIs. For example, LIS relies on two enablers referred in Fig. 1 as *Ontology Model Manager* and *Inference Engine*.

3.2 Description of a light data model for context management

As previously stated, the framework needs to handle a generic data model, built to enable communication among the horizontal services and with the applications on top of

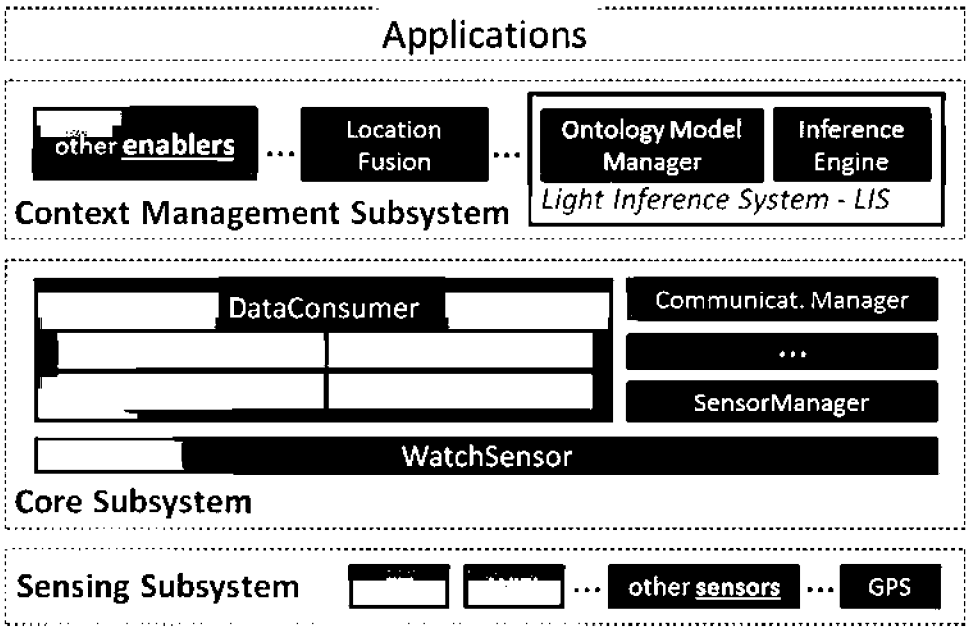

Fig. 1 Architecture of the mobile service-oriented framework supporting the light inference system

it. This data model will also be shared between LIS and the framework components invoking its reasoning services. It has been designed to be: *light* to be easily managed by mobile devices; *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; and *adaptable/extensible to different knowledge domains*, as it is intended to support a wide range of heterogeneous applications.

The data model has been implemented by means of ontologies, as this approach results in a versatile structure in terms of distribution, validation, formalization and completeness [18]; it has been built taking into account the design principles in [47].

Figure 2 shows an overview of the core ontology used to build the data model, considering common concepts on which the framework works. This first version of the ontology considers general concepts common to embedded context-aware deployments, being extensible by using domain ontologies in order to cope with the particular aspects of different application requirements.

Initially, five common packages of classes have been included in the core ontology: *User*, *Device*, *Context*, *Service* and *Event*. Figure 2 highlights the classes and relationships modelled in the *Context package* and the relationships among some of the most important concepts in other packages:

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 by the user or a system administrator.

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

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 modelled here.

The *Service package* mainly defines the attributes characterizing the structure of the horizontal services, i.e., the context information they offer and how to access this information.

The information provided by the enablers is usually modelled as events. Different types of events are modelled in the *Event package* (e.g. calendar appointments, points of interest and networked resources).

Following the analysis in [19], we have opted for OWL-Lite as the language to develop a light ontology suitable to work in the mobile device. OWL-Lite supports a classification hierarchy and simple constraints; classes and

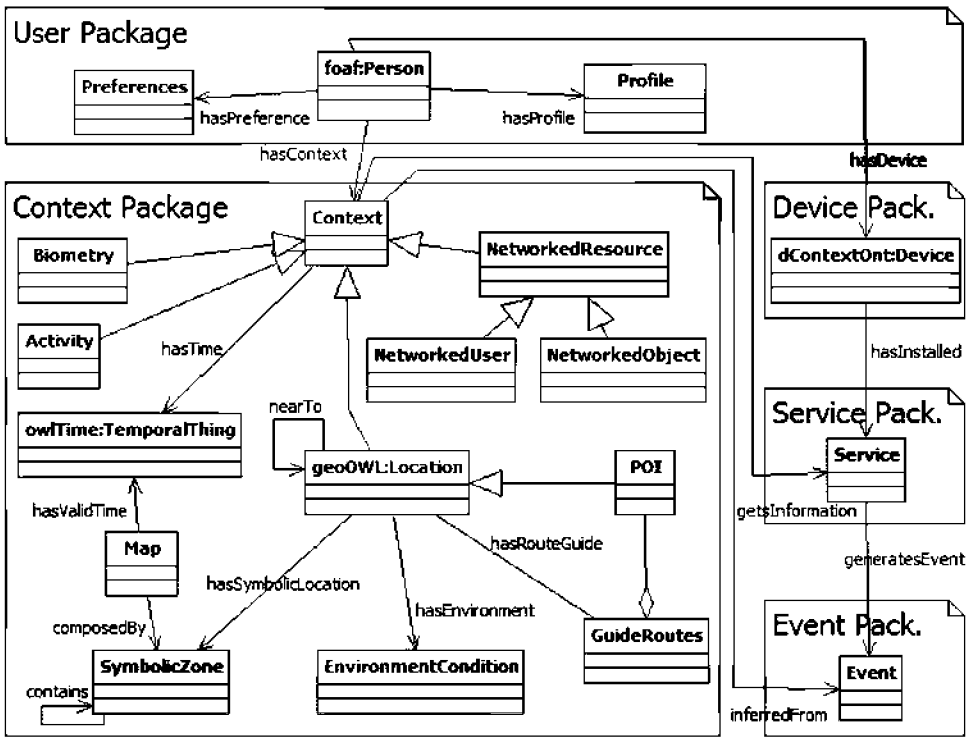

Fig. 2 Overview of the framework ontology (highlighting *Context package* classes and relations)

properties can be defined as equivalent, making possible schema-matching and ontology alignment. The fact that OWL-Lite is less complex than OWL-DL (offering less language constructs and less expressivity power) may have a positive impact on the reasoner's efficiency, while being sufficient for many real applications [19].

However, in the initial stages of the framework ontology 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 that has computational completeness and decidability (which means that all computations are guaranteed to be computable within finite time). When possible, the entities considered in the OWL-DL data model have been implemented through standard ontologies such as OWL-Time (*Time*), Geo-OWL (*Location*), FOAF (*User*), RDF Cal (*Calendar*) or Delivery Context Ontology (*Device*).

In order to adapt the data model to be processed in a mobile device, the initial OWL-DL model was manually transformed to accomplish OWL-Lite features (to the best of our knowledge, no software tool exists that automatically performs this OWL-DL to OWL-Lite conversion). 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 (v3.4.4) utility was used to develop the framework ontology and also to verify that it 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. The extension of the core ontology with domain-specific ones should be revisited in future works in order to check how these extensions may deteriorate the overall system performance.

4 Design of a light inference system

LIS is conceived as a general-purpose rule-based reasoner with ontology management capabilities, which operates on the data model described in Sect. 3.2. LIS's features need to be accessible both to every component in the framework and to context-aware applications built on top of it, so the inference system has to be versatile enough to be configured to perform fusion tasks on plain data or to manage reasoning on context features. This approach allows customized development strategies (where the programmer decides to which extent the application logic should be retained into the application layer or delegated to the framework).

In order to invoke LIS reasoning service, next elements should be defined in advance:

1. A background ontology modelling a knowledgebase with specific facts to reason over (e.g. ontology data model described in Sect. 3.2).

2. A set of rules/queries used to infer new knowledge from the facts stated in the knowledgebase (these rules/ queries may be dynamically provided and are tightly coupled to the ontology model).
3. One or more information sources (sensors, horizontal services, etc.) updating the knowledgebase (i.e. updating the facts used in the rule/query sets).
4. A consumer service (application or in-framework element), in charge of configuring the rule/query set and invoking the reasoning process.

This version of LIS employs the data model previously presented as a static ontology, but the rule/query sets and the information sources and consumers can be dynamically configured. Note that reasoning service consumers are in charge of feeding LIS with a rule set adapted to obtain the desired information.

This Section details how LIS has been built. It first discusses the selection of the software tools that were finally used to enable the reasoning service and how these tools have been integrated into the service-oriented software architecture; secondly, it describes how data flow when an application or component uses LIS.

4.1 Embedding semantic tools into a light service-oriented framework

LIS is composed by an ontological manager to handle the knowledgebase—shaped according to the data model previously explained—and an ontological and rule-based reasoner.

As previously commented, LIS is conceived to offer a generic problem-solving approach. Thus, most of the tools analysed in the state-of-the-art were discarded to implement LIS features, as they are solutions designed to solve domain-specific problems (despite the general approach of *androjena*, it was not taken into account as it is not fully compatible with our development tools). Among the rest of tools (i.e. *μJena*, *Pocket KRHyper* and *Bossam*), *μJena* has been selected as ontology model and knowledgebase manager as it is the only one with this kind of capabilities, also being capable of dealing with OWL-based ontologies. Regarding the reasoner, both *PocketKRHyper* and *Bossam* are currently publicly available but the fact that the former is not standard oriented led us to choose *Bossam* as rule-based reasoner (it is able to manage most of the well-known ontology standards). In fact, selecting *μJena* led us to choose also *Bossam* as both share a common ontology syntax (N-Triples, see Table 1).

In practice, to implement LIS into the service-oriented framework presented in Sect. 3.1, it has been necessary to design and encapsulate the reasoning tools into two new modules: (1) one enabler in charge of managing the ontological knowledgebase (*Ontology Model Manager* in Fig. 3) and (2) another one encapsulating the reasoner itself (*Inference Engine* enabler in Fig. 3). Thereafter, *μJena* and *Bossam* could be managed as framework *enablers* and may use the *Core Subsystem* capabilities: e.g., they can be registered into the framework, being able to be discovered, configured and invoked by other enablers.

It is important to remark that the lightweight framework also had to be extended in order to be able to host the services offered by *μJena* and *Bossam*. It was initially designed to manage event-based notifications between framework enablers/sensors/applications, but the new reasoning processes required to be invoked on an 'on-demand' basis. Once integrated into the embedded context-aware framework as enablers, *μJena* and *Bossam* had to be tuned up in order to use the same syntax when dealing with the data model and knowledgebase. Although *Bossam* is able to cope with several ontology syntaxes (see Table 1), the only choice was N-Triples, as *μJena* only accepts this format.

With respect to initial versions of LIS [48], the use of separated enablers for data model management and reasoning enhances and simplifies the strategy adopted, as every operation to be performed on the knowledgebase is now centralized in the new *Ontology Model Manager*. Initially, the knowledgebase management (update and retrieval) had to be performed on each enabler willing to invoke the reasoning services, increasing their complexity. However, knowledgebase management is now totally decoupled from the implementation of the logic for a given enabler, also ensuring knowledgebase integrity.

Next Section addresses how LIS is configured and used by consumer modules.

4.2 Data flow in LIS

The operation of the reasoning system is explained throughout the following data flow (Fig. 3). It starts when an application asks the framework for some context information to be provided by a (let's call it) *Generic Enabler* through the use of sensors and other services in the framework:

1. *Context parameter request*: an application demands information on a context parameter, the framework dynamically discovers the enabler providing this information (*Generic Enabler* in Fig. 3) and starts and configures it to fulfil the application requirements.
2. *External context-data retrieval*: the *Generic Enabler* may need information coming from other enablers or sensors; it can then subscribe or query other components in the framework in order to get this information (by using its *External Context Manager*). As shown in Fig. 3, the *Generic Enabler* can access the information

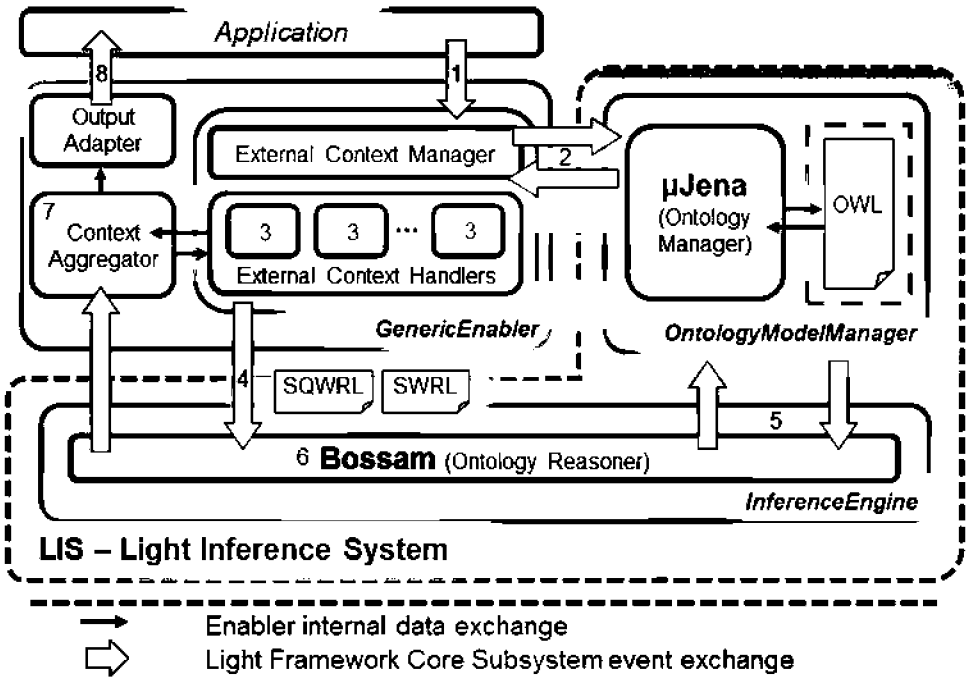

Fig. 3 Software structures and information flow for building intelligent services based on ontological reasoning over the described framework

stored in the knowledgebase using the common interface offered by the *Ontology Model Manager*.

3. *Context parameters pre-processing*: 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. *Reasoner service invocation*: the set of rules (SWRL) and queries (SQWRL) to be applied by the *Inference Engine* enabler need to be configured by the *Generic Enabler*.
5. *Knowledgebase facts retrieval*: *Bossam* needs to know the current status of those variables appearing in the rule and query sets. It uses the common interface of the *Ontology Model Manager* (described below) to access these data.
6. *Reasoning*: *Bossam* is used as ontology and rule base reasoner. It reasons about the set of facts obtained from merging both OWL and SWRL ontologies, returning the answers to the SQWRL queries.
7. *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). For example, time alignment is performed here.
8. *Service output adjustment*: finally, the *Generic Enabler* output is adapted to meet the application requirements (mainly a syntactic adaptation according to the application needs).

In practice, the *Ontology Model Manager* common interface includes general *get()*, *create()*, *set()* and *delete()* operations over the knowledgebase, which may be used by any enabler. In a similar manner, the *Inference Engine* offers a common interface to allow any other enabler to invoke reasoning services; in this case, this interface includes *getters* and *setters* operations managing (1) the set of rules to apply in the reasoning process and (2) the set of queries the reasoner needs to answer. Although *Bossam* is able to deal with standard SWRL rules and SQWRL queries, a *Bossam*'s proprietary format (*Buchingae*) was actually used due to performance requirements (using SWRL/SQWLR led to memory overloads, see Sect. 6 for more details on this).

Section 5 explores the use of LIS capabilities to build a domain application; the data flow between components is analysed for this example in Sect. 5.3.

5 Case of use: reasoning in the Activity Monitor, an application to prevent sedentary lifestyles

The framework described so far, equipped with LIS for ontology model management and inference capabilities, has been used to build the Activity Monitor [49], a context-aware mobile application capable of tracking and evaluating the user's daily movement, in order to provide feedback about his/her level of sedentary behaviour (Fig. 4a shows the Activity Monitor's graphical user interface). The final objective of this application is to deliver adequate context-aware notifications to motivate the user to increase his/her activity level, adapting the exercise goals with respect to his/her physical fitness and health and lifestyle restrictions.

Fig. 4 **a** Application interface delivering an alarm, according to the estimated location and activity of the user [49]. The main interface for feedback is a puzzle of 24 blocks hiding a picture; blocks are unveiled when the user achieves a sufficiently active behaviour. The interface provides information about the estimated type of movement, the user's heart rate and the place where the user is located. **b** External sensors feeding the application: Shimmer motes for movement estimation and Zephyr band with accelerometer, inclinometer and biometric sensors

The Activity Monitor works with measurements taken by sensors (Fig. 4b) both embedded in the mobile device (accelerometers for movement estimation, GPS for outdoors positioning and Bluetooth and WiFi for indoor positioning) and external sensors (wearable inertial sensors for movement estimation and biometric sensors—heart rate, respiration rate, skin temperature and inclinometer—for movement and health status estimation). The application's logic fuses different types of entries to get its best location, movement and subsequent activity estimates.

In particular, the reasoning process that evaluates the user's activity level and decides when and how to deliver notifications regarding the user's lifestyle is fully accomplished through LIS. LIS is used to:

- *Evaluate the user's activity level*: physical movement is quantified and evaluated in different timeframes in order to estimate the most plausible activity.
- *Generate context-aware alarms*: notifications encouraging users to follow healthier lifestyles are generated having into account the following: (1) the application configuration (gathering user's preferences), (2) the time elapsed since last notification and (3) the result of evaluating short- and long-term activity levels.
- *Configure multimodal channels for context-aware feedback delivery*: once a notification is to be generated, LIS is invoked to decide the more suitable interface (in-device alarm—e.g., soundless, vibrating—email notification, social network post, etc.) to provide the alert, according to the user context (e.g. current location or activity).

The Activity Monitor also accomplishes some other low-level reasoning tasks (e.g. user activity detection from accelerometers and biometric data or seamless zone-based location estimation), which are not configured to make use of the embedded ontology reasoner.

5.1 Software components in the Activity Monitor

The complete set of enablers used in the Activity Monitor is shown in Fig. 5. Apart from sensors and specific framework enablers, four new enablers have been developed to support the application's functionalities: *Energy Cost Meter*, *Activity Evaluator*, *Alarm Generator* and *Activity Application GUI/Logic*. The *Activity Evaluator* and *Alarm Generator* enablers are the ones invoking LIS.

After processing several types of data acquired from sensors, the *Activity Evaluator* enabler is in charge of assessing the set of activities that the user performs in the short (W_s) and long term (W_l) (both short- and long-term performances are taken into account in order to globally evaluate user's activities; W_s and W_l are configurable parameters, currently set to 1 and 24 h, respectively). This evaluation is made by comparing the 'energy cost' of each atomic activity performed by the user with a set of 'minimum energy cost' values set by experts. Energy cost is specifically measured in PARs (*Physical Activity Ratio*), which are multiples of BMR (*Basal Metabolic Rate*) per

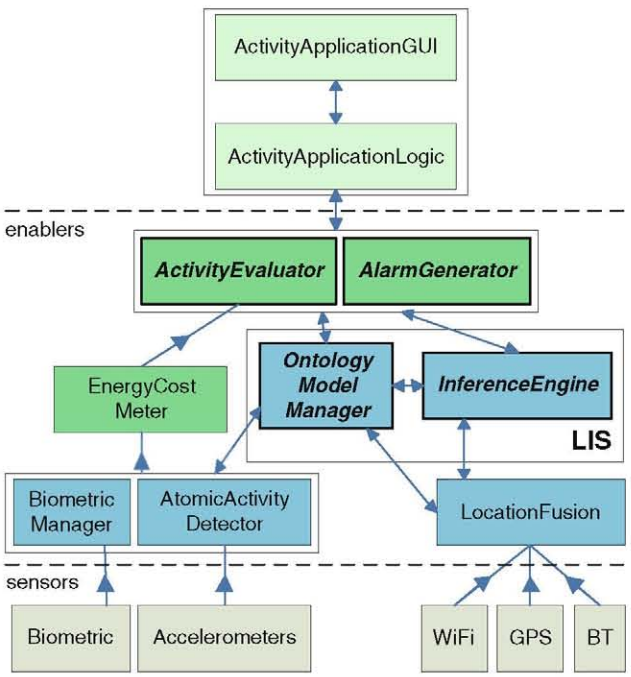

Fig. 5 The Activity Monitor's architecture [49]

minute, the minimal rate of energy expenditure compatible with life [50]. The *Energy Cost Meter* enabler is in charge of quantifying user's activity into PARs. The *Activity Evaluator* uses the general-purpose rule reasoner encapsulated in the *Inference Engine* in order to execute any rule it needs to obtain its outputs. Specifically, several rules and queries are configured, aiming at verifying that the user's performance in the short and long term is not less than a minimal value dynamically established, generating an alarm if this happens.

Once the user's activity level has been evaluated, the *Alarm Generator* enabler is in charge of providing feedback by managing the alarms sent to the user. When an alarm is detected, this enabler first decides whether to notify the user depending on the user's context. Three parameters are currently taken into account for this decision:

- *User's explicit alarms configuration*: no alarm would be generated if the user states so in the static configuration.
- *Short- and long-term evaluations*: an alarm is sent to the user only when both short- and long-term evaluations of the user's activity level are determined to be below the expected minimum.
- *Elapsed time since last alarm acceptance*: in order not to constantly disturb the user, a time between alarms (t_{ba}) needs to be guaranteed before generating new alarms.

Next, once an alarm is generated, the user will be informed through alternative channels depending on his/her context: different types of in-device alarms can be configured (soundless, vibrating, etc.), and email or social network notification could be also considered. Currently, two context parameters are taken into account in this case: user's location and activity. Apart from the default configuration, the user may customize the application's performance, e.g., to use a vibration alarm if working or a soundless one if practicing sports.

The *Activity Evaluator* and *Alarm Generator* enablers follow the architecture defined for the *Generic Enabler* presented in Sect. 4.2. Their detailed logic is explained in the next Section.

5.2 Logic in the Activity Monitor

The instantiated entities stored in the knowledgebase, together with rules and queries, are supporting the reasoning process (Fig. $6C_2$). Entities have been modelled according to the data model described in Sect. 3.2. Data for instantiation can be classified depending on when and how they are generated:

- *Static data* containing the information used to quantify user's activities (into 'energy costs', measured in PARs). These data are pre-stored in the knowledgebase and cannot be modified (Fig. 6A).
- *Asynchronous data* gathering the current user's context information. As the user changes his/her location, activity, walked distance, profile, these data are asynchronously updated in the knowledgebase (Fig. 6B).
- *Pre-processed data*. User's asynchronous context data are periodically pre-processed in order to complete the set of information needed for reasoning purposes. These data are also stored in the knowledgebase (Fig. $6C_1$).

The knowledgebase update process is detailed in Fig. 6, which also gathers the full relationship of context concepts included in the reasoning process.

The ontology model was first adapted to meet the lightweight infrastructure requirements. In addition, the initial package structure (Fig. 2) was tailored, removing those concepts not used in the Activity Monitor to avoid 'out of memory' problems in the reasoner invocation. Then, from the initially planned set of five packages, only *User*, *Context* and *Event* packages were used in practice. Profiling parameters (gender, height, etc.) and configuration (saved POIs, alarms ON/OFF, etc.) are considered in the *User package*; context parameters (location, activity, activity level and heart rate) are included in the *Context package*; finally, alarms status and characteristics are modelled in the *Event package*.

5.3 Reasoning process using LIS

Figure 7 details the model management and reasoner invocation process for the *Activity Evaluator* and the *Alarm Generator* (and multimodal interface selector).

For the *Activity Evaluator*, the process is as follows (Fig. 7, left): (1) the application requests subscription to the user's activity levels measurements and the framework addresses this subscription towards the service in charge of that kind of measure: the *Activity Evaluator* enabler; (2) this enabler knows that it needs to update the data model with the last user's activity level before invoking the reasoner; (3) after updating the knowledgebase, the *Inference Engine* is configured (with a set of SWRL rules and SQWRL queries) and invoked; (4) the *Inference Engine* accesses the knowledgebase in order to obtain every fact appearing in the rule/query sets (in this case, minimum activity level values); (5) the *Inference Engine* answers to the query set informing the *Activity Evaluator* if any activity level alarm needs to be generated; (6) the *Activity Evaluator* updates the knowledgebase with the activity level alarms status (ON/OFF), offering in (7) the estimation

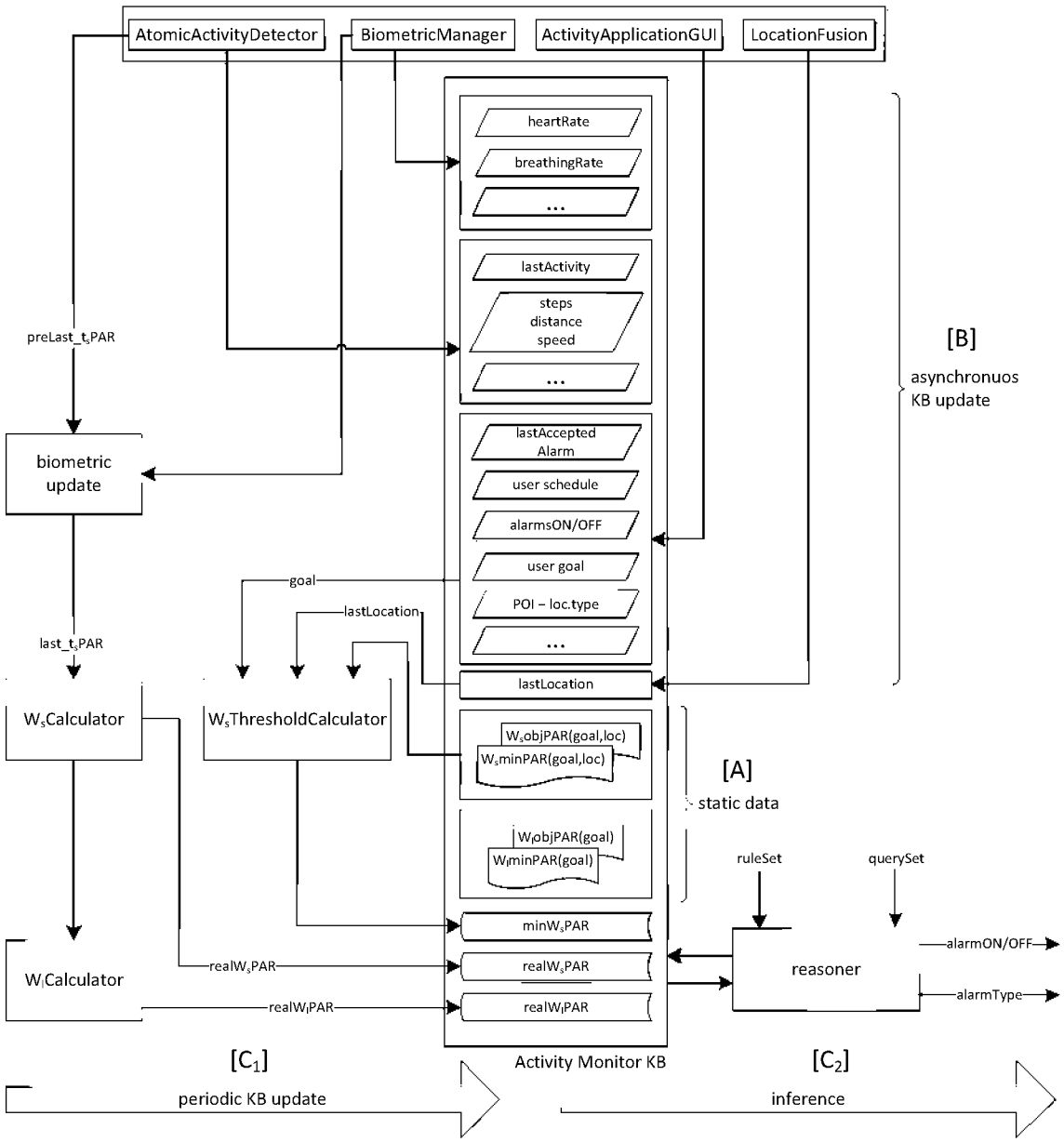

Fig. 6 The Activity Monitor knowledgebase update process

for the user's activity level. Figure 8 details the rules used to evaluate user's activity level at short- and long-term, respectively.

A similar process takes place next for inferring context-aware alarms (Fig. 7, right): (a) the application requests subscription to the context-aware alarms, being this subscription addressed towards the service in charge of that kind of measure: the *Alarm Generator* enabler; (b) the *Inference Engine* is configured (with a set of SWRL rules and SQWRL queries) and invoked; (c) the *Inference Engine* accesses the knowledgebase in order to obtain every fact appearing in the rule/query sets, in this case, current configuration of the application regarding the alarm generation process and context information related to the user; (d) the *Inference Engine* answers to the query set informing the *Alarm Generator* whether an alarm should be generated or not and, if so, the kind of notification to use (in-device alarm, email notification or social network post) according to the user's current activity level and context; finally (e), the *Alarm Generator* updates the data model with the characteristics of the alarm to be shown to the user and (f) also offers this information to the application. Figure 8 details the rules used to decide whether generate an alarm to the user and, if so, the type of alarm to be generated (depending on the user's location, activity or heart rate).

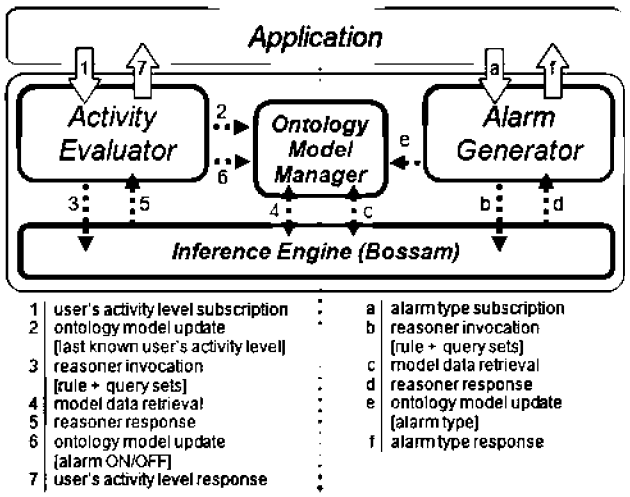

Fig. 7 Activity evaluator (*left*) and context-aware alarm generator and multimodal interface selector (*right*) service overview

6 Conclusions

The light inference system (LIS) presented in this paper is built on available tools for data model management and ontology rule-based reasoning, which have been adapted to work properly in a service-oriented embedded framework to deal with context management in mobile applications. LIS provides a domain-agnostic and configurable reasoning tool to release other framework components from internally implementing particular reasoning mechanisms. Reasoning processes encapsulated within LIS are offered using a common interface, so it can be dynamically configured and invoked from either internal framework components or final applications.

When initiating the design process, we aimed at building 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 have not really been designed to be deployed in mobile devices. On the other hand, state-of-the-art analysis reveals a (limited) number of developments just meeting prerequisite (3). So, to the best of our knowledge, *μJena* (as ontology manager) and *Bossam* (as ontology and rule-based reasoner) are the only ones satisfying all three key requirements (*androjena* has not been taking into account as it was released after making our decision, and it was not compatible with our development tools). Nevertheless, their integration inside the service-oriented framework has not been smooth.

It is important to note that both *μJena* and *Bossam* are research developments. 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* can just 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* communicate between them by exchanging documents stored in the mobile file system, an error prone practice that definitely increases response times. This is, from our point of view, a relevant limitation. Besides, although *Bossam* is able to handle several well-known ontological standards, we experienced some *stack overflow* problems when trying to use SWRL and SQWRL for rules and queries definitions. As argued in the state-of-the-art (Sect. 2), everything suggests that parsing this kind of XML-based annotations overloads resource-constrained devices (a Samsung Omnia with Windows Mobile 6.1, 97 MB RAM and 528 MHz processor was used for our experiments). Finally, a *Bossam*'s proprietary rule and query language (*Buchingae*) were used with success.

Moreover, *μJena* also lacks flexibility regarding the supported standard formats. Although future extensions are planned, at present, it just can read OWL ontologies coded in N-Triples syntax. No report about the performance of this syntax in mobile devices has been found, but updated references regarding this issue [24, 26] point out to Manchester or KRSS2 syntax as some of the most appropriate for resource-constrained devices. In addition, *μJena* does not support any rule or query language standard; besides, it does not allow external ontologies to be imported.

There are still few developments capable of dealing with ontology management and reasoning in mobile devices, and the existing ones are 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 none of the 'well-known' ontological reasoners (Pellet, FaCT++, RacerPro, etc.) are planned to be migrated to mobile environments for the time being. At this point, it is worth mentioning again *androjena*, which is supposed to integrate a reasoner and an ontological manager, overcoming the *μJena* and *Bossam* integration problems stated above.

The application described in Sect. 5 shows the feasibility of using the light inference system to solve common problems in ordinary mobile applications (for instance, the context-aware notifications generation and multimodal interface configuration may be easily integrated in other kind of applications). Anyway, it also reveals semantic tools immaturity regarding their adaptability to resource-constrained devices needs (as the used tools needed to be

Fig. 8 Main rules used in the activity evaluation, context-aware alarm generation and multimodal interface selector processes

Long term evaluation rule:
The minimum PARs associated to the desired user's long term activity goal is compared to the actual PARs related to the user's long term activity, generating a positive or negative evaluation accordingly.

```
IF User(?user) AND
hasPreference(?user, ?goal) AND
ActivityLevelGoal(?goal) AND
hasMinPARlongTerm(?goal, ?minPARlongTerm) AND
hasContext(?user, ?activity) AND
Activity(?activity) AND
lastRealPARlongTerm(?activity, ?realPARlongTerm) AND
associatedEvaluation(?activity, ?evaluation) AND
[?realPARlongTerm < ?minPARlongTerm]
THEN hasNegativeLongTermEvaluation(?evaluation, true)
```

Short term evaluation rule:
The minimum PARs associated to the desired user's short term activity goal is compared to the actual PARs related to the user's short term activity, generating a positive or negative evaluation accordingly.

```
IF User(?user) AND
hasContext(?user, ? activity) AND
Activity(?activity) AND
hasNowMinPARshortTerm(?activity, ?minPARshortTerm) AND
lastRealPARshortTerm (?activity, ?realPARshortTerm) AND
associatedEvaluation(?activity, ?evaluation) AND
[?realPARshortTerm < ?minPARshortTerm]
THEN hasNegativeShortTermEvaluation(?evaluation, true)
```

Context-aware alarm generation:
If the required time between alarms has already expired and the user has enabled the alarms notification and both short and long term evaluations are negative, then, an alarm notification should be sent to the user.

```
IF User(?user) AND
hasContext(?user, ?lastAlarm) AND
LastAlarm(?lastAlarm) AND
hasValue(?lastAlarm, ?lastAlarmTime) AND
hasPreference(?user, ?time_between_alarms) AND
TimeBetweenAlarm(?time_between_alarms) AND
[?lastAlarmTime < (now - ?time_between_alarms)]
hasPreference(?user, ?alerts) AND
AlertStatus(?alerts) AND
hasValue(?alerts, true) AND
associatedEvaluation(?lastAlarm, ?alarm) AND
hasNegativeShortTermEvaluation(?alarm, true) AND
hasNegativeLongTermEvaluation (?alarm, true) then
THEN hasGlobalAlarmState(?alarm, true)
```

Multimodal interface selector (by activity):
A particular alarm type is used to inform the user depending on the user's activity being performed.

```
IF User(?user) AND
hasContext(?user, ?activity) AND
Activity(?activity) AND
hasValue(?activity,<activity_id>) AND
associatedAlarm(?activity, ?alarm) AND
THEN hasAlarmType(?alarm,<alarmType_id>)
```

Multimodal interface selector (by heart rate):
A particular alarm type is used to inform the user depending on the user's heart rate level.

```
IF User(?user) AND
hasContext(?user, ?heartRate) AND
HeartRate(?heartRate) AND
hasValue(?heartRate, ?hr_value) AND
[<maxHRvalue> < ?hr_value] AND
associatedAlarm(?heartRate, ?alarm) AND
THEN hasAlarmType(?alarm,<alarmType_id>)
```

Multimodal interface selector (by location):
A particular alarm type is used to inform the user depending on the user's location.

```
IF User(?user) AND
hasContext(?user, ?poi) AND
POI(?poi) AND
hasPOItype(?poi,<POI_type>) AND
associatedAlarm(?poi, ?alarm) AND
THEN hasAlarmType(?alarm,<alarmType_id>)
```

constantly tuned up in order to avoid, e.g., stack overflow problems). Regarding the Activity Monitor application, it has to be noted that, to date, all the ontological reasoning processes have been applied to 'high-level' context parameters (user's location, activity, etc.), with particular features regarding reasoning invocation frequency, number of facts to reason over, etc. Future extensions of this Activity Monitor will consider the feasibility and convenience of applying these reasoning processes to 'lower levels of abstraction' (e.g. pre-processed accelerometer signals). To what extent it would be convenient to integrate every semantic concept in the ontology is usually a design factor that has to be defined taking into account the computational cost (and subsequent effects) when performing this in resource-constrained devices.

We are currently 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. Having this reasoning platform in a mature state would open the possibility to run performance tests comparing this kind of light embedded architectures with infrastructure-based ones (e.g. those accessing to infrastructure reasoners from mobile devices via RESTful or OWLlink interfaces). Although LIS can be dynamically configured with different rule sets, these rules are currently statically generated by each component invoking LIS reasoning services; this should be reviewed in order to add intelligence and dynamism to the rule generation process.

We are already working on extending our LIS with imperfect information support, which is an important research line in context information management. We are currently analysing different kinds of probabilistic and fuzzy approaches (e.g. [51, 52]). The probabilistic approach is suitable for dealing with the uncertain nature of the information (inherently present in sensor-based context-aware services) whereas the fuzzy logic one is able to manage the vagueness of concepts arising from human perception and cognition processes (also present in most of the real world applications).

Future works also consider empirically analysing performance, flexibility and stability, by measuring the effect of different application invoking LIS services in parallel and comparing the time saving with respect to an infrastructure reasoning service. Ultimately, the impact on the global user's experience, understood as QoS, is to be also considered.

Acknowledgments This work has been supported by the Spanish Ministry of Industry, Tourism and Commerce and the European Fund for Regional Development under grant TSI020301-2008-2, the Ministry for Science and Innovation under grant TIN2008-06742-C02-01 and the Government of Madrid under grant S2009/TIC-1485. The authors also acknowledge enriching discussions with Marcos Sacristán and Alejandro Álvarez of the PIRAmIDE project.

References

1. Baldauf M, Dustdar S, Rosenberg F (2007) A survey on context aware systems. Int J Ad Hoc Ubiquitous Comput 2:263–277
2. Perttunen M, Riekki J, Lassila O (2009) Context representation and reasoning in pervasive computing: a review. Int J Multimed Ubiquitous Eng 4(4):1–28
3. Gu T, Pung HK, Zhang DQ (2004) Toward an osgi-based infrastructure for context-aware applications. IEEE Pervasive Comput 3:66–74
4. Chen H, Finin T, Joshi A (2004) Semantic web in the context broker architecture. In: Proceedings of the second IEEE international conference on pervasive computing and communications (PERCOM'04), Orlando, Florida, USA, Mar 14–17, 2004, IEEE Computer Society, Washington, pp 277–286
5. Fahy P, Clarke S (2004) Cass: a middleware for mobile context-aware applications. In: Proceedings of the workshop on context awareness (MobiSys'04), Boston, Massachusetts, USA, June 6–9, 2004, pp 1–6
6. Chan ATS, Chuang SN (2003) Mobipads: a reactive middleware for context-aware mobile computing. IEEE Trans Softw Eng 29:1072–1085
7. Kleemann T (2006) Towards mobile reasoning. In: Parsia B, Sattler U, Toman D (eds) CEUR workshop proceedings on description logics, vol 189, Lake District, UK, May 30–June 1, 2006, CEUR-WS.org
8. Toma E, Simperl GH (2009) A joint roadmap for semantic technologies and the internet of things. In: Workshop proceedings of the 3rd STI roadmapping workshop charting the next generation of semantic technology at the 6th European semantic web conference (ESWC 2009), June 1, 2009, Heraklion, Greece
9. Bettini C, Brdiczka O, Henricksen K, Indulska J, Nicklas D, Ranganathan A, Riboni D (2010) A survey of context modelling and reasoning techniques. Pervasive Mob Comput 6:161–180
10. Strang T, Linnhoff-Popien C (2004) A context modeling survey. In: Workshop on advanced context modelling reasoning and management as part of UbiComp, Nottingham, UK, Sept 9, 2004, pp 1–8
11. Beigl M, Gellersen H (2003) Smart-its: an embedded platform for smart objects. In: Proceedings of smart objects conference (SOC'03), Grenoble, France, May 15–17, 2003, pp 15–17
12. Siegemund F (2004) A context-aware communication platform for smart objects. In: Ferscha A, Mattern F (eds) Proceedings of the international conference on pervasive computing (PERVASIVE'04), vol 3001, Linz, Viena, April 18–23, 2004, ser. LNCS, Springer, Berlin, pp 69–86
13. Corredor I, Martínez JF, Familiar MS (2011) Bringing pervasive embedded networks to the service cloud: a lightweight middleware approach. J Syst Archit Embed Softw Des (To appear), Special Issue: Emerging applications of embedded systems, Elsevier
14. JSON. http://www.json.org
15. Knappmeyer M, Kiani SL, Fra C, Moltchanov B, Baker N (2010) Contextml: a lightweight context representation and context management schema. In: Proceedings of the 5th IEEE international symposium on wireless pervasive computing (ISWPC'10), Modena, Italy, May 5–7, 2010, IEEE Press, Piscataway, NJ, USA, pp 367–372
16. Ye J, Coyle L, Dobson S, Nixon P (2007) Ontology-based models in pervasive computing systems. Knowl Eng Rev 22:315–347
17. SOPRANO: Service oriented programmable smart environments for older Europeans (2007) Deliverable D1.1.2: review state-of-the-art and market analysis, Version 1.1
18. Boury-Brisset AC (2003) Ontology-based approach for information fusion. In: Proceedings of the sixth international

conference of information fusion, Cairns, Queensland, Australia, July 8–11, 2003, International Society of Information Fusion, vol 1, pp 522–529
19. OWL Web Ontology Language Overview (2004). http://www.w3.org/TR/owl-features/
20. Spyns P, Meersman R, Jarrar M (2002) Data modelling versus ontology engineering. SIGMOD Rec 31(4):12–17
21. RDF/XML Syntax Specification (2004). http://www.w3.org/TR/REC-rdf-syntax/
22. Su X, Riekki J (2010) Transferring ontologies between mobile devices and knowledge-based systems. In: Proceedings of the 2010 IEEE/IFIP international conference on embedded and ubiquitous computing (EUC'10), Hong Kong, China, Dec 11–13, 2010, IEEE Computer Society, pp 127–135
23. Horridge M, Drummond N, Goodwin J, Rector A, Wang HH (2006) The Manchester owl syntax. In: Proceedings of the OWL experiences and directions workshop (OWLED), vol 216, Athens, Georgia, USA, Nov 10–11, 2006, CEUR-WS.org
24. Koziuk M, Domaszewicz J, Schoeneich R, Jablonowski M, Boetzel P (2008) Mobile context-addressable messaging with dl-lite domain model. In: Roggen D, Lombriser C, Trster G, Kortuem G, Havinga P (eds) Smart sensing and context (lecture notes in computer science), vol 5279. Springer, Heidelberg, pp 168–181
25. Schneider PP, Swartout B (1993) Description-logic knowledge representation system specification from the KRSS group of the ARPA knowledge sharing effort
26. Kleemann T, Sinner A (2005) Description logic based matchmaking on mobile devices. In: Baumeister J, Seipel D (eds) Proceedings of 1st workshop on knowledge engineering and software engineering (KESE'05), Koblenz, Germany, Sept 11, 2005, pp 37–48
27. RDF Test Cases (2004) W3C Recommendation. http://www.w3.org/TR/rdf-testcases/#ntriples
28. Cuenca B, Motik B, Wu Z, Fokoue A (2008) OWL 2 web ontology language: profiles. W3C Working Draft
29. Crivellaro F (2007) microJena: Gestione di ontologie sui dispositivi mobile. M.Sc. Thesis, Politecnico di Milano
30. Carroll JJ, Dickinson I, Dollin C, Reynolds D, Seaborne A, Wilkinson K (2004) Jena: implementing the semantic web recommendations. In: Proceedings of the 13th international world wide web conference on alternate track papers & posters, (WWW Alt.'04), New York, NY, USA, May 17–22, 2004, ACM, New York, pp 74–83
31. Sinner A, Kleemann T (2005) Krhyper in your pocket. In: Nieuwenhuis R (ed) Automated deduction CADE-20 (lecture notes in computer science), vol 3632. Springer, Heidelberg, pp 452–457
32. Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B, Dean M (2004) SWRL: a semantic web rule language combining OWL and RuleML. W3C member submission
33. Gu T, Kwok Z, Koh KK, Pung HK (2007) A mobile framework supporting ontology processing and reasoning. In: Proceedings of the 2nd workshop on requirements and solutions for pervasive software infrastructures (RSPSI'07), in conjunction with the 9th international conference on ubiquitous computing (Ubicomp'07), Innsbruck, Austria, Sept 16–19
34. Ali S, Kiefer S (2009) Micro a micro owl dl reasoner for ambient intelligent devices. In: Abdennadher N, Petcu D (eds) Advances in grid and pervasive computing (lecture notes in computer science), vol 5529. Springer, Heidelberg, pp 305–316
35. Vazquez Gomez JI (2007) A reactive behavioural model for context-aware semantic devices. Ph.D. thesis
36. Jang M, Sohn JC (2004) Bossam: an extended rule engine for owl inferencing. In: Antoniou G, Boley H (eds) Rules and rule markup languages for the semantic web (lecture notes in computer science), vol 3323. Springer, Heidelberg, pp 128–138
37. Boley H (2006) The RuleML family of web rule languages. In: Alferes J, Bailey J, May W, Schwertel U (eds) Principles and practice of semantic web reasoning (lecture notes in computer science), vol 4187. Springer, Heidelberg, pp 1–17
38. Tramp S, Frischmuth P, Arndt N, Ermilov T, Auer S (2011) Weaving a distributed, semantic social network for mobile users. In: Proceedings of the 8th extended semantic web conference (ESWC'11), Heraklion, Greece, May 29–June 2, 2011. Springer, Heidelberg, pp 200–217
39. Toninelli A, Pathak A, Issarny V (2011) Yarta: a middleware for managing mobile social ecosystems. In: Riekki J, Ylianttila M, Guo M (eds) Advances in grid and pervasive computing (lecture notes in computer science), vol 6646. Springer, Heidelberg, pp 209–220
40. Specht G, Weithoner T (2006) Context-aware processing of ontologies in mobile environments. In: Proceedings of the 7th international conference on mobile data management, MDM 2006, Nara, Japan, May 9–13, 2006, IEEE Computer Society Washington, DC, USA, pp 86–89
41. Kofod-Petersen A, Aamodt A (2003) Case-based situation assessment in amobile context-aware system. In: University des Saarlandes (ed) Proceedings of artificial intelligence in mobile systems (AIMS'03), Seattle, WA, USA, Oct 12, pp 41–49
42. Raento M, Oulasvirta A, Petit R, Toivonen H (2005) Contextphone: a prototyping platform for context-aware mobile applications. IEEE Pervasive Computing 4(2):51–59
43. Yamabe T, Takagi A, Nakajima T (2005) Citron: a context information acquisition framework for personal devices. In: Proceedings of 11th IEEE international conference on embedded and real-time computing systems and applications (RTCSA'05), Hong-Kong, China, Aug 17–19, IEEE Computer Society, pp 489–495
44. Vázquez JI, de Ipiña DL, Sedano I (2006) Soam: an environment adaptation model for the pervasive semantic web. In: Gavrilova ML, Gervasi O, Kumar V, Tan CJK, Taniar D, Laganá A, Mun Y, Choo H (eds) ICCSA (4) (lecture notes in computer science), vol 3983. Springer, Berlin, pp 108–117
45. Tai W, Brennan R, Keeney J, O'Sullivan D (2009) An automatically composable OWL reasoner for resource constrained devices. In: Proceedings of the 2009 IEEE international conference on semantic computing (ICSC'09), IEEE Computer Society, Washington, DC, USA, pp 495–502
46. Organization for the Advancement of Structured Information Standards (2006) Reference model for service oriented architecture 1.0. OASIS
47. Noy NF, McGuinness DL (2001) Ontology development 101: a guide to creating your first ontology. Tech. Rep. Stanford Knowledge Systems Laboratory and Stanford Medical Informatics, Stanford, CA, USA
48. Iglesias J, Bernardos AM, Alvarez A, Sacristan M (2010) A light reasoning infrastructure to enable context-aware mobile applications. In: Proceedings of the 2010 IEEE/IFIP international conference on embedded and ubiquitous computing (EUC'10), IEEE Computer Society, Washington, DC, USA, pp 386–391
49. Iglesias J, Cano J, Bernardos AM, Casar J (2011) A ubiquitous activity-monitor to prevent sedentariness. In: Proceedings of the IEEE international conference on pervasive computing and communications workshops (PERCOM'11 workshops), Seattle, WA, USA, Mar 21–25, IEEE, pp 319–321
50. Food and Agricultural Organization of the United Nations, United Nations University, World Health Organization (2004) Human energy requirements: report of a joint FAO/WHO/UNU expert consultation. Rome, 17–24 Oct 2001. FAO food and nutrition technical report series. Food and Agricultural Organization of the United Nations
51. Klinov P (2008) Pronto: a non-monotonic probabilistic description logic reasoner. In: Bechhofer S, Hauswirth M, Hoffmann J,

Koubarakis M (eds) The semantic web: research and applications (lecture notes in computer science), vol 5021. Springer, Heidelberg, pp 822–826
52. Bobillo F, Straccia U (2010) Representing fuzzy ontologies in OWL 2. In: Proceedings of the IEEE international conference on fuzzy systems (FUZZ'10), Barcelona, Spain, July 2010, IEEE, pp 2695–2700
53. Prud'hommeaux E, Seaborne A (2008) SPARQL query language for RDF. W3C recommendation. http://www.w3.org/TR/rdf-sparql-query/
54. Hori M, Euzenat J, Patel-Schneider PF (2003) OWL web ontology language XML presentation syntax. W3C Note. http://www.w3.org/TR/owl-xmlsyntax/
55. O'Connor MJ, Das AK (2008) SQWRL: a query language for OWL. In: Hoekstra R, Patel-Schneider PF (eds) Proceedings of the 6th international workshop on OWL: experiences and directions (OWLED'09), vol 529, Chantilly, VA, USA, Oct 23–24, Springer, Berlin/Heidelberg