International Journal on Semantic Web and Information Systems
Volume 13 • Issue 1 • January-March 2017

Enabling Interoperability in the Internet of Things: A OSGi Semantic Information Broker Implementation

Alfredo D'Elia, ARCES, University of Bologna, Bologna, Italy
Fabio Viola, ARCES, University of Bologna, Bologna, Italy
Luca Roffia, DISI, University of Bologna, Bologna, Italy
Paolo Azzoni, Eurotech Group, Trento, Italy
Tullio Salmon Cinotti, ARCES, DISI, University of Bologna, Bologna, Italy

ABSTRACT

Semantic Web technologies act as an interoperability glue among different formats, protocols and platforms, providing a uniform vision of heterogeneous devices and services in the Internet of Things (IoT). Semantic Web technologies can be applied to a broad range of application contexts (i.e., industrial automation, automotive, health care, defense, finance, smart cities) involving heterogeneous actors (i.e., end users, communities, public authorities, enterprises). Smart-M3 is a semantic publish-subscribe software architecture conceived to merge the Semantic Web and the IoT domains. It is based on a core component (SIB, Semantic Information Broker) where data is stored as RDF graphs, and software agents using SPARQL to update, retrieve and subscribe to changes in the data store. This article describes a OSGi SIB implementation extended with a new persistent SPARQL update primitive. The OSGi SIB performance has been evaluated and compared with the reference C implementation. Eventually, a first porting on Android is presented.

KEYWORDS

Android, Benchmark, OWL, Performance, Publish-Subscribe, RDF, Smart-M3, SPARQL

1. INTRODUCTION

The recent interest on the application of Semantic Web technologies to the Internet of Things (IoT) (Barnaghi, Wang, Henson et al., 2012) inspired international research projects and academic works aimed at exploiting their potential. The Semantic Web project was born to change the Web and drive it towards the original vision that Tim Berners Lee had in mind: a web of data (Berners-Lee, Hendler, Lassila et al., 2001). Semantic Web technologies allow to represent (i.e., RDF - Resource Description Framework, http://www.w3.org/RDF/, OWL - Web Ontology Language Reference, W3C Recommendation, http://www.w3.org/TR/owl-ref/) and retrieve (i.e., SPARQL - SPARQL 1.1 Overview W3C Recommendation, http://www.w3.org/TR/sparql11-overview/) semantically enriched information, that have a high level of generality and inter-disciplinarity. One of the most interesting

DOI: 10.4018/IJSWIS.2017010109

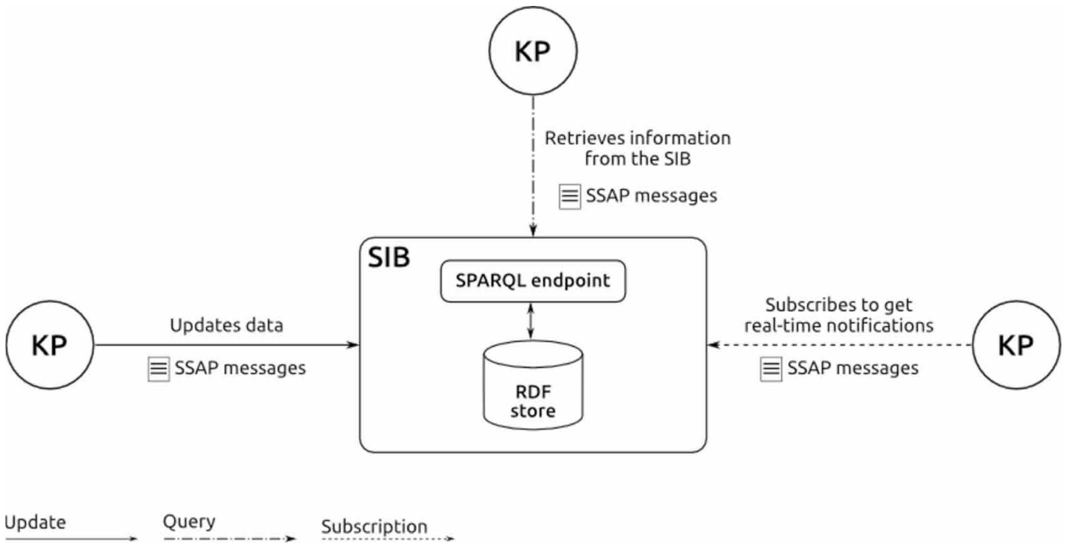
fields of application is that of collaborative agents offering advanced services for private users and enterprises. The Semantic Web also applies to classification, abstraction, mining and reasoning over large amount of data. Information about energy consumption, user profiles, environmental monitoring, financial data and other similar data sources can be analysed by smart software agents and used to perform context-aware actions (where context can be defined as "any information that can be used to characterize the situation of an entity, where an entity can be a person, a place or physical or computational object" (Abowd et al., 1999)).

In 2008, the ARTEMIS joint undertaking funded the FP7 European Project SOFIA (Smart Objects for Intelligent Applications) aimed at developing a platform for sharing semantic information for the widest possible range of devices and agents. Since the conclusion of SOFIA, the platform has been adopted, evaluated and extended in several EU projects (e.g., CHIRON, IoE, RECOCAPE, IMPRESS, ARROWHEAD) in partnership with industrial players. Some of these projects are still ongoing and new EU projects proposals are currently in preparation to continue from the achieved results.

The resulting multipurpose architecture was called Smart-M3 (Multi-platform, Multi-vendor, Multi-part) and is shown in Figure 1. Smart-M3 follows a publish-subscribe model and proposes a simple approach based on two components, the Semantic Information Broker (SIB) and the Knowledge Processors (KPs), interacting through the Smart Space Access Protocol (SSAP). The SIB stores and shares the semantic information that is exchanged through the SSAP and elaborated by KPs. KPs can update and query the stored information, and subscribe to changes. The SIB is not bound to a specific implementation and it is generally intended as a software module able to read, process and produce SSAP messages while coherently managing the semantic knowledge base (KB). As shown in Figure 1, the SIB can be seen as a wrapper over a SPARQL endpoint to enable the publish-subscribe mechanism.

When the Smart-M3 architecture began to be used in many heterogeneous scenarios, the interest of the community for new more performing and featured SIB versions grew and brought to different implementations optimized for different requirements and vertical domains. In the recent years, the research community improved many aspects of the Smart-M3 original platform including, for example, the programming model, the SSAP protocol, the supported standards and the SIB module itself. The KPs can be written in several programming languages (Java, C#, Python, Javascript, PHP, C, Ruby and Lua), are agnostic to the transport channel and may run on different platforms, but must communicate with the SIB through the SSAP.

Figure 1. Smart-M3 architecture



This paper describes a SIB implementation based on the OSGi framework, built on top of the authors experience on the development of the Smart-M3 platform for smart space applications (Bedogni et al., 2013; Manzaroli et al., 2010; Morandi, Roffia, D'Elia, Vergari, & Cinotti, 2012; Ovaska, Cinotti, & Toninelli, 2012; Vergari et al., 2011). This implementation is oriented to the IoT and M2M industrial domains.

The previous authors experiences in Smart-M3 development put in evidence the main strengths and weaknesses of the existing implementations in these domains and allowed the definition of the requirements for a new implementation. First of all, it is necessary to clarify that many SIB implementations should exist and coexist: depending on the application scenario, on the target hardware and on the required features, one SIB implementation may behave better than others in terms of costs, resource allocation, performance or functionalities. Therefore, we focused on three main aspects: the IoT vision, the dynamicity of the requirements changes in the envisaged scenarios and the need to support mobile devices. It resulted evident how the current technological trend led by the Internet of Things requires features not supported by the existing SIB implementations. In fact, in 2012 (i.e. when the currently most used implementation was released) the IoT vision was not yet as stable and relevant as today. The aim of this research work is to implement a SIB prototype acting as a IoT middleware with a dynamic feature set and that can be ported on mobile and embedded systems. The OSGI SIB allows for example to develop IoT gateways, to manage low resources devices (i.e., sensors and actuators) and expose functionalities to high-level services and web applications. The new SIB primitives improve the existing set of functionalities, proposing innovative features that, in their turn, can be also augmented to support the high dynamicity with which the functional requirements change for the modern scenarios. Software modularity, reliability and portability together with performance and simple deployment of new features are the main research challenges motivating the new SIB implementation and, at the same time, the choice of Java and OSGi that provides interesting advantages in terms of portability, productivity and compatibility with existing libraries and frameworks. OSGi is widely diffused both in the research and in the industrial communities because it offers a reliable Service Oriented Architecture (SOA) decoupling the atomic software entities called bundles. A complex software like the SIB can be decomposed into bundles interacting through service calls where, according to the "separation of concerns" principle, each bundle becomes a module with a well-defined interface.

Three major contributions are described in this paper. First, we improved the flexibility of a Smart-M3 based architecture: it is possible to define many bundle sets optimized for different situations that can be started-stopped or substituted even at runtime without interruptions to service provision. A second important contribution of this work is the implementation and evaluation of a new functionality of the SIB module: the persistent SPARQL update primitive (PU) to implement rule-like facilities. The last major contribution described in this paper is the porting on Android of the SIB achieved with a minimal effort thanks to the Java and OSGi portability. The introduction of mobile devices as host of a SIB increments the number of scenarios where Smart-M3 based software architectures can be used to provide environmental intelligence and advanced smart services in the Internet of Things. Eventually, the Smart-M3 implementation based on OSGi represents also a benefit for the OSGi framework itself. Currently, OSGi specifications do not address interoperability issues at the information level, therefore the introduction of native interoperability technologies represents an important improvement of the OSGi platform.

2. RELATED WORK

The level of interoperability, dynamicity, flexibility, expressivity and extendibility required in the development of IoT applications could be provided by Semantic Web based interoperability platforms like the Task Computing Environment (TCE) (Masuoka, Parsia, & Labrou, 2003), Context Broker Architecture for Pervasive Computing (CoBrA) (Chen et al., 2004; Chen, Finin, & Joshi, 2005), Context Aware Platform (CAP) (Lamorte et al., 2007), Semantic Space (Wang, Dong, Chin et al., 2004), Semantic middleware for IoT (Song, Cardenas, Masuoka et al., 2010), Smart objects awareness and adaptation Model (SoaM) (Vazquez, de Ipiña, & Sedano, 2007), Amigo (Thomson, Bianco, Mokhtar, Georgantas, & Issarny, 2008), SPITFIRE (Pfisterer et al., 2011) and OpenIoT (Le Tu'n et al., 2012), to name a few. The main drawback of Semantic Web technologies concerns the low level of performance, that makes it difficult to achieve responsiveness and scalability, required in many IoT applications. The main reason for the poor performance is that Semantic Web technologies have been designed to process data sets consisting of big amounts of RDF triples (e.g., Open Linked Data project - Connect Distributed Data across the Web, http://linkeddata.org/) that evolve constantly but at a much slower rate compared to the rate of elementary changes occurring in the physical environment.

In (Gyrard et al., 2014) an approach for combining, enriching and reasoning on machine to machine (M2M) data is presented. This approach, called M3, is based on SenML (Jennings et al., 2012) and defines an ontology to map concepts such as sensors, observations, unit of measures in a uniform way among different applications domains. The interoperability of multiple heterogeneous sources is granted by Semantic Web technologies. Despite looking similar to M3 (even in the name), in Smart-M3 the research effort is focused on interoperability, generality and reactivity obtained through the publish-subscribe mechanism. M3 is instead focused on the sensor domain whose support is strengthened by interesting instruments of reasoning and learning, but doesn't provide asynchronous notifications and the multi-domain feature.

Semantic Web technologies, as well as the OSGi framework, play a central role also in Clerezza (Clerezza, 2016), a service platform developed by Apache. Despite having an underlying RDF store, the scope of Clerezza is quite different from Smart-M3 being focused on the fast development of web applications. Smart-M3 is instead aimed at the development of IoT and ubiquitous computing applications. Furthermore, Smart-M3 uses SPARQL not only as a query language, but also to subscribe to RDF graphs.

Research approaches aiming at adopting Semantic Web technologies to promptly react to changes in physical environment have been investigated and can be framed in general within the research topics known as Stream Reasoning (Della Valle, Ceri, van Harmelen et al., 2009), Linked Stream Data Processing (Le-Phuoc, Parreira, & Hauswirth, 2012) and Content-Based Publish-Subscribe (Eugster, Felber, Guerraoui et al., 2003). With reference to semantic publish-subscribe systems, in (Kjær & Hansen, 2010) are presented a model of ontology-based publish-subscribe that uses the Semantic Web Rule Language to define event dissemination and its implementation. In (Wang, Jin, & Li, 2004) the authors define an event as an RDF graph and propose their own subscription language and matching algorithm, while in (Chirita, Idreos, Koubarakis et al., 2004) a solution to incorporate publish-subscribe capabilities in an RDF-based P2P network is presented. A similar approach is proposed in (Shi, Yin, Li et al., 2007), where events and subscriptions are described as RDF graphs and a fast graph-matching algorithm is presented. To the best of our knowledge, a first attempt to use SPARQL as subscription language with notifications and events expressed using XML was presented in (Skovronski, 2007). In (Pellegrino, Baude, & Alshabani, 2012; Pellegrino, Huet, Baude et al., 2013) another approach based on structured P2P networks (i.e., named CAN) is investigated. In (Murth, 2008), Murth et al. focus their attention on coordination problems, expressing subscriptions as RDF graph patterns (i.e., corresponding to SPARQL basic graph patterns) and introducing the idea of "continuous insertion" (i.e., described as a SPARQL CONSTRUCT form) as a way to generate new events when specific events occur (i.e., the Persistent Update primitive implemented in the OSGi SIB). The same authors also present a formal model based on first order logic and temporal propositional logic (Murth & Kühn, 2009b), a heuristic to optimize the notifications latency (Murth & Kühn, 2009a) and knowledge-based interaction patterns based on a set of Semantic Space API Operations (Murth & Kuhn, 2010).

With reference to Smart-M3, in this paper we compare the OSGi implementation with RedSIB, the C implementation of the SIB component that is currently the most used. The RedSIB implementation extends the Smart-M3 original implementation (Morandi et al., 2012). It is based on Redland RDF libraries (Beckett, 2002) and it can use one of the RDF stores supported by Redland (e.g., hashes, Berkeley DB, Virtuoso (Erling & Mikhailov, 2009)). The SSAP protocol (Honkola, Laine, Brown et al., 2010) has been extended to support two new primitives (i.e., SPARQL UPDATE and SPARQL SUBSCRIBE), and otherwise was left unchanged to maintain backward-compatibility. RedSIB can also be deployed on low cost single core platforms (e.g., on a RaspberryPi). Smart-M3 had a considerable impact on research and literature inspiring works from a large community in different applicative fields like library generation (Lomov, Vanag, & Korzun, 2011), blogging (Korzun et al., 2011), electric mobility (Bedogni et al., 2013), maintenance (D'Elia et al., 2010; Pantsar-Syväniemi et al., 2011), advertisement (Hamza et al., 2014), application development (Palviainen & Katasonov, 2011), healthcare (Vergari et al., 2010). These works led to different SIB implementations derived from the original idea such as solutions integrated in Eclipse (Gómez-Pimpollo & Otaolea, 2010) or optimized for embedded devices (Eteläperä, Kiljander, & Keinänen, 2011; Viola et al., 2016). The OSGi SIB presented in this paper differs from the original and from the other implementations because, as explained in the rest of the article, it is mainly focused on portability, modularity and extendibility.

3. THE OSGI SIB

The choice of the OSGi framework to develop the SIB functionalities is motivated by many factors. An OSGi implementation of Smart-M3 represents a strategic step in order to develop an interoperability platform able to address the requirements of vertical application contexts. The OSGi framework, with a service oriented architecture and a modular philosophy, allows to optimize costs, provides agility and flexibility, ensuring systems evolution. Moreover, OSGi based applications can integrate easily and dynamically, without writing custom specific integration code and without extra costs.

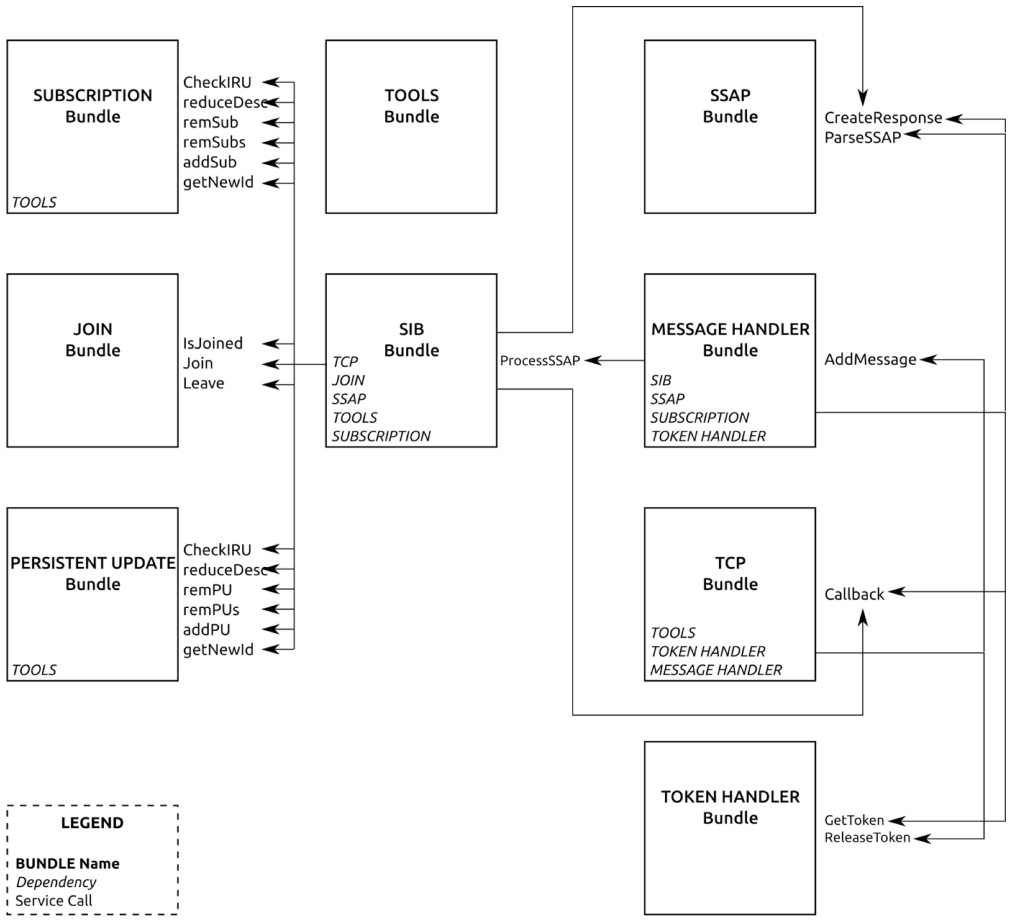
The OSGi applications framework (Open Services Gateway initiative - http://www.osgi.org) provides a programming model for developing, assembling and deploying modular applications based on Java EE technologies. It defines a standardized, component-oriented computing environment for general networked services that is intended to significantly increase the quality of the produced software. OSGi was designed as a services gateway for set-top boxes and, since its introduction in 1998, it has become broadly adopted by industry. It has been extensively used in several application contexts (e.g., automotive, mobile and fixed telephony, industrial automation, gateways & routers, private branch exchanges) and, today, it is supported in many integrated development environments (e.g., IBM Websphere, SpringSource Application Server, Oracle Weblogic, Sun's GlassFish, Eclipse, and Redhat's JBoss) and by key companies (e.g., Oracle, IBM, Samsung, Nokia, IONA, Motorola, NTT, Siemens, Hitachi, Ericsson).

OSGi provides the environment for running many different applications implemented with a strong component-based SOA oriented model: the framework provides a set of comprehensive and advanced features for installing, starting, stopping, updating, and uninstalling applications dynamically. Furthermore, the framework is adaptive, because bundles can find out what capabilities are available on the system through a service registry and can adapt consequently the functionality they provide. The dynamicity, modularity and flexibility of this SOA framework have been introduced to cover some lacks of the Java platform and improve it. The specification defines a security model, an application lifecycle management model, a service registry, an execution environment and modules. The bundle is the basic building block when developing and deploying an application. An OSGi application emerges from a set of bundles, of different types, to provide a coherent business logic.

The software architecture of the OSGi SIB is shown in Figure 2. The blocks correspond to bundles, the oriented arrows are service calls while inside the blocks the bundle names are indicated with bold characters and the dependencies are written in italic.

The implemented SIB architecture is strongly based on the OSGi modular approach that reduces the complexity, in terms of system architecture and components development. This approach

Figure 2. OSGi SIB software architecture



contributes to the rationalization of applications and exploits components reuse, both at system level and at developer community level. The OSGi SIB components are designed as configurable OSGi Declarative Services exposing service API and raising events. Following the OSGi development model, the services offered by the SIB are provided on a per-bundle basis. There is not a one to one mapping between services and bundles and, in many cases, a set of bundles represents a service. The binary version of bundle(s) and the service API are the only elements required for application development and, in some cases, services require multiple implementations. Currently, the OSGi SIB runs on an open source OSGi implementation called Equinox (release 3.7.2) from the Eclipse Foundation. This implementation is very simple and potentially ensures a wide diffusion of the SIB in terms of application contexts, operating systems and devices.

From the developer perspective, all the services registered to the framework can be called without worrying of service availability (e.g., the SSAP bundle always exposes the ParseSSAP service and if a message is currently under the parsing process the framework will take care of the synchronization issues). The module implemented to manage the communication with the KPs is the TCP bundle. If a message is sent from a KP to the SIB, the TCP bundle reads its content and adds a reference to the message to be processed in a queue managed by the Message Handler bundle (i.e., see AddMessage in Figure 2). The Message Handler bundle receives requests as generic strings, then it delegates the SSAP bundle to transform these requests into a structured internal representation. The Message

Handler bundle associates the structured requests with an internal identifier through the GetToken service of the Token Handler bundle, and asks to the SIB bundle to process the request and to provide an appropriate response. When the SIB bundle processes a message, different data flows are possible depending on occurred errors (i.e., the KP making the request does not occur in the list of the Joined KPs managed by the Join bundle) and notifications to be sent (as it will be verified through the services exposed by the Subscription bundle). In all the cases, the SIB bundle generates an internal representation of the messages to be sent back. Messages are serialized by the SSAP bundle through the CreateResponse primitive and sent back to KPs by the callback function of the TCP bundle. The Tools bundle is a reference for the other bundles, it does not expose any service, but exports the relevant classes used to represent the main internal entities like the input-output messages, the notifications, the subscription template and so on. The RDF store is managed by the SIB bundle: for this purpose the well-known Jena libraries (Jena, 2007) have been used to have a solid and well maintained interface toward the stored information. In the Jena framework are also included the ARQ libraries and the TDB libraries, used respectively to efficiently manage SPARQL queries/updates and to store persistent data on the disk.

The modular decomposition of the SIB may bring to many considerations and analysis. In this work the discussion is limited to the three major contributions identified in the introduction (i.e., flexibility, new functionalities and mobile version of the OSGi SIB). Flexibility is inherent in the modular nature of OSGi based systems and it can be best appreciated with regard to transport and protocol. New transport technologies can be supported by adding new bundles with the same interface of the TCP bundle: Bluetooth, ZigBee, COAP - Constrained Application Protocol - the Knowledge Sharing Protocol (KSP) (Kiljander, Morandi, & Soininen, 2012), or other relevant transport technologies can be substituted or juxtaposed to TCP bundle with the safe addition of modules that can be developed independently by specific area experts. Besides supporting new transport methods, future scenarios may require modification of data serialization (e.g., the migration from XML to JSON in order to limit the dimension of the messages). In this case, a new version of the SSAP bundle will be enough to complete the migration to the new message serialization format.

Concerning new functionalities, these can be achieved by adding new bundles or extending existing ones. In the next section, a new feature is described: the persistent SPARQL Update (PU) that has been implemented by adding one bundle and by doing modifications localized in the SIB bundle and in the SSAP bundle.

3.1. Persistent Update

The persistent update (PU) is a new primitive specifying a SPARQL 1.1 update which is executed once when it is sent to the SIB and then acting persistently on the RDF store until it is deactivated.

The PU is the answer to a very important research challenge for Smart-M3 based systems: making the semantic knowledge base (KB) able to change itself depending on context. The KB in the SIB has always been a passive entity queried or modified by clients, but with the implementation of PU, a set of rules persistently acts on the KB making it possible a totally new whole set of scenarios.

In general a SPARQL update is made up of three clauses: delete, insert and where. The insert and the delete clauses contain variables which are bound by the SPARQL endpoint at the query execution time. The triples inserted or removed by a SPARQL update depend on the RDF store content (i.e., the same SPARQL update may have different effects on the RDF store if launched at different times) so the PU semantics can be considered similar to that of a rule. The rule "If A is a friend of B then A knows B" can be translated in the PU as:

```
INSERT { ?a <#knows> ?b }
WHERE { ?a <#friendOf> ?b }
```

which acts on the RDF store promptly and persistently: every time a statement about friendship between two people is asserted, the PU checks if also the assertion about being acquaintances is present and if not it inserts it in the semantic store. The expressivity of PUs is high so they can be used for heterogeneous tasks like generating alarms in front of sensor observations, proposing personalized advertisements depending on user profiles and context, ontology alignment, data abstraction and so on. The implementation of the PU consisted of three phases. First, the SSAP protocol has been modified to support the messages related to the new primitive. Four messages have been added to the SSAP specification: MakePU request is sent from a KP to the SIB and it encapsulates the SPARQL update that will be persistently executed. The corresponding response is the MakePU response message which contains the PUID (i.e., a unique identifier used from the KPs to refer to a specific PU). The RemovePU request is sent from a KP to delete a PU through its PUID. If the deletion is correctly performed, then a RemovePU response message is sent back to the KP. Second, an OSGi bundle has been implemented to perform all the logic necessary for the PU management; the corresponding calls to its services have been added in the SIB bundle. The third phase of the PU implementation consisted in a performance optimization to limit the overhead due to PUs execution. This optimization has been the critical part of the PU integration and will be more detailed in Section 4.5.

3.2. Android SIB in Java

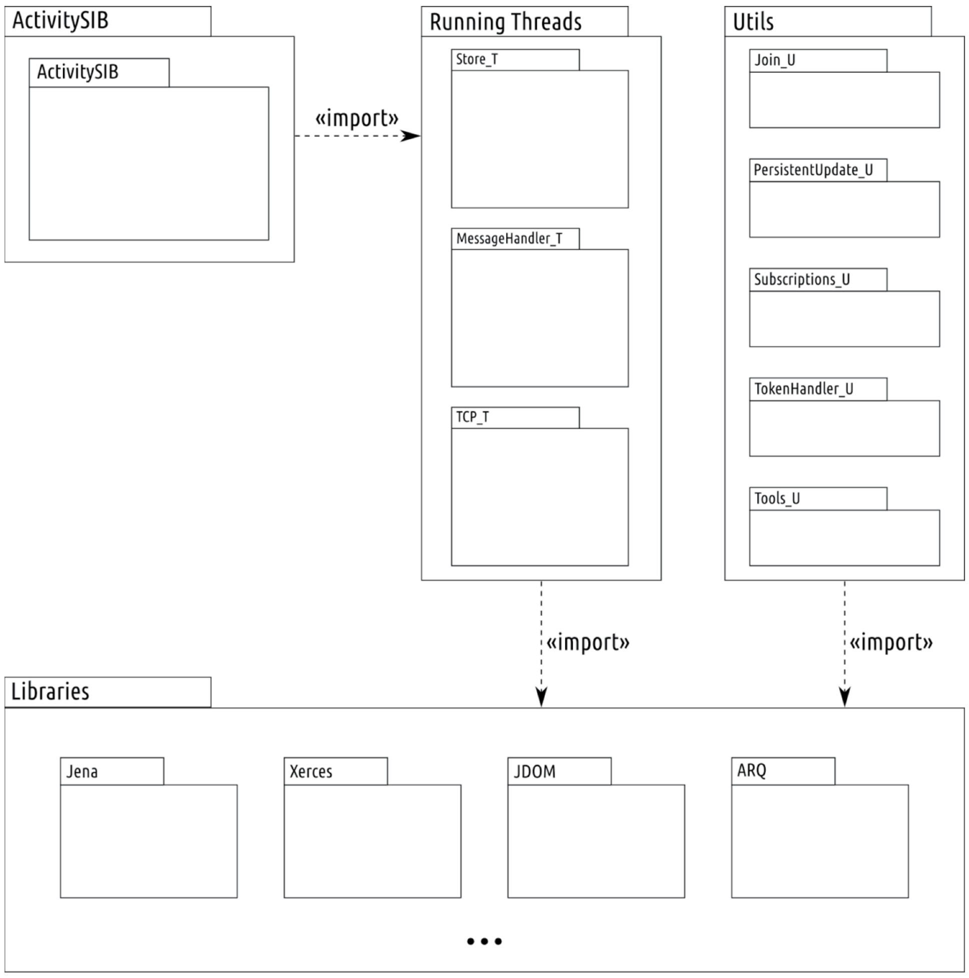
The OSGi SIB was conceived for its flexibility and portability. The Java language and OSGi allowed to simply install and run SIBs not only on Linux platforms (i.e., like the reference implementation that is bound to Linux dependencies), but also on Windows and MAC OS. A more difficult task was to port the SIB on mobile devices like tablets and smartphones. The porting on mobile devices is a relevant task because it extends the scenarios covered by Smart-M3 with new features and potentialities. Healthcare, electric mobility, social services and many other domains may benefit of a mobile SIB representing data in a semantic (i.e., interoperable) format. In the healthcare domain, for example, end users equipped with wearable sensors could store their biometrical parameters on their personal smartphone in a semantic format. Stored data, together with other information on the mobile SIB (e.g., the user profile and data observed from the internal equipment of the device) can be elaborated at run time or offline by medical personnel or interoperable services to protect the end user health.

As the OSGi SIB is written in Java, the porting to a mobile platform was directed towards the Android OS whose Dalvik virtual machine supports most of the Java language. We found that the OSGi SIB does not work without modifications on Android 4.2 or less, so we implemented a new Java SIB version specific to solve this portability issue, leaving to future works the porting of the main OSGi SIB distribution. The main encountered issues were:

- Dynamic class loading poorly supported in Dalvik VM and often used in OSGi framework;
- I/O to be redirected from Java console or managed in a way specific for the Android OS;
- Some of the imported packages, like ARQ, contain too many methods for the Dalvik virtual machine.

The dynamic class loading problem was solved simplifying the software architecture into the one shown in Figure 3, not based on OSGi bundles but on simple Java classes and threads. An Android Activity (ActivitySIB) has been implemented for Input-Output operations and coordinates with other three main threads (TCP-IP, MessageHandler and Store) to provide the typical SIB functionalities. As it is possible to infer from the figure some of the bundles have been transformed in simple classes (Join_U, PersistentUpdate_U, Subscription_U, TokenHandler_U, Tools_U) while others become threads. Thanks to the use of symbolic links and auxiliary libraries (Jena, Xerces, JDOM, ARQ), the deployed packages meet the Dalvik VM requirements.

Figure 3. Android SIB software architecture



4. EVALUATION

The performance evaluation takes into consideration separately the system behaviour and reactivity of the KB modification, of the subscriptions, the new persistent update (PU) primitive and KB query. The analysis is always done, where applicable, in a comparative way. The comparison between the performances of the old (namely RedSIB, Morandi et al., 2012) and of the proposed implementation allows to make clear statements about the quality of the work done and to quantify, if present, the price paid to provide the new features and to offer the code modularity and portability.

While frameworks, benchmarks and methods for performance evaluation of Semantic Web systems have been proposed in the literature (Bizer & Schultz, 2009; Garcia-Castro & et al., 2013; Guo, Pan, & Heflin, 2005; Guo, Qasem, Pan, & Heflin, 2007), these methods are not suitable to evaluate the OSGi SIB with reference to its specific features (i.e., SPARQL subscription and SPARQL persistent update). Since a functional and a quantitative characterization are needed for the characterization of our prototype, we defined a benchmark inspired by a smart public lighting system that will be briefly described later on in this section and for which we refer the interested reader to our detailed description in (Roffia et al., 2016). This benchmark has been adopted to:

- Compare the OSGi SIB and the RedSIB subscriptions notification latency;
- Compare the OSGi SIB and the RedSIB update time;
- Evaluate the OSGi SIB specific persistent SPARQL update primitive.

Being not possible a comparative analysis with previous SIB implementations, in this case we evaluated the overall effect of PU on the SIB performances. As PU are persistent operations like subscriptions. In fact, they require computational resources that is important to characterize, in particular, if they may affect the overall performance.

As the literature offers benchmarks to evaluate the SPARQL query performance, an existing benchmark (Schmidt, Hornung, Lausen, & Pinkel, 2009) has been chosen in order to obtain results that can be compared with other solutions.

4.1. Smart Lighting System Benchmark

The benchmark designed to evaluate the OSGi SIB implementation is inspired by the public lighting system of a city with large, medium, small and very small roads (i.e., roads with up to 100, 50, 25 and 10 lamp-posts). Each post is supposed to be equipped with a lamp and two sensors (i.e., temperature and presence). Each road and each lamp within a road are identified by a URI respectively in the form: ROAD_URI_X and LAMP_URI_X_Y, where X is a road identifier, while Y is a lamp identifier within a road (i.e., Y varies from 1 to N_{LAMP}, where N_{LAMP} is the amount of lamp-posts in road X). Each lamp is characterized by a status (i.e., ON, OFF, BROKEN), a dimming value (i.e., 0-100%) and a type (i.e., LED, TRADITIONAL). Each post is identified by its geographical position (i.e., latitude and longitude), while each sensor is represented by a set of properties: the type (e.g., TEMPERATURE, PRESENCE), the unit of measurement (e.g., °C), the value (e.g., 32, True, False) and a timestamp (i.e., expressed as Unix time extended to µs).

Table 1 provides the details about the ontology and the SPARQL endpoint RDF store size (i.e., number of RDF triples) of a city with 9500 posts (i.e., 42 triples are needed to describe a single lamp-post).

The benchmark considers the following primitives, where X indicates the index of a road, while Y indicates the index of a lamp post within a road:

Table 1. Smart lighting benchmark knowledge base summary

OWL Ontology T-BOX Content				
Classes	27			
Individuals	26			
Object Properties	16			
Datatype Properties	8			
Lamp-Post Instances				
Road Type	**NLAMP/Road**	**Roads**	**Lamp-Posts (Sensors)**	**RDF Triples**
Very small	10	100	1K (2K)	35K
Small	25	100	2.5K (5K)	88K
Medium	50	100	5K (10K)	175K
Large	100	10	1K (2K)	35K
	Total	310	9.5K (19K)	334K

Update U1 (i.e., set to "n" the dimming value of lamp Y within road X)

```
PREFIX unibo: <http://www.unibo.it/benchmark#>
DELETE { unibo:LAMP_URI_X_Y unibo:hasDimmingValue ?dimming }
INSERT { unibo:LAMP_URI_X_Y unibo:hasDimmingValue "n" }
WHERE { unibo:LAMP_URI_X_Y unibo:hasDimmingValue ?dimming }
```

Update U2 (i.e., the presence sensor of lamp Y within road X detects a presence)

```
PREFIX unibo: <http://www.unibo.it/benchmark#>
DELETE { unibo:SENSOR_URI_X_Y unibo:hasSensorDataValue ?value }
INSERT { unibo:SENSOR_URI_X_Y unibo:hasSensorDataValue "true" }
WHERE { unibo:SENSOR_URI_X_Y unibo:hasSensorDataValue ?value }
```

Subscription S1 (i.e., subscribe to changes of the dimming value of any lamp within road X)

```
PREFIX unibo: <http://www.unibo.it/benchmark#>
SELECT ?lamp ?dimming
WHERE { unibo:ROAD_URI_X unibo:isConnectedTo ?post .
?post unibo:hasLamp ?lamp .
?lamp unibo:hasDimmingValue ?dimming}
```

Persistent update PU1 (i.e., if a presence sensor of any lamp of road X detects a presence then all the lamps of road X are turned on)

```
PREFIX unibo: <http://www.unibo.it/benchmark#>
DELETE { ?lamp unibo:hasDimmingValue ?dimming }
INSERT { ?lamp unibo:hasDimmingValue "100" }
WHERE { unibo:ROAD_URI_X unibo:isConnectedTo ?post .
unibo:ROAD_URI_X unibo:isConnectedTo ?postUpdate .
?postUpdate unibo:hasLamp ?lamp .
?lamp unibo:hasDimmingValue ?dimming .
?post unibo:hasSensor ?sens .
?sens unibo:hasSensorData ?data .
?data unibo:hasSensorDataValue "true" .
?data unibo:hasMeasurand unibo:PRESENCE }
```

4.2. Test Bed

Both the SIB instances (OSGi and RedSIB) run on the same server machine (12 CPUs Intel(R) Xeon(R) CPU E5-2430 v2 @ 2.50GHz 6 Cores, 64 GB RAM, Ubuntu Server 15.04) on a dedicated Virtual Machine (Ubuntu 12.04 LTS, 32 GB RAM, 8 CPUs) while the KPs are written using the Python API and run on a Samsung RC530 PC (CPU Intel i7-2670QM eight core 2.2 GHz, 8 GB RAM, Linux Mint 17 Qiana). The RedSIB is tested using both hash-tables in RAM and Virtuoso. Instead, the OSGi is evaluated using Jena running in RAM. KPs are connected to the SIB through a 100Mbps LAN connection. In all the tests (except for the ones on the notification latency of subscriptions and persistent updates), the time has been measured from the start of a KP request to the delivery of a reply. Timing components can be expressed as:

$$T = T_{request} + T_{elaboration} + T_{reply}$$

where $T_{request}$ represents the time spent by the KP to build an SSAP message and transmit it to the SIB, $T_{elaboration}$ is the time required by the SIB to elaborate the request, T_{reply} is the time interval required to send the response message from the SIB to the KP, including the time spent by the KP to parse the message.

4.3. Insert Time Evaluation

The first test measures the time required to insert a block of *n* lamp-posts (i.e., with *n* in the range [1..100]). Taking into account that a lamp-post is represented by 42 RDF triples, the number of inserted RDF triples varies from 42 to 4,2K. Insert time is a relevant metric to compare SIB performance because it is almost independent from the RDF store content. Moreover, insertions are quite common in the IoT (e.g., a new sensor node joins a Wireless Sensor Network), so the tests results will reflect SIB responsiveness in real conditions, not limited to lab environment. When the number of triples to insert grows, both $T_{request}$ and $T_{elaboration}$ increase, since a longer message will be built and sent, a longer time is needed for the parsing phase and a higher number of triples must be inserted. T_{reply} is not affected by the block size because, according to the SSAP specifications, the reply is always the same (i.e., a confirmation message) and does not depend on the number of inserted triples. The results (see Figure 4) show that the OSGi SIB outperforms the other two SIB implementations and this is more evident as the number of inserted lamp-posts increases. The gap with the RedSIB with Virtuoso is clear and it can be expected as it uses a persistent on-disk store.

To evaluate the scalability with the number of active subscriptions, the test is performed also in presence of active subscriptions (i.e., from 10 to 100 with steps of 10 subscriptions), where none of these is triggered. Figure 5 shows how all the SIB implementations scale very well with the number of active subscriptions and, at the same time, are not affected by the size of the update (i.e., 1, 10, 100 lamp-posts). The RedSIB Virtuoso trend with 100 lamp-posts is not shown in Figure 5 just because of readability reasons as its values span from 5673 ms, with 10 subscriptions, to 5840 ms, with 100 subscriptions. This means that also RedSIB with Virtuoso scales well with the number of active subscriptions.

4.4. Subscriptions Notification Latency Evaluation

Subscriptions represent a relevant feature of the SIB which allows reactivity to context changes and reduce network traffic by avoiding unnecessary polling. At the same time, subscriptions grant KPs to be notified of any relevant changes in the RDF store allowing them to keep track of the context. As drawback, subscriptions require resources and processing time to be executed. A SIB implementation should be characterised by a negligible impact on performance due to subscriptions. More precisely, subscriptions require, in principle, an unavoidable effort (i.e., required for every operation which modifies the RDF store) to check if an RDF store update may trigger a notification and, if this

Figure 4. Time to insert a block of n lamp-posts without any active subscription or persistent update

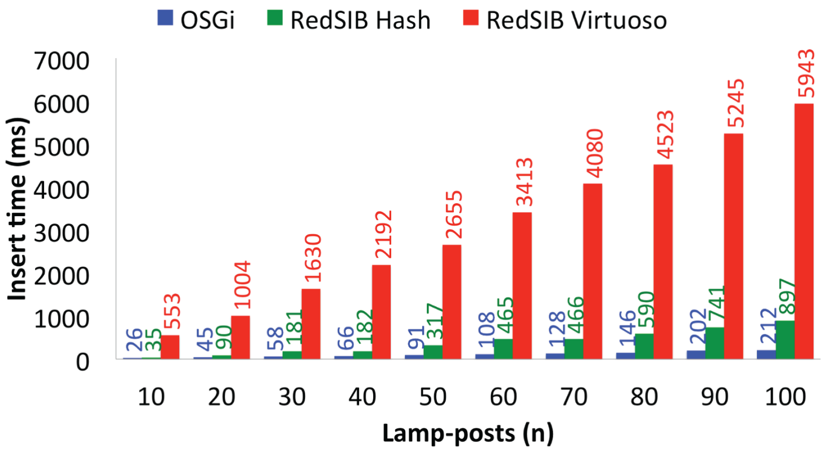
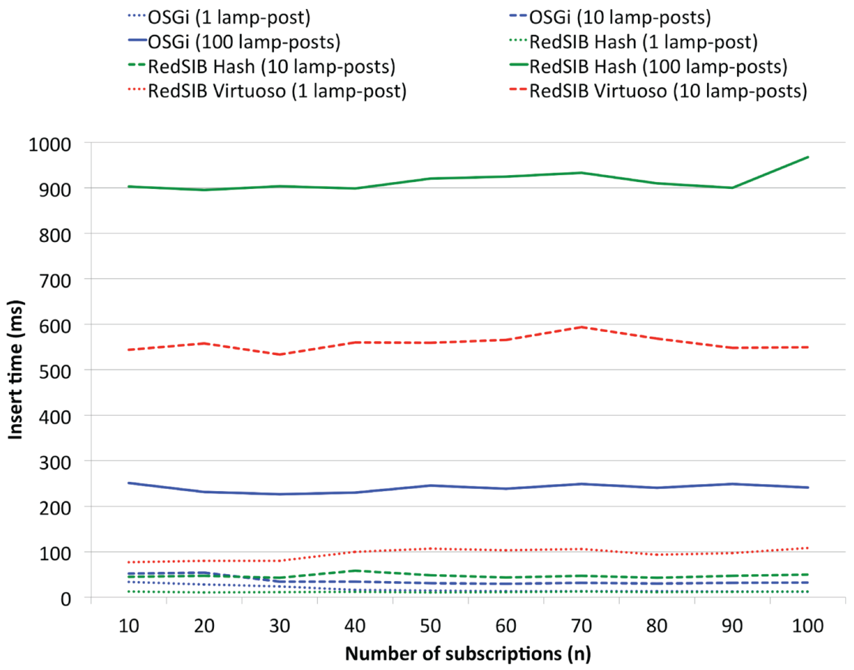


Figure 5. Time to insert a block of 1, 10, 100 lamp-posts with n active subscriptions



happens, an additional effort to determine the content of notification. We evaluated the former effort in Section 4.3, where the scalability with the number of active subscriptions (i.e., not triggered) has been proved. Instead, the following test aims at measuring the latency of notifications. An increasing number of subscriptions (see S1 in Section 4.1) is spanned over roads of different sizes (see Table 2).

Several updates of the dimming value of a single lamp (see U1 in Section 4.1) always triggering the same subscription (i.e., if the update changes the dimming value of LAMP_X_Y only the subscription to ROAD_X is triggered) are performed and the average latency taken from when the

Table 2. Subscriptions distribution

	Road Size				Total Number of S1 Subscriptions
	Very Small	Small	Medium	Large	
Number of S1 subscriptions	3	3	3	1	10
	6	6	6	2	20
	9	9	9	3	30
	12	12	12	4	40
	15	15	15	5	50
	18	18	18	6	60
	21	21	21	7	70
	24	24	24	8	80
	27	27	27	9	90
	30	30	30	10	100

updates are issued to when the corresponding notifications are received (on the KP side) is measured. The measure is repeated four times considering each time, as triggered subscription, a subscription to a road of different size (i.e., very small, small, medium, large).

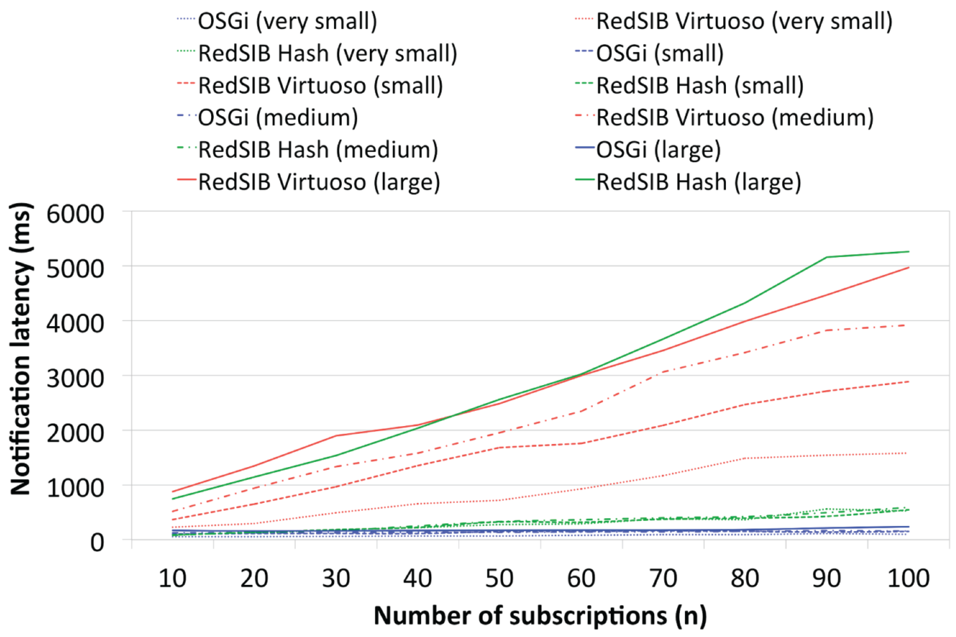
As shown in Figure 6, the OSGi SIB outperforms the other two SIB implementations. If the number of active subscriptions is less than 40, no difference can be appreciated between the OSGi and RedSIB hash (excluding the case of RedSIB hash with the triggered subscription on a large road). The RedSIB Virtuoso performance, using a persistent store, performs well if the triggered subscription is on a very small road. Instead, considering the large road case, the RedSIB performance decrease both using hash-tables and Virtuoso as RDF store.

4.5. Persistent Updates Evaluation

A naive implementation of PUs consists in letting the PU bundle make his business logic for every operation implying a RDF store modification. This means that, for every insertion or removal, all the PUs must be executed at least once (excluding PUs firing other PUs) even if no modifications to KB will be performed after the execution of all the PUs. This implementation lead to evident performance problems because the SIB is a platform to share context data among many interacting entities and it should be responsive and efficient. PUs must be executed only when necessary (i.e., when the conditions in the WHERE clause change). For this reason, we proposed an optimization of the naive PU execution which, similarly to what happens with subscriptions, makes a pre-analysis of the incoming updates in order to filter the PUs that are not sensitive to the triples modified by such updates. This optimization makes it possible to have many PUs active at the same time with an impact on performance only due to triggered PUs and that should be negligible, or at least acceptable, if the RDF store modifications does not trigger any PU.

Given the previous considerations, like for the subscriptions, also for the PUs is possible to state that in principle they imply an unavoidable effort to check if updates have to be performed and an

Figure 6. Notification latency versus number of active subscriptions



additional effort to be performed only if the PU is triggered by some modification on the KB. Figure 7 shows the unavoidable impact evaluated inserting a different number of lamp-posts (i.e., 1 or 10) and increasing the number of active persistent updates, where none of them is triggered.

The second additional component depends on the RDF store content, on the PU query and on the SPARQL engine used (i.e., Apache Jena). To evaluate the effort needed when a PU is triggered, we measure the notification latency of a subscription triggered by that PU. The test considers the persistent update PU1, the update U2 and the subscription S1 (see Section 4.1). When the presence sensor Y of a lamp-post within the road X detects a presence, PU1 is activated, it turns on all the lamps of road X and subscription U1 is triggered. The test measures the elapsed time from the instant when the presence sensor updates its data to the instant when a notification of S1 is received. The test is performed several times and average latencies are calculated considering four roads of different size (i.e., very small, small, medium and large).

Results analysis shows a not negligible impact of not triggered PUs on insert time: this impact must be taken into consideration and reduced as much as possible in future releases as it limits the usage of more PUs at the same time. Figure 8 instead shows a good performance of a single triggered PU. In this case, the latency increases with the road size mainly because of the dimension of the notification: a very small road notification produces up to 10 results, while a large road notification produces up to 100 results.

4.6. SP^{2}B Benchmark Evaluation

Schmidt et al. in (Schmidt et al., 2009) propose a SPARQL performance benchmark based on the DBLP scenario and composed by a set of seventeen queries with different selectivity, query size and output size. DBLP (http://dblp.uni-trier.de) is a web service offered by the University of Trier and Schloss Dagstuhl that provides bibliographic information about computer science journals and

Figure 7. Time to insert a block of 1 or 10 lamp-posts with *n* active persistent updates

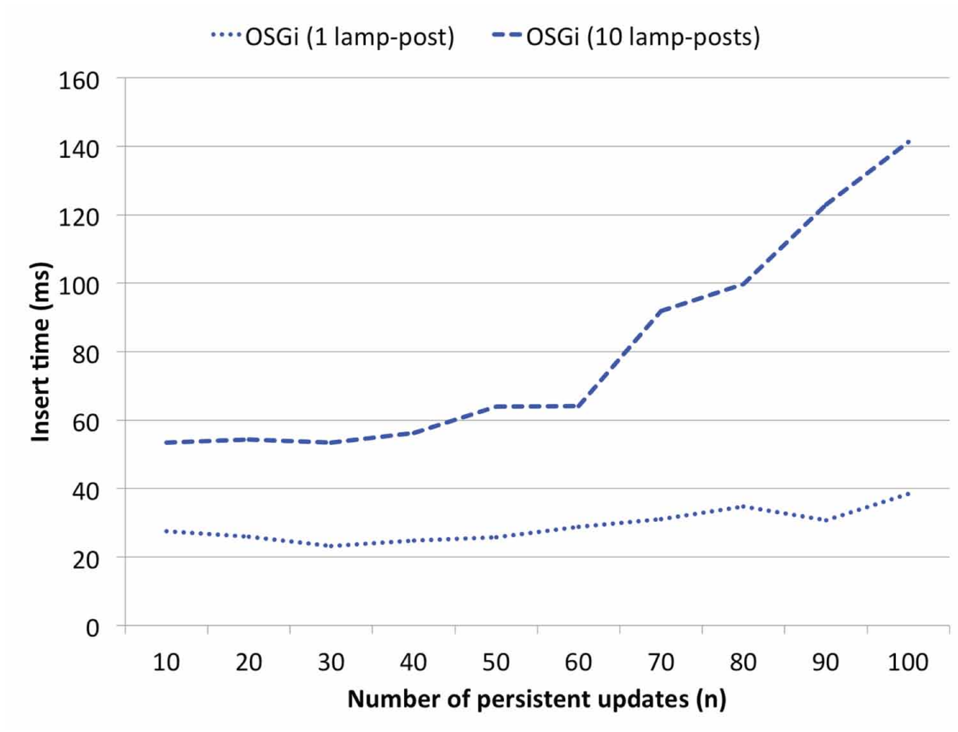
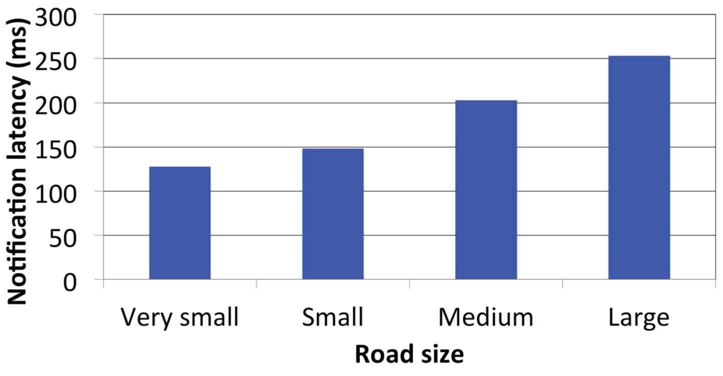


Figure 8. Time to receive a notification fired by a persistent update subscribed to roads of different size



proceedings. This benchmark, called SP^{2}B (SPARQL Performance Benchmark) has been specifically designed for the SPARQL evaluation, so the proposed queries address language specific issues. The benchmark is provided with a data generator that produces an N3 file with *n* triples, where *n* can vary from a minimum of 10k up to a maximum of 25M triples. Table 3 report the average time elapsed to perform the queries by the OSGi SIB. Such queries have been evaluated with a data set composed respectively by 10K and 50K triples. The first column in Table 3 contains the query identifier while for the two data sets both the expected number of results and the query time are reported (queries Q12a, Q12b and Q12c are SPARQL ASK queries, so they only expect result is True or False).

Table 3. SP^{2}B query times with 10K and 50K triples data sets

Query	10K Triples Data Set		50K Triples Data Set	
	Query Time (s)	Expected Results	Query Time (s)	Expected Results
Q1	0.022	1	0.011	1
Q2	0.219	147	0.807	965
Q3a	0.125	846	0.401	3647
Q3c	0.014	0	0.028	25
Q3b	0.010	9	0.021	0
Q4	9.181	23226	>300	104746
Q5a	0.774	155	22.569	1085
Q5b	0.324	155	10.684	1085
Q6	0.623	229	24.707	1769
Q7	0.146	0	5.183	2
Q8	0.049	184	0.062	264
Q9	0.025	4	0.041	4
Q10	0.045	166	0.116	307
Q11	0.021	10	0.020	10
Q12a	0.009	True	0.027	True
Q12b	0.014	True	0.009	True
Q12c	0.007	True	0.009	True

Analyzing the benchmark and the results it is possible to state that the query engine used (i.e., the ARQ libraries in the Jena framework) are adequate as all the queries have been answered correctly in a finite time. $T_{request}$ is almost the same for each query, since the differences related to the length of the queries can be neglected. The relevant time components here are $T_{elaboration}$ and T_{reply} respectively influenced by the complexity of the query and by the number of results.

5. CONCLUSION

In this paper we presented an OSGi based Semantic Information Broker, the core component of the Smart-M3 interoperability platform. The main motivations are based on the interest that the research and industrial community demonstrated on the Smart-M3 platform and on the emerging IoT and M2M scenarios. Besides making a solution fully compatible with existing legacy applications based on the RedSIB implementation, the OSGi SIB includes a new and expressive primitive, the SPARQL Persistent Update, which allows to integrate rules into Smart-M3 applications. We have also presented a specific SIB implementation for the Android OS conceived to increase the number of envisioned scenarios, including also those where interoperable data and services are required on mobile devices. This demonstrated the portability of the OSGI SIB solution. With reference to the proposed benchmarks, the evaluation revealed good performance both for insert and query operations. The OSGi SIB scales with the number of active subscriptions, granting at the same time a low latency on notifications.

As stated in (Bellavista, Corradi, Fanelli et al., 2012; Gubbi, Buyya, Marusic et al., 2013; Perera, Zaslavsky, Christen et al., 2014; Suo, Wan, Zou et al., 2012), security in context aware systems is still an open issue. Future work on the OSGi SIB will focus on support for security mechanisms in order to provide confidentiality, integrity and availability of data (the so called CIA Triad). This task will be achieved through the development of a proper set of bundles implementing a security infrastructure that exploits Semantic Web technologies to define powerful, efficient and effective access control methods.

REFERENCES

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., & Steggles, P. (1999). Towards a better understanding of context and context-awareness. In Handheld and ubiquitous computing (pp. 304–307). doi:10.1007/3-540-48157-5_29

Apache Clerezza. (n. d.). Retrieved from https://clerezza.apache.org/

Barnaghi, P., Wang, W., Henson, C., & Taylor, K. (2012). Semantics for the Internet of Things: Early Progress and Back to the Future. *International Journal on Semantic Web and Information Systems*, *8*(1), 1–21. doi:10.4018/jswis.2012010101

Beckett, D. (2002). The design and implementation of the Redland RDF application framework. *Computer Networks*, *39*(5), 577–588. doi:10.1016/S1389-1286(02)00221-9

Bedogni, L., Bononi, L., Di Felice, M., D'Elia, A., Mock, R., & Montori, F. … others. (2013). An interoperable architecture for mobile smart services over the internet of energy. *Proceedings of the 2013 IEEE 14th International Symposium and Workshops on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)* (pp. 1–6).

Bellavista, P., Corradi, A., Fanelli, M., & Foschini, L. (2012). A survey of context data distribution for mobile ubiquitous systems. *ACM Computing Surveys*, *44*(4), 24. doi:10.1145/2333112.2333119

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American*, *284*(5), 28–37. doi:10.1038/scientificamerican0501-34 PMID:11341160

Bizer, C., & Schultz, A. (2009). The Berlin SPARQL Benchmark. *International Journal on Semantic Web and Information Systems*, *5*(2), 1–24. doi:10.4018/jswis.2009040101

Chen, H., Finin, T., & Joshi, A. (2005). *Semantic web in the context broker architecture*.

Chen, H., Finin, T., Joshi, A., Kagal, L., Perich, F., & Chakraborty, D. (2004). Intelligent agents meet the semantic web in smart spaces. *IEEE Internet Computing*, *8*(6), 69–79. doi:10.1109/MIC.2004.66

Chirita, P.-A., Idreos, S., Koubarakis, M., & Nejdl, W. (2004). Publish/subscribe for rdf-based p2p networks. In The Semantic Web: Research and Applications (pp. 182–197). Springer.

D'Elia, A., Roffia, L., Zamagni, G., Vergari, F., Toninelli, A., & Bellavista, P. (2010). Smart applications for the maintenance of large buildings: How to achieve ontology-based interoperability at the information level. *Proceedings of the IEEE Symposium on Computers and Communications* (pp. 1077–1082).

Della Valle, E., Ceri, S., van Harmelen, F., & Fensel, D. (2009). Its a Streaming World! Reasoning upon Rapidly Changing Information. *IEEE Intelligent Systems*, *24*(6), 83–89. doi:10.1109/MIS.2009.125

Erling, O., & Mikhailov, I. (2009). RDF Support in the Virtuoso DBMS. In *Networked Knowledge-Networked Media* (pp. 7–24). Springer. doi:10.1007/978-3-642-02184-8_2

Eteläperä, M., Kiljander, J., & Keinänen, K. (2011). Feasibility Evaluation of M3 Smart Space Broker Implementations. *Proceedings of the 2011 IEEE/IPSJ 11th International Symposium on Applications and the Internet (SAINT)* (pp. 292–296). doi:10.1109/SAINT.2011.56

Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM Computing Surveys*, *35*(2), 114–131. doi:10.1145/857076.857078

Garcìa-Castro, R. et al.. (2013). *Web Semantics: Science, Services and Agents on the World Wide Web, Special Issue on Evaluation of Semantic Technologies* (Vol. 21). Elsevier.

Gómez-Pimpollo, J. F., & Otaolea, R. (2010). Smart Objects for Intelligent Applications-ADK. *Proceedings of the 2010 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 267–268). doi:10.1109/VLHCC.2010.52

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*(7), 1645–1660. doi:10.1016/j.future.2013.01.010

Guo, Y., Pan, Z., & Heflin, J. (2005). LUBM: A Benchmark for OWL Knowledge Base Systems. *Web Semantics: Science, Services, and Agents on the World Wide Web*, *3*(2-3), 158–182. doi:10.1016/j.websem.2005.06.005

Guo, Y., Qasem, A., Pan, Z., & Heflin, J. (2007). A Requirements Driven Framework for Benchmarking Semantic Web Knowledge Base Systems. *Knowledge and Data Engineering*. *IEEE Transactions on*, *19*(2), 297–309. doi:10.1109/TKDE.2007.19

Gyrard, A., Bonnet, C., & Boudaoud, K. (2014, March). Enrich machine-to-machine data with semantic web technologies for cross-domain applications. *Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT)* (pp. 559-564). IEEE. doi:10.1109/WF-IoT.2014.6803229

Hamza, H. S., Enas Ashraf, A. K., Nabih, M. M. A., Ahmed, M., Alaa, S., Hosny, K., … Attallah, A. (2014). SALE--An Innovative Platform for Semantically Enriching Next Generation Advertising Services. *Social Media and Publicity*, 53.

Honkola, J., Laine, H., Brown, R., & Tyrkko, O. (2010). Smart-M3 information sharing platform. Proceedings of the IEEE symposium on Computers and Communications (pp. 1041–1046). doi:10.1109/ISCC.2010.5546642

Jena, A. (2007). Semantic web framework for Java.

Jennings, C., Arkko, J., & Shelby, Z. (2012). *Media types for sensor markup language*. SENML.

Kiljander, J., Morandi, F., & Soininen, J.-P. (2012). Knowledge sharing protocol for smart spaces. *International Journal of Advanced Computer Science and Applications*, *3*(9). doi:10.14569/IJACSA.2012.030915

Kjær, K. E., & Hansen, K. M. (2010). Modeling and Implementing Ontology-Based Publish/Subscribe Using Semantic Web Technologies. *Proceedings of the 2010 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)* (pp. 63–71).

Korzun, D. G., Galov, I. V., Kashevnik, A. M., Shilov, N. G., Krinkin, K., & Korolev, Y. (2011). Integration of Smart-M3 applications: Blogging in smart conference. In Smart Spaces and Next Generation Wired/Wireless Networking (pp. 51–62). Springer.

Lamorte, L., Licciardi, C. A., Marengo, M., Salmeri, A., Mohr, P., & Raffa, G., … Cinotti, T. S. (2007). A platform for enabling context aware telecommunication services. *Proceedings of theThird Workshop on Context Awareness for Proactive Systems*, Guildford, UK.

Le-Phuoc, D., Parreira, J. X., & Hauswirth, M. (2012). *Linked stream data processing*. Springer. doi:10.1007/978-3-642-33158-9_7

Le Tu'n, A., Quoc, H. N. M., Serrano, M., Hauswirth, M., Soldatos, J., Papaioannou, T., & Aberer, K. (2012). Global sensor modeling and constrained application methods enabling cloud-based open space smart services. *Proceedings of the 2012 9th International Conference on Ubiquitous Intelligence & Computing and the 9th International Conference on Autonomic & Trusted Computing (UIC/ATC),* (pp. 196–203).

Lomov, A. A., Vanag, P. I., & Korzun, D. G. (2011). Multilingual ontology library generator for Smart-M3 application development.*Proc. 9th Conf. of Open Innovations Framework Program FRUCT and 1st Regional MeeGo Summit Russia--Finland* (pp. 82–91).

Manzaroli, D., Roffia, L., Cinotti, T. S., Azzoni, P., Ovaska, E., Nannini, V., & Mattarozzi, S. (2010). Smart-M3 and OSGi: The interoperability platform. *Proceedings of the 2010 IEEE Symposium on Computers and Communications (ISCC)* (pp. 1053–1058).

Masuoka, R., Parsia, B., & Labrou, Y. (2003). Task computing--the semantic web meets pervasive computing. In *The Semantic Web-ISWC 2003* (pp. 866–881). Springer. doi:10.1007/978-3-540-39718-2_55

Morandi, F., Roffia, L., D'Elia, A., Vergari, F., & Cinotti, T. S. (2012). RedSib: a Smart-M3 semantic information broker implementation.*Proc. 12th Conf. of Open Innovations Association FRUCT and Seminar on e-Tourism* (pp. 86–98).

Murth, M., & Kühn, E. (2009a). A heuristics framework for semantic subscription processing. In The Semantic Web: Research and Applications (pp. 96–110). Springer. doi:10.1007/978-3-642-02121-3_11

Murth, M., & Kühn, E. (2009b). Knowledge-based coordination with a reliable semantic subscription mechanism. *Proceedings of the 2009 ACM symposium on Applied Computing* (pp. 1374–1380). doi:10.1145/1529282.1529588

Murth, M., & Kuhn, E. (2010): A Semantic Event Notification Service for Knowledge-Driven Coordination. *Proc. of 1st Int'l. workshop on emergent semantics and cooperation in open systems (ESTEEM), cooperation with the 2nd Int'l. Conf. on Distributed Event-Based Systems (DEBS 2008), Rome, Italy*.

Murth, M., & Kuhn, E. (2010). Knowledge-based interaction patterns for semantic spaces. *Proceedings of the 2010 International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)* (pp. 1036–1043). doi:10.1109/CISIS.2010.31

Ovaska, E., Cinotti, T. S., & Toninelli, A. (2012). The Design Principles and Practices of Interoperable Smart Spaces. In Advanced Design Approaches to Emerging Software Systems: Principles, Methodologies and Tools (pp. 18–47). Hershey, PA, USA: IGI Global. doi:10.4018/978-1-60960-735-7.ch002

Palviainen, M., & Katasonov, A. (2011). Model and ontology-based development of smart space applications. In *Pervasive Computing and Communications Design and Deployment: Technologies, Trends, and Applications*, 126–149.

Pantsar-Syväniemi, S., Ovaska, E., Ferrari, S., Cinotti, T. S., Zamagni, G., & Roffia, L. … Nannini, V. (2011). Case study: Context-aware supervision of a smart maintenance process. *Proceedings of the 11th IEEE/IPSJ International Symposium on Applications and the Internet SAINT '11* (pp. 309–314).

Pellegrino, L., Baude, F., & Alshabani, I. (2012). Towards a scalable cloud-based rdf storage offering a pub/sub query service. *Proceedings of theInternational conference on Cloud Computing, GRIDs, and Virtualization* (pp. 243–246).

Pellegrino, L., Huet, F., Baude, F., & Alshabani, A. (2013). A distributed publish/subscribe system for RDF data. In Data Management in Cloud, Grid and P2P Systems (pp. 39–50). Springer.

Perera, C., Zaslavsky, A., Christen, P., & Georgakopoulos, D. (2014). Context aware computing for the internet of things: A survey. *IEEE Communications Surveys and Tutorials*, *16*(1), 414–454. doi:10.1109/SURV.2013.042313.00197

Pfisterer, D., Römer, K., Bimschas, D., Kleine, O., Mietz, R., Truong, C., … others. (2011). SPITFIRE: toward a semantic web of things. *Communications Magazine, 49*(11), 40–48.

Roffia, L., Morandi, F., Kiljander, J., Elia, A. D., Vergari, F., Viola, F., ... & Salmon Cinotti, T. (2016). A Semantic Publish-Subscribe Architecture for the Internet of Things.

Schmidt, M., Hornung, T., Lausen, G., & Pinkel, C. (2009). SP^ 2Bench: a SPARQL performance benchmark. *Proceedings of the IEEE 25th International Conference on Data Engineering ICDE '09* (pp. 222–233).

Shi, D., Yin, J., Li, Y., Qian, J., & Dong, J. (2007). An RDF-Based Publish/Subscribe System. *Proceedings of the Third International Conference on Semantics, Knowledge and Grid* (pp. 342–345).

Skovronski, J. (2007). *An ontology-based publish-subscribe framework*. ProQuest.

Song, Z., Cardenas, A., & Masuoka, R. et al. (2010). Semantic middleware for the Internet of Things. In *Internet of Things* (pp. 1–8). IOT. doi:10.1109/IOT.2010.5678448

Suo, H., Wan, J., Zou, C., & Liu, J. (2012). Security in the internet of things: a review. *Proceedings of the 2012 International Conference on Computer Science and Electronics Engineering (ICCSEE)* (Vol. 3, pp. 648–651). doi:10.1109/ICCSEE.2012.373

Thomson, G., Bianco, S., Ben Mokhtar, S., Georgantas, N., & Issarny, V. (2008). Amigo aware services. In *Constructing Ambient Intelligence* (pp. 385–390). Springer. doi:10.1007/978-3-540-85379-4_43

Vazquez, J. I., de Ipiña, D., & Sedano, I. (2007). Soam: A web-powered architecture for designing and deploying pervasive semantic devices. *International Journal of Web Information Systems*, *2*(3/4), 212–224. doi:10.1108/17440080780000301

Vergari, F., Bartolini, S., Spadini, F., D'Elia, A., Zamagni, G., Roffia, L., & Cinotti, T. S. (2010). A smart space application to dynamically relate medical and environmental information.*Proceedings of the Conference on Design, Automation and Test in Europe* (pp. 1542–1547). doi:10.1109/DATE.2010.5457056

Vergari, F., Cinotti, T. S., DElia, A., Roffia, L., Zamagni, G., & Lamberti, C. (2011). An integrated framework to achieve interoperability in person-centric health management. *International Journal of Telemedicine and Applications*, *2011*, 5. doi:10.1155/2011/549282 PMID:21811499

Viola, F. (2016, April). A Modular Lightweight Implementation of the Smart-M3 Semantic Information Broker. *Proceedings of the 2016 18TH Conference of Open Innovations Association (FRUCT)* (pp. 370-377). IEEE. doi:10.1109/FRUCT-ISPIT.2016.7561552

Wang, J., Jin, B., & Li, J. (2004). An ontology-based publish/subscribe system.*Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware* (pp. 232–253).

Wang, X., Dong, J. S., Chin, C. Y., Hettiarachchi, S. R., & Zhang, D. (2004). Semantic Space: An infrastructure for smart spaces. *Pervasive Computing, 3*(3), 32–39. doi:10.1109/MPRV.2004.1321026

Alfredo D'Elia studied in University of Bologna where he obtained the master degree summa cum laude in 2006. After the degree he continued his activity in University of Bologna, obtaining a PhD in information technology in 2012 and then he worked as a research assistant. He worked on several European projects like SOFIA, CHIRON, Internet of Energy and Arrowhead. He also collaborated with important industries like Telecom Italia and he was Intern in Nokia for six months where he was a co-inventor of a Patent. He also teaches at University of Bologna and Cesena as assistant or responsible of didactic modules. The main topics of interest in his research are Semantic Web, interoperability, information representation techniques, software architecture and system characterization and optimization.

Fabio Viola graduated in information engineering in 2011 (University of Salento, Italy) and received his master's degree in computer science engineering in 2014 (University of Bologna, Italy). Since November 2014 he has been working as an assistant researcher at the University of Bologna, ARCES (Advanced Research Center on Electronic Systems). The research focus is on semantic technologies for the Internet of Things.

Luca Roffia received an MSc in Computer Science Engineering in 2001 (Videogame Engines and 3D Interactive Graphics) and a Ph. in Electronics, Computer Science and Telecommunication Engineering in 2005 (Context Related Information Sharing and Retrieval in Mobile Cultural Heritage) from the University of Bologna (Italy). He is currently an Adjunct Professor at the University of Bologna in charge of courses on Computer Architecture (School of Engineering and Architecture) and Information Technology for Arts Organization (School of Economics, Management and Statistics). His current research interests are mainly focused on enabling semantic interoperability in the Internet of Things for the technology transfer in the areas of agricultural, smart city and cultural heritage.

Paolo Azzoni is Research Program Manager at Eurotech Group. He is responsible for the international research projects and his main working areas include machine-to-machine (M2M) distributed systems, device to cloud architectures and solutions, semantic M2M and Internet of Things (IoT). Previously, he was involved in academic research and teaching activities in the areas of formal verification, hw/sw co-design and co-simulation for embedded systems and microprocessors. In 2006 he joined ETHLab (Eurotech Research Center) as Research Project Manager and he has been responsible for international research projects in the area of embedded systems. He holds a Master Degree in Computer Science and a second Master Degree in Artificial Intelligence both from the University of Verona.

Tullio Salmon Cinotti graduated in electrical engineering at the University of Bologna in 1974. He is Associate Professor at the School of Engineering and Architecture of the University of Bologna, and he is in charge of courses on computer architecture, logic design and interoperability of embedded systems. His research interest is focused on embedded systems and semantics based data distribution architectures for cyber-physical systems. He is co-author of researchers from Intel Labs, Nokia Research, Siemens Corporate Technology, Telecom Italia Lab, VTT, Polytechnic of Milano, and University of Kent. Prof. Salmon Cinotti is co-ordinator of the University of Bologna participation to European research initiatives in the areas of open cultural heritage, smart environments and electric mobility. He is currently Director of ARCES, an inter-department research center of the University of Bologna.