The M3 Architecture for Smart Spaces: Overview of Semantic Information Broker Implementations

Fabio Viola[1], Alfredo D'Elia[1], Dmitry Korzun[2], Ivan Galov[2], Alexey Kashevnik[3,4], Sergey Balandin[5]
[1]University of Bologna, Bologna, Italy
[2]Petrozavodsk State University (PetrSU), Petrozavodsk, Russia
[3]SPIIRAS, St.Petersburg, Russia
[4]ITMO University, St.Petersburg, Russia
[5]FRUCT Oy, Helsinki, Finland
{fabio.viola2, alfredo.delia4}@unibo.it, {dkorzun, galov}@cs.karelia.ru, alexey@iias.spb.su, sergey.balandin@fruct.org

***Abstract*—A smart space enhances a networked computing environment by enabling information sharing for a multitude of local digital devices and global resources from the Internet. We consider the M3 architecture (multi-device, multi-vendor, multi-domain) for creating smart spaces, which integrates technologies from two innovative concepts: the Semantic Web and the Internet of Things. Our research focus is on analyses of the capabilities of Smart-M3 platform, which provides software implementations for such a central element of an M3 smart space as Semantic Information Broker (SIB). The paper presents a state-of-the-art and contributes our systematized vision on the SIB design and implementation. The analyzed open source SIB implementations include the original Smart-M3 piglet-based SIB, its optimized descendant RedSIB, OSGi SIB for Java devices, pySIB for Python devices, and CuteSIB for Qt devices. We also analyze the design of proprietary or incomplete SIB implementations: RIBS for embedded devices and ADK SIB built upon the OSGi framework with integration in the Eclipse Integrated Development Environment. The theoretical study is augmented with experimental evaluation of available SIB implementations.**

I. INTRODUCTION

Smart spaces form a programming paradigm, which is now augmented with the rapidly advancing suit of information and communication technologies (ICT), for creating a certain class of ubiquitous computing environments [1], [2], [3], [4], [5]. Such an environment is typically associated with a physical spatial-restricted place (office, room, home, city square, etc.) equipped with a variety of devices (sensors, data processors, actuators, consumer electronics, personal mobile devices, multimodal systems, etc.). In addition to local networking, the environment has access to the global Internet with its diversity of information services and computational resources. The key postulate of a smart space is to enable information sharing in the environment, supporting construction of "smart" services [6], [7], [8], [9], where the term "smart" emphasizing the new level of service recognition (detection of user needs), construction (automated preprocessing of large and multisource data amounts), and perception (derived information provision to the user for decision-making).

The smart spaces ICT suit is based on the disruptive technologies coming from two innovative concepts: the Semantic Web (SW) and the Internet of Things (IoT). The SW concept was born to drive the Web towards the original Tim Berners Lee's vision, the so-called web of data [10]. The SW technology stack is primarily composed by technologies allowing the representation (RDF, RDFS, OWL) and retrieval (SPARQL) of semantically annotated data [11]. The IoT concept [12] is a large-scale evolution of the innovative vision of Mark Weiser about ubiquitous computing: the Internet, in addition to personal desktops and mobile computers, is also populated with billions of heterogeneous interconnected smart devices, which represent (and advance) physical things. Everyday life objects, alongside traditional computers, become data processors and service constructors to their users [13], [14]. Both SW and IoT form a vast research area characterized by a high interdisciplinary level, a high process dynamicity, and heterogeneity of the involved devices and applications. A very wide range of application domains is covered: from collaborative work environments and electronic health to cybermedicine, from electronic tourism and cultural heritage education to smart cities, from transport logistics and Industrial Internet to socio-cyber-physical systems, and many more.

This paper considers the M3 architecture, which represents a promising ICT suite for creating smart spaces. In particular, the Smart-M3 platform is one of the most suggestive examples of applying the SW technologies [15] to the case of ubiquitous computing and emerging IoT environments. The RDF and OWL standards are used to represent data and bind each resource with its meaning. The SPARQL UPDATE [16] and QUERY [17] languages are used to update and retrieve data from the shared information store, which constitutes a knowledge base (KB) for the environment. Through these technologies, Smart-M3 becomes a candidate middleware platform for hosting a wide range of context-aware applications based on ontology-driven and multi-agent approaches [18], [19], [20], [21], [22]. The context is here intended, as from the well-known Dey definition, as any information that can be used to characterize the situation of an entity, where the latter can be a person, a place or a physical or computational object [23]. Many applications have been developed in the latest years exploiting the Smart-M3 opportunities [5]: blogging [24], mobile tourist guiding [25],

ISSN 2305-7254

smart conference system [26], ridesharing service [27] just to name a few. The ongoing application development activity is also covering new directions like electro mobility [28], [29] and mobile robotics [30].

Our study is focused on the central component of the M3 architecture: the semantic information broker (SIB). Each SIB manages and shares a knowledge base (KB) with all the smart space participants. The KB is semantic, in the form of a RDF triplestore. Starting from 2008, when the first SIB prototype was produced, several SIB implementations have been appeared optimized for a specific purpose like portability and performance. In this paper, we provide a systematized view on the existing SIB designs and available SIB implementations. We review their architectures and design solutions, analyzing the main strengths and weaknesses of each version. The examined SIB implementations include the five open source projects: Piglet-based SIB [15], its optimized descendant RedSIB [31], OSGi SIB for Java-based systems [32], pySIB for embedded and resources constrained devices with Python [33], and CuteSIB [34] for Qt crossplatform device family. We also analyze the two SIB designs with no open source implementations: RIBS [35] for embedded devices and ADK SIB [36] built upon the OSGi framework and integrated in the Eclipse Integrated Development Environment. Recently the open source SIB development is supported mainly under the umbrella of the FRUCT Association by efforts of University of Bologna (Italy) and Petrozavodsk State University (Russia). The theoretical study is augmented with experimental evaluation and comparison of available SIB implementations.

The rest of the paper is organized as follows. Section II describes the essentials of M3 architecture and systemizes architectural and design solutions for a generic M3 SIB. Section III considers available SIB implementations and introduces internal details for each of them. Section IV provides experimental evaluation and comparison of the SIB implementations. Finally, Section V concludes the paper.

II. THE M3 ARCHITECTURE

The M3 architecture has been initially defined by a consortium participating to the Artemis JU funded SOFIA project (Smart Objects for Intelligent Applications) and to the Finnish nationally funded program DIEM (Device Interoperability Ecosystem), working in strong collaboration with the Nokia Corporation. M3 stands for Multi-device, Multi-vendor, and Multi-domain [1], [14], [37], [38]. The Smart-M3 platform [15] is an open source middleware that implements the M3 architecture. Smart-M3 was released as open source platform at the NoTA Conference on October 1, 2009. Soon after its first release, the Smart-M3 potential was understood and applied in other European projects, e.g., in eHealth and eMobility. Furthermore, EIT ICT Labs, an Innovation Factory for ICT Innovation in Europe, included smart spaces among its innovation areas. The platform was adopted by the smart space infrastructure recently established at the Helsinki Node of the EIT ICT Labs. At the moment, the main developers of Smart-M3 platform are several communities including the FRUCT Association, the SOFIA Community, and the ARCES (Advanced Research Center on Electronic Systems “Ercole De Castro”) at University of Bologna.

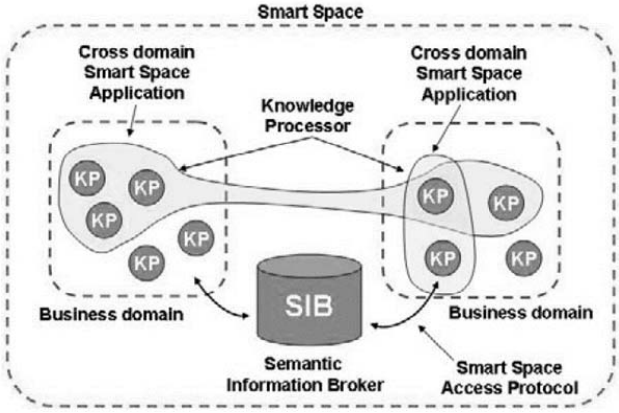

Fig. 1. The M3 architecture in a nutshell [15]

The M3 architecture (Fig. 1) defines two principal software components and an interaction protocol for them: the SIB, the Knowledge Processors (KPs) and the Smart Space Access Protocol (SSAP).

From a functional point of view, SIB implements an information hub forming a logical rendezvous and information-level interoperability infrastructure on the top of an RDF triple-store (or a SPARQL endpoint). Each SIB acts as an access point to a shared KB that describes the overall information state and context of the environment. The information representation is semantic, based on an oriented labeled graph, i.e., following the SW concept. The basic SIB role is to manage the read&write accesses to this graph. Advanced access operations are possible, including such persistent queries as subscription: a subscription notification mechanism to improve the reactivity and the band usage where the subscribe-notify paradigm is applicable.

The generic SIB architecture is shown in Fig. 2. It consists of several modules: network handler, request/response handler, operations handler and RDF triplestore. Network handler implements network communication between SIB and KPs. They exchange messages, which follows the SSAP rules and syntax, recently has being generalized to Knowledge Sharing Protocol (KSP) [15]. The SSAP is a communication protocol acting at application level and for which it exists a well supported encoding in XML and a younger, less supported, but thinner JSON serialization. Request/response handler process network messages according to SSAP/KSP protocol rules and syntax and determines which operations should be performed in triplestore. Protocols provides read-write operations for inserting, removing, updating, querying, and (un)subscribing. The set of operations can be extended with advanced SPARQL queries and persistent operations. Operations are performed in operation handlers using a particular triplestore library to manage information in the RDF triplestore.

Each KP is a software agent and a participant to the Smart-M3 based scenario. The way in which such an application scenario evolves and its intelligence is provided, is the cooperative knowledge processing over the shared data and

context, which is a much powerful approach with respect to an autonomous participation in the IoT environment [19].

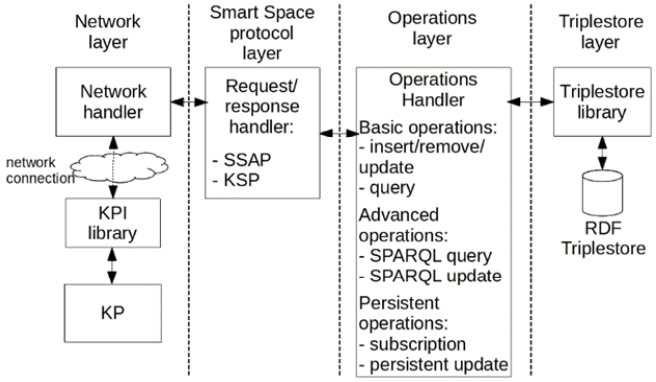

Fig. 2. Architecture of general SIB

Each KP acts as an agent of adjustable autonomy [39]. A KP can be developed exploiting a Knowledge Processor Interface (KPI) [20]. KPI enables KP to participate in the smart space through implementation of the SSAP client part: primitives for local manipulation of shared information, functions for security and reliability, notification management, serialization and de-serialization of the raw XML messages. Thanks to the success of the M3 architecture and the community efforts, many KPIs are currently available, including Python, C, Java, C#, Ruby, PHP, Javascript, Lua.

III. SEMANTIC INFORMATION BROKER IMPLEMENTATIONS

Let us consider architectures and design solutions of the following SIB implementations: the piglet-based SIB M3 SIB [15], RedSIB [32] for generic-purpose computers, OSGi SIB [32] for Java-based systems, PySIB [33] for a modular SIB supporting resources constrained devices and for didactics purposes, CuteSIB [34] for a wide spectrum of Qt-based IoT devices, RIBS [35] for resource limited devices operating with embedded sensors, and ADK SIB [36] built upon the OSGi framework with integration in the Eclipse Integrated Development Environment. Open source SIB implementations are released as open source software, see the SourceForge resource (https://sourceforge.net/projects/smart-m3/). At the moment, the SIB development is supported mainly under the umbrella of the FRUCT Association by efforts of University of Bologna (Italy) and Petrozavodsk State University (Russia).

A. The Piglet-Based SIB

This implementation has been first released in 2009. It was used by the FRUCT community in such application development projects as SmartConference (http://fruct.org/sc), SmartScribo (http://fruct.org/smartscribo), and Ridesharing (http://fruct.org/ridesharing).

The piglet-based SIB architecture is shown in Fig. 3. SIB consists of two main parts: the SIB daemon (sibd application written in C language with Glib library) and network handlers. SIB daemon handles the information access, operations processing and the storage of the RDF Graph. Network handlers maintain network communication with KPs. The Piglet SIB supports two communication technologies: TCP/IP and Nota implemented as a separate applications (sib-tcp and sib-nota respectively). They are connected to the SIB daemon over D-Bus.

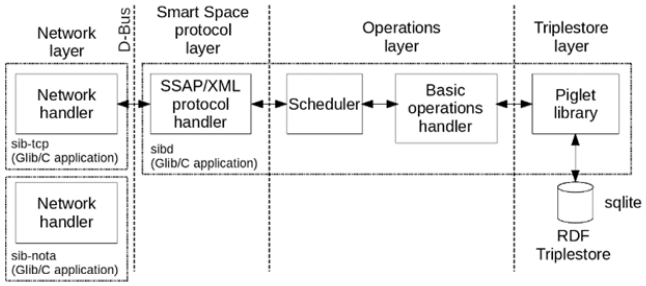

Fig. 3. The piglet-based SIB architecture

The architecture offered the opportunity to add new interfaces by implementing the corresponding daemons and connecting them to the D-Bus. The principles guiding the design of Smart-M3 are simplicity, extensibility and being agnostic to the used communication mechanisms.

The simplicity ensures scalability for small devices and for large number of users, while the extensibility makes it possible to tailor the implementation easily to uses where the standard functionality is not sufficient. Furthermore, by not dictating a specific communication mechanism, the Piglet SIB should be easy to deployed on top of many existing infrastructures.

The layer runs in a single thread which schedules and executes the requests from the threads handling the SSAP operations. The communication between the SSAP operations threads is handled by using asynchronous queues. The triple operations layer is currently implemented by using Piglet RDF store. The triple operations layer is not tied to any specific RDF store, and any RDF store supporting the basic operations of read, write and delete may be substituted in the place of Piglet. However, changing the RDF store will require changing the code in the graph operations layer to adapt to the concrete interface provided by the new RDF store.

B. RedSIB

RedSIB is a direct descendant of the Piglet SIB implementation. They share the same architectural design and the code is essentially inherited. RedSIB was built upon the experiences gained in the early Smart-M3 applications. The goal was to solve the most relevant issues the application developers detected as well as improving the performance and avoiding criticalities. Feedbacks of the Smart-M3 community were used to improve the SIB adding more functionalities.

At a high level of the abstraction, the RedSIB architecture (Fig. 4) is the same of the piglet-based SIB implementation with one RDF store and two main daemons communicating through D-Bus: the monolithic SIB daemon (redsibd application) and the TCP one (sib-tcp application). A deepened analysis highlights the presence of a high quantity of new code and data structures whose main functionalities are summarized, but not limited to, the following points.

- Support for Virtuoso and for volatile storage (previously supporting only the BDB RDF store).
- Prototype of data access control mechanism [40].

- Optimization of the subscription handling with several improvements acting specifically for those situations that were considered most common by the community [41]: shortly many subscriptions with low number of triples to be notified at a time.
- Management of the situation of abrupt disconnection happening when a subscribed KP temporarily loose connectivity and the SIB has internal bounds to the active subscriptions.

The RedSIB has been applied in many domains like maintenance [42], telemedicine [43] and computer-human interaction [44] to name a few. The efforts performed by the research community during the years revealed also some limits among which two led to the decision to create new versions. First, the RedSIB implementation is still bound to the D-Bus, an interprocess communication daemon. The D-Bus daemon limits performances for certain specific tasks. Second, the RedSIB implementation has a monolithic architecture, which requests huge efforts to the developers when new features are needed.

Being a result of the SW concept applied to IoT, which is still an experimental topic of the research scenario, many features and setups are to be implemented, deployed and prototyped and this conflicts with the monolithic approach of the code. Much efforts are still needed to optimize the subscription management or to include access control mechanisms. The features to be added and tested are in a queue that the community cannot dispose due to the lack of modularity. One issue is the lack of portability and extensibility, which leads to the need for many different modular version optimized for different target architectures. The described needs and the proliferation of SIB versions are not unexpected because in the overall vision the integration point is the SSAP protocol which remains unchanged. Therefore, coherently with the IoT concept, multi-SIB architectures with different SIB versions running on heterogeneous devices are possible and welcomed [45], [46].

C. OSGi SIB

The OSGi SIB [32] was created and is currently maintained by the University of Bologna and Eurotech. The focus of the OSGi SIB developers is on the IoT and M2M industrial domains. The main strength of the OSGi SIB is its portability: the Java programming language and the OSGi framework grant the ability to run on different operating systems. The development of the OSGi SIB led to the creation of a specific Android version of the Semantic Information Broker, suitable for mobile devices. With respect to the other implementations of the SIB, the OSGi SIB introduces a new primitive called Persistent Update (PU): it consists of a SPARQL 1.1 update executed once when the command is issued to the SIB and then acting persistently on the data-store until it is deactivated. Together with the Python lightweight implementation, the OSGi SIB is the only one providing support for the JSON encoding of the SSAP protocol which grants a bandwith usage ranging from the 60% to the 90% of the current XML encoding (still in its early stage). The OSGi SIB also provides support for persistent storage thanks to TDB module of the Jena libraries. The OSGi SIB is implemented as OSGi Java application and is made of several interacting modules – bundles registered to the OSGi framework (Fig. 5).

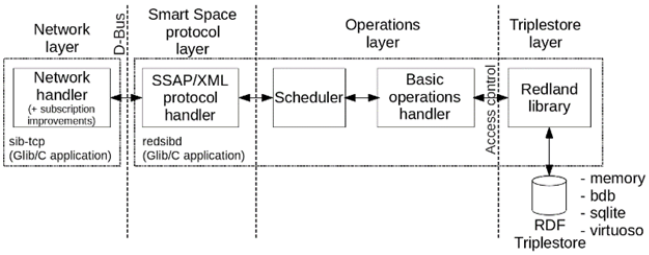

Fig. 4. The RedSIB architecture

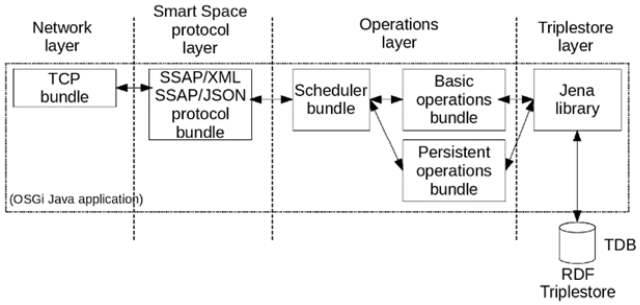

Fig. 5. The OSGi SIB Architecture

TCP Bundle is responsible for managing the network connection with KPs. It receives messages from KPs and manages a queue of the requests to be satisfied. Protocol bundle parses each message received from the TCP bundle in order to build an internal representation of the request. Scheduler bundle binds an identifier to each request processed by the Protocol bundle and sends request on processing. Operation bundles process each request with help of Jena library and provide a reply. Persistent operations bundle is responsible for the management of every active Persistent Update operation.

D. pySIB

Developed by ARCES department of the University of Bologna, pySIB is a lightweight SIB implementation designed to run mainly on embedded devices and System on Chips (SoCs). The implementation is written in Python and relying on the Python bindings of the RDFlib, pySIB results easy to install and run. As highlighted by Viola et al. [46] pySIB, despite being in its earliest stable releases, shows good performance both in updating the knowledge base and retrieving data from it.

The modular architecture of pySIB makes it easy for the developers to extend it by adding new features or replacing existing modules with different ones (e.g. to support a different SSAP parser). The architecture is represented in Fig. 6. The Network handler module constitutes the interface between the SIB and the external world. Every message received from the outside (currently over TCP) is forwarded to the Protocol handler that builds an internal dictionary represetation of the SSAP message. The current implementation supports by default the JSON encoded version of the SSAP protocol. Security manager checks access rights

on the requested operation and then passes approved operations to Operation handler.

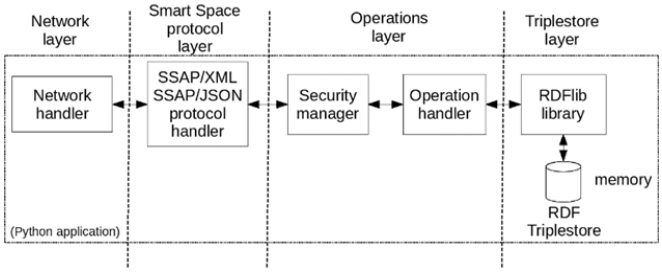

Fig. 6. pySIB internal architecture

Operation handler performs the actions required by the KP on the RDF store, then sends back a dictionary to the Protocol handler which transform it into a reply message. The Network handler module sends the reply packet to the KP. As triplestore pySIB uses RDFlib library which maintains in-memory volatile triplestore which is fast but not persistent. Due to the modular architecture and to the simplicity of Python, pySIB is also used for educational matters into the Interoperability of Embedded Systems course of the University of Bologna where Smart-M3 has a central role.

E. CuteSIB

The CuteSIB implementation is developed and maintained by Petrozavodsk State University (PetrSU). CuteSIB is a reengineered version of RedSIB. The implementation is based on the Qt framework in order to support a wide spectrum of Qt-based IoT devices. A modular SIB design was proposed [34] to support such important properties as extensibility, dependability, and portability.

The first distinctive property is elimination of D-Bus. One reason is that D-Bus is used only in Unix-based systems, thus preventing the use of SIB in other operating systems (e.g., Windows). Another reason is that D-Bus does not effectively support transfer of big amounts of data. Operation becomes unstable when transferring fast data streams of triples. As a result of the D-Bus elimination, the interprocess communication has simpler structure.

SIB communication modules for various network protocols (e.g., TCP or UDP) become plug-ins. They can be loaded/unloaded from the main SIB program as dynamic libraries. When higher portability is needed, such plug-ins can be integrated to SIB using static compilation. In this case, SIB does not load external libraries and is used as monolith application with the customizable set of network protocols. This feature targets SIB portability, taking into account devices with operating systems that have limited or no support of dynamic libraries.

The second distinctive property is the plug-ins based architecture in order to achieve higher extensibility due to the modular approach, see Fig. 7. The architecture allows inclusion/exclusion of certain modules in compilation phase or in runtime. The feature affords to customize the SIB functionality for given host device and IoT environment.

Network layer is implemented as a pool of access points, each is an external module for SIB. Protocol manager interacts with a specific access point and performs request parsing and response generation.

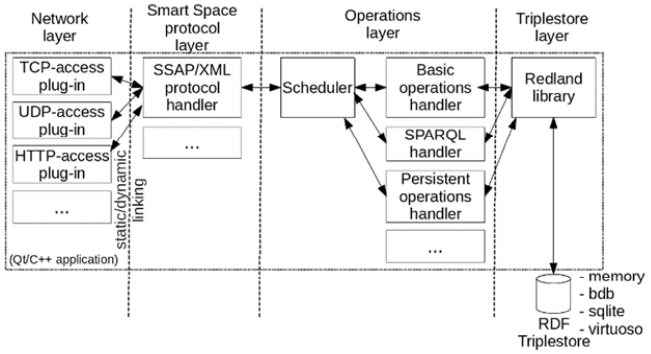

Fig. 7. The plug-ins based architecture of CuteSIB

Access protocol (such as SSAP or KSP) is implemented as a separate module, which parses request messages and creates response messages. In particular, it becomes possible to implement SPARQL over HTTP to access SIB as a common SPARQL access point. Scheduler module controls processing of CuteSIB commands with KPs requests/responses and internal notifications (to control runtime of other modules).

The scheduler delegates each command to an appropriate operation handler. Three command handlers can be distinguished: basic operation handler (for insert, remove, update, and query operations), persistent operations handler contains persistent operations (such as subscription), and SPARQL handler for advanced search queries. Persistent operations are always stored on the SIB side (continuous in time) and a response is generated whenever a specified event occurs.

F. RIBS and ADK SIBs: implementations are not available at the moment

Finally in this section, let us consider two SIB designs that had reached a considerable level of interest in the past, when the Smart-M3 platform was introduced. Although these SIBs have no open source implementation or their implementation status is unknown to the authors, a summary consideration is still important to contrast the ideas with the other SIB proposals.

The RDF Information Bases Solution (RIBS) [35] is a SIB design with the focus on security aspects and targeted to low-resources devices. The prototype unified in a pioneering way two of the main issues that are currently faced by the whole IoT community: the security based on a dynamic set of concurrent policies and the portability on resource-constrained devices. RIBS was born and developed during the SOFIA project (2008-2011) and led to a prototype demonstration in the final event of the project. Despite its good points, being not totally open source, the project failed to build a community of developers large enough to carry the work on after the end of SOFIA. Then, to the best of the authors' knowledge, the development process of RIBS was suspended.

The SIB ADK (Advanced Development Kit) [36] is a SIB version built upon the OSGi framework and integrated in the Eclipse Integrated Development Environment. It was designed

to have a powerful suite for ontology based code generation and model based application development [36]. It is possible to state that the ADK SIB and the frameworks based on it, approach to smart applications in a different but not clashing way with respect to the classical approach matured since the times of the Piglet SIB. Research work and interesting ideas derived from both the approaches and recently it is possible to find also comparison articles [47] even if direct performance comparison is difficult due to the many differences between this specific SIB version and the other described implementations, in particular with regard to the subscription interpretation and management.

IV. EXPERIMENTAL EVALUATION

This section presents the evaluation results of the currently accessible SIB implementations: OSGi SIB, RedSIB, PySIB, and CuteSIB. All of the SIBs were executed with volatile storage. The evaluation is performed using the Performance Evaluation Suite [48] developed by the University of Bologna. The setup adopted in the evaluation phase is the following: PES runs on a personal computer (named *Smart-M3-AIO*) provided with 4 Intel(R) Core(TM) i5-4430S CPU @ 2.70GHz and 3,5 GB RAM. Each of the four SIBs mentioned above runs on a dedicated VirtualBox virtual machine (1 CPU, 512 MB RAM) on a server hosted at the ARCES department of the University of Bologna. This server, called *mml*, has 12 processors Intel(R) Xeon(R) CPU E5-2430 v2 @ 2.50GHz. *mml* and *Smart-M3-AIO* are connected through a Gigabit LAN.

A. Update Test

This test (Fig. 8) is used to assess the performance of the insertion mechanism of the SIBs. The update of the knowledge base can be requested using the SPARQL UPDATE language or the RDF-M3 formalism. The following test is related to the insertion of a block of *n* triples (with *n* ranging from 100 to 2000) composed by:

- a subject: http://ns#sub<X> (URI)
- a predicate: http://ns#pred<X> (URI)
- an object: “X” (Literal)

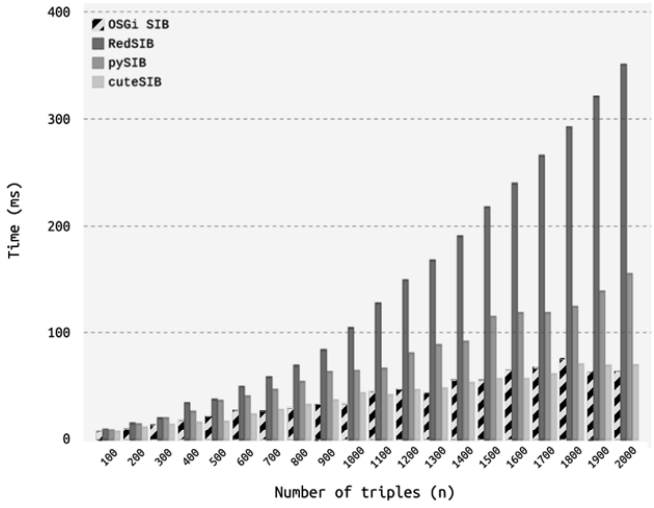

Fig. 8. Time to insert a block of n triple with the RDF-M3 formalism

B. Query Test

The query mechanism of the four SIBs has been tested with both the SPARQL QUERY language and with the triple pattern based formalism. In the first case the whole content of the KB (made by n triples) has been retrieved with the most general query:

```
SELECT ?s ?p ?o
WHERE { ?s ?p ?o }
```

while in the second case the same task has been performed issuing the triple pattern composed by three wildcards (for the subject, the predicate and the object).

For every possible size of the knowledge base (identified by *n*) the query has been performed ten times and the mean time has been used to plot the charts. The results of the test with the SPARQL query are reported in Fig. 9. Fig. 10 presents the results of the RDF-M3 queries for different SIBs implementation.

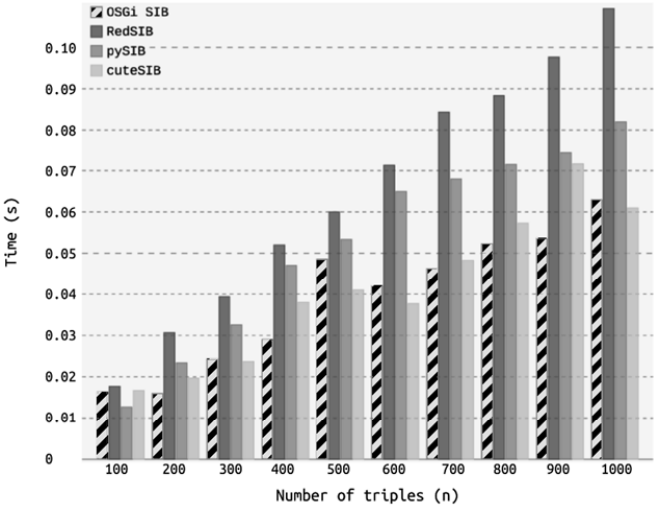

Fig. 9. Time to retrieve the whole KB of n triples with an SPARQL query

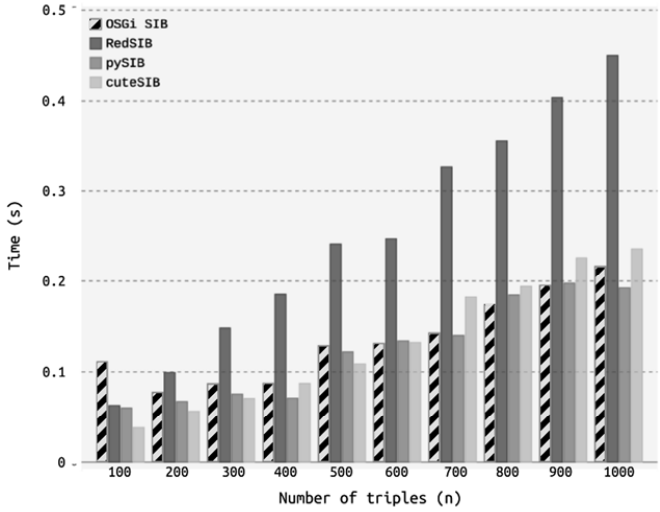

Fig. 10. Time to retrieve the whole KB of n triples with an RDF-M3 query

In both cases it is clear that RedSIB is the slowest broker. CuteSIB and the OSGi SIB show similar performances ad

compete for the role of the fastest SIB. PySIB RDF-M3 engine provides the results of the query in a time sensibly longer than the OSGi SIB and CuteSIB but on the other hand results very fast in replying to the SPARQL query.

The four SIB implementations have also been evaluated against the SPARQL QUERY language by using the SP^{2}B benchmark of the University of Freiburg [49]. This benchmark provides a dataset generator used by authors to generate 10k and 50k triples. All of the seventeen SPARQL queries provided by the benchmark have been used to assess the performance of the SPARQL engines. Table I reports the execution time in seconds for all the queries on the dataset with 10k triples, while Table II contains the results of the same benchmark on the dataset composed by 50k triples.

Results of the SP^{2}B benchmark highlights that RedSIB and CuteSIB, especially with the larger dataset are not able to reply within the allowed time interval (set to 5 minutes). The OSGi SIB managed to complete all of the assigned query but one into the assigned time. pySIB behaves quite well with the smallest dataset, while only half of the queries were successfully completed.

TABLE 1: SP2B BENCHMARK RESULTS ON THE 10K DATASET

Test	Cute SIB	OSGi SIB	PySIB	RedSIB
Q1	0.463	0.008	0.017	0.413
Q2	0.69	0.173	0.301	2.645
Q3a	16.832	0.079	1.349	15.148
Q3b	16.977	0.017	1.324	15.242
Q3c	17.000	0.008	1.306	15.269
Q4	timeout	7.681	timeout	timeout
Q5a	0.209	0.687	1.194	timeout
Q5b	timeout	0.312	31.455	timeout
Q6	0.621	0.779	timeout	timeout
Q7	20.865	0.195	44.840	19.895
Q8	17.634	0.065	0.073	14.836
Q9	34.715	0.037	0.682	29.216
Q10	0.056	0.043	0.064	0.060
Q11	0.333	0.018	0.184	0.026
Q12a	0.004	0.010	0.250	0.016
Q12b	17.484	0.011	0.052	14.830
Q12c	0.005	0.006	0.013	0.005

TABLE 2: SP2B BENCHMARK RESULTS ON THE 50K DATASET

Test	Cute SIB	OSGi SIB	PySIB	RedSIB
Q1	9.216	0.012	0.595	8.947
Q2	9.893	2.698	timeout	timeout
Q3a	timeout	0.298	timeout	timeout
Q3b	timeout	0.018	timeout	timeout
Q3c	timeout	0.012	5.657	timeout
Q4	timeout	timeout	timeout	timeout
Q5a	timeout	26.494	timeout	timeout
Q5b	timeout	12.380	timeout	timeout
Q6	10.241	3.630	timeout	3.687
Q7	timeout	4.952	timeout	timeout
Q8	timeout	0.049	0.627	timeout
Q9	timeout	0.057	3.110	timeout
Q10	0.182	0.171	0.109	0.057
Q11	0.118	0.022	0.973	0.112
Q12a	0.004	0.040	0.872	1.166
Q12b	timeout	0.022	0.054	timeout
Q12c	0.004	0.006	0.018	0.010

C. *Evaluation of the Subscription mechanism*

In publish-subscribe platforms the performance evaluation must take into account the timeliness of each notification and the number of the notifications that get lost.

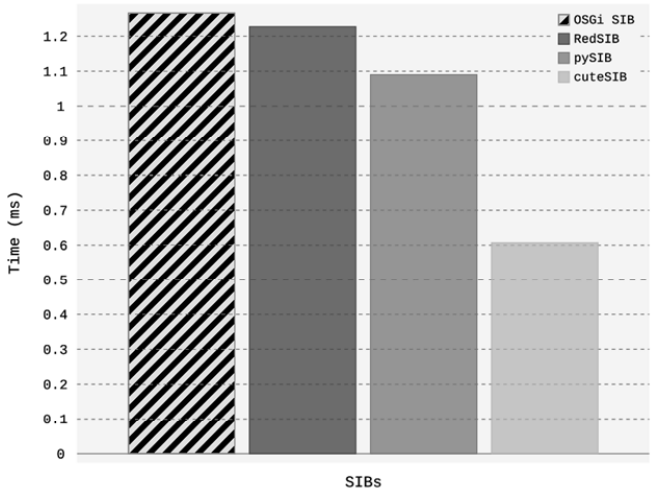

Fig. 11. Time to receive the notification for the update of a triple

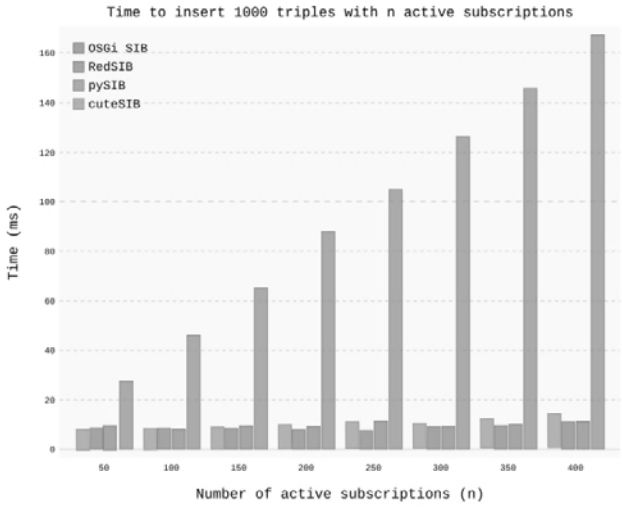

Fig. 12. Time to insert 100 triples with a varying number of subscriptions

In this section an example scenario has been taken into consideration: it is about an instant messaging application which uses the SIB to store information about its users. Users are characterized with five triples stating the class of the user, the user ID and its password, the status and the personal message. In the example 1000 users are registered to the application, resulting in 5000 triples. In the first test the suite subscribed to the status of a specific user. The time interval between the sending of the status update and the receiving of the related notification has been measured. The test was performed ten times then the mean values are plotted in Fig. 11.

For publish-subscribe systems it is also important that active subscriptions do not affect the performance of the platform. Fig. 12 shows the performance evaluation of the four compared SIB implementations in the scenario where an insertion of 100 triples into an empty knowledge base has to

be performed. Each SIB maintains *n* active subscriptions (not triggered by the updates of the KB). It is instantly visible that pySIB fails this test since the performance of update requests are dramatically affected by the number of active subscriptions. This performance degradation is due to the experimental support for subscriptions that yet needs to be optimized in this relatively young SIB implementation. The other SIB implementations show no significant influence of the subscriptions on the speed of the update process.

V. Conclusion

This paper analyzed recent capabilities of the Smart-M3 platform, which provides a promising technology and open source middleware to create smart spaces in accordance with the M3 architecture. The Smart-M3 is considered one of the most suggestive examples of applying SW technologies to the case of emerging IoT environments. We provided a summarized view on the generic M3 SIB architecture; this result has own value for smart spaces middleware development. We reviewed existing SIB designs and available implementations, discussing their purpose, strengths, and weaknesses. The theoretical study is augmented with experimental evaluation and comparison of the five SIB implementations: Piglet-based SIB, RedSIB, OSGi SIB, pySIB, and CuteSIB. In particular, OSGi SIB seems solid against even very complex SPARQL tests, which is achieved due to the use of such a very effective RDF backend as Jena. On the other hand, OSGi SIB is not slim enough to run on resource constrained devices. Despite being built with orientation to a wide range of devices the performance of PySIB and CuteSIB is observed reasonable in specific tests. Indeed, a weak point is the SPARQL management, which needs further development.

Acknowledgment

Authors thank to professor Tullio Salmon Cinotti for giving us the chance to experiment with smart environments in its laboratory. The presented results are part of the research carried out within the project funded by grant #16-07-00462, #16-29-12866, and #14-07-00252 of the Russian Foundation for Basic Research. The work has been partially financially supported by Government of Russian Federation, Grant 074-U01.

References

[1] S. Balandin and H. Waris "Key Properties in the Development of Smart Spaces" *Human Computer Interaction International conference (HCII 2009)*, LNCS 5615, San Diego, USA, 2009, pp. 3-12.

[2] I. Oliver and S. Boldyrev, "Operations on spaces of information," *Proc. IEEE Int'l Conf. Semantic Computing (ICSC '09)*, IEEE Computer Society, 2009, pp. 267–274.

[3] E. Ovaska, T. Cinotti, A. Toninelli, "The design principles and practices of interoperable smart spaces", *Advanced Design Approaches to Emerging Software Systems*, 2012, 30 p.

[4] E. Gilman, O. Davidyuk, X. Su, J. Riekki, "Towards interactive smart spaces", *Journal of Ambient Intelligence and Smart Environments*, 2013, vol.5, no 1, pp. 5–22.

[5] D. Korzun, A. Kashevnik, S. Balandin, and A. Smirnov, "The Smart-M3 platform: Experience of smart space application development for Internet of Things", *Internet of Things, Smart Spaces, and Next Generation Networks and Systems*, LNCS 9247, Springer, 2015, pp. 56–67.

[6] D. Korzun, "On the smart spaces approach to semantic-driven design of service-oriented information systems," *in Proc. 12th Int'l Baltic Conf. on Databases and Information Systems (DB&IS 2016)*, Springer, 2016, pp. 181–195.

[7] J. Augusto, V. Callaghan, D. Cook, A. Kameas, I. Satoh, "Intelligent environments: a manifesto", *Human-centric Computing and Information Sciences*, vol. 3, no. 1, 2013, 3:12.

[8] A. Smirnov, T. Levashova, N. Shilov, "Online Communities for Agent Collaboration in Cyber-Physical-Social Systems", *Joint Proceedings of the BIR 2015 Workshops and Doctoral Consortium co-located with 14th International Conference on Perspectives in Business Informatics Research (BIR 2015)*, Tartu, Estonia, 2015., vol. 1420, pp. 124–135.

[9] D. Korzun, I. Nikolaevskiy, A. Gurtov, "Service Intelligence and Communication Security for Ambient Assisted Living", *International Journal of Embedded and Real-Time Communication Systems (IJERTCS)*, 2015, vol. 6, issue 1, pp. 76–100.

[10] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web. Scientific american, vol. 284(5), 2001, pp. 28–37.

[11] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions" *Future Generation Computer Systems*, vol, 29(7), 2013, pp.1645–1660.

[12] M. Weiser, "The computer for the 21st century", *Scientific american*, vol. 265(3), 1991, pp. 94–104.

[13] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton, "Smart objects as building blocks for the Internet of Things," *IEEE Internet Computing*, vol. 14, no. 1, 2010, pp. 44–51.

[14] D. Korzun, S. Balandin, and A. Gurtov, "Deployment of Smart Spaces in Internet of Things: Overview of the design challenges," *in Proc. 13th Int'l Conf. Next Generation Wired/Wireless Networking and 6th Conf. on Internet of Things and Smart Spaces (NEW2AN/ruSMART 2013)*, LNCS 8121, Springer, 2013, pp. 48–59.

[15] J. Honkola, H. Laine, R. Brown, and O. Tyrkkö, "Smart-M3 information sharing platform", *in Proc. IEEE Symp. Computers and Communications (ISCC'10)*, IEEE Computer Society, 2010, pp. 1041–1046.

[16] A. Seaborne, G. Manjunath, C. Bizer, J. Breslin, S. Das, I. Davis, B. Nowack, "SPARQL/Update: A language for updating RDF graphs", *W3c member submission*, vol. 15., 2008.

[17] E. Prud'Hommeaux, A. Seaborne, "SPARQL query language for RDF", *W3C recommendation*, vol. 15, 2008.

[18] J. Honkola, H. Laine, R. Brown, and I. Oliver, "Cross-domain interoperability: A case study," *in Proc. 9th Int'l Conf. Next Generation Wired/Wireless Networking (NEW2AN'09) and 2nd Conf. Smart Spaces (ruSMART'09)*, LNCS 5764, Springer-Verlag, 2009, pp. 22–31.

[19] A. Smirnov, A. Kashevnik, N. Shilov, I. Oliver, S. Balandin and S. Boldyrev "Anonymous Agent Coordination in Smart Spaces: State-of-the-Art", *2nd Russian Conference on Smart Spaces – ruSMART 2009*, LNCS 5764, St-Petersburg, Russia, 2009, pp. 42-51.

[20] D. Korzun, A. Lomov, P. Vanag, J. Honkola, and S. Balandin, "Generating modest high-level ontology libraries for Smart-M3", *Proc. 4th Int'l Conf. Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010)*, 2010, pp. 103–109.

[21] A. Smirnov, A. Kashevnik, N. Shilov, S. Balandin, I. Oliver, S. Boldyrev "On-the-Fly Ontology Matching in Smart Spaces: A Multi-Model Approach", *3rd Russian Conference on Smart Spaces – ruSMART 2010*, LNCS 6294, Russia, 2010. pp. 72-83.

[22] V. Luukkala and J. Honkola, "Integration of an answer set engine to smart-m3", *Proc. 3rd Conf. Smart Spaces (ruSMART'10) and 10th Int'l Conf. Next Generation Wired/Wireless Networking (NEW2AN'10)*, Springer-Verlag, 2010, pp. 92–101.

[23] G. Abowd, A. Dey, P. Brown, N. Davies, M. Smith, P. Steggles, P. Session, "Towards a better understanding of context and context-awareness", *CHI 2000 Workshop on the What Who Where When and How of Context awareness*, 2000, pp. 1–6.

[24] D. Korzun, I. Galov, A. Kashevnik, N. Shilov, K. Krinkin, Y. Korolev, "Integration of Smart-M3 applications: Blogging in smart conference", *In Smart Spaces and Next Generation Wired/Wireless Networking*, Springer, 2011, pp. 51–62.

[25] A. Smirnov, A. Kashevnik, S. Balandin, S. Laizane, "Intelligent mobile tourist guide", *Internet of Things, Smart Spaces, and Next Generation Networking*, 2013, Springer, pp. 94–106.

[26] D. Korzun, I. Galov, A. Kashevnik, S. Balandin, Virtual shared workspace for smart spaces and M3-based case study, *Proc. 15th Conf. of Open Innovations Association FRUCT*, 2014, pp. 60–68.

[27] A. Smirnov, N. Shilov, A. Kashevnik, N. Teslya, S. Laizane, "Smart Space-based Ridesharing Service in e-Tourism Application for Karelia Region Accessibility: Ontology-based Approach and Implementation", *8th International Joint Conference on Software Technologies*, 2013, Reykjavik, Iceland, pp. 591–598.

[28] L. Bedogni, L. Bononi, M. Di Felice, A. D'Elia, T. Cinotti, "A Route Planner Service with Recharging Reservation: Electric Itinerary with a Click", *IEEE Intelligent Transportation Systems Magazine*, vol. 8(3), 2016, pp. 75–84.

[29] L. Bedogni, L. Bononi, M. Di Felice, A. D'Elia, R. Mock, "An integrated simulation framework to model electric vehicles operations and services", *IEEE Trans. Veh. Technol*, vol. 65, 2016, pp. 5900–5917.

[30] Smirnov A., Kashevnik A., Mikhailov S., Mironov M.. Smart M3-Based Robot Interaction Scenario for Coalition Work, *International Conference on Interactive Collaborative Robotics (ICR 2016)*, Budapest, Hungary, LNAI 9812, Springer, 2016, pp. 199–207.

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

[32] D. Manzaroli, L. Roffia, T. Cinotti, E. Ovaska, P. Azzoni, V. Nannini, S. Mattarozzi, "Smart-M3 and OSGi: The interoperability platform", *in Proc. IEEE Symp. Computers and Communications (ISCC'10)*, IEEE Computer Society, 2010, pp. 1053–1058.

[33] F. Viola, A. D'Elia, L. Roffia, and T. S. Cinotti, "A modular lightweight implementation of the Smart-M3 semantic information broker", *in Proc. 18th Conf. of Open Innovations Association FRUCT and ISPIT Conference*, IEEE, 2016, pp. 370–377.

[34] I. Galov, A. Lomov, D. Korzun, "Design of semantic information broker for localized computing environments in the Internet of Things," *in Proc. 17th Conf. of Open Innovations Association FRUCT*, ITMO University, IEEE, 2015, pp. 36–43.

[35] J. Suomalainen, P. Hyttinen, and P. Tarvainen, "Secure information sharing between heterogeneous embedded devices", *in Proc. 4th European Conf. Software Architecture (ECSA '10): Companion Volume*, ACM, 2010, pp. 205–212.

[36] J. Gomez-Pimpollo, R. Otaolea, "Smart Objects for Intelligent Applications-ADK", *In 2010 IEEE Symposium on Visual Languages and Human-Centric Computing*, 2010, pp. 267–268.

[37] J. Kiljander, A. Ylisaukko-oja, J. Takalo-Mattila, M. Eteläperä, J.-P. Soininen, "Enabling semantic technology empowered smart spaces", *Journal of Computer Networks and Communications*, 2012, Vol. 2012, 14 p.

[38] D. Korzun, A. Lomov, P. Vanag, S. Balandin, J. Honkola, "Multilingual Ontology Library Generator for Smart-M3 Information Sharing Platform", *International Journal on Advances in Intelligent Systems*, vol. 4, no 3 & 4, 2011, pp. 68–81.

[39] M. Ball, V. Callaghan, "Managing Control, Convenience and Autonomy: A Study of Agent Autonomy in Intelligent Environments", *Journal of Ambient Intelligence and Smart Environments*, 2012, Vol. 12, pp. 1–38.

[40] D'Elia, A., Honkola, J., Manzaroli, D., & Cinotti, T. S., "Access control at triple level: Specification and enforcement of a simple RDF model to support concurrent applications in smart environments". *In Smart Spaces and Next Generation Wired/Wireless Networking*, 2011, Springer, pp. 63-74.

[41] L. Roffia, F. Morandi, J. Kiljander, A. Elia, F. Vergari, F. Viola, T. Cinotti, "A Semantic Publish-Subscribe Architecture for the Internet of Things", *IEEE Internet of Things Journal*, vol. PP Iss. 99, 2016.

[42] A. D'Elia, L., Roffia, G., Zamagni, F. Vergari, A. Toninelli, P. Bellavista, "Smart applications for the maintenance of large buildings: How to achieve ontology-based interoperability at the information level", *2010 IEEE Symposium on Computers and Communications (ISCC)*, 2010, pp. 1–6.

[43] F. Vergari, T. Cinotti, A. D'Elia, L. Roffia, G. Zamagni, C. Lamberti, "An integrated framework to achieve interoperability in person-centric health management", *International journal of telemedicine and applications*, 2011, vol. 5.

[44] F. Vergari, S. Bartolini, F. Spadini, A. D'Elia, G. Zamagni, L. Roffia, T. Cinotti, "A smart space application to dynamically relate medical and environmental information", *In Proceedings of the Conference on Design, Automation and Test in Europe*, European Design and Automation Association, 2010, pp. 1542–1547.

[45] J. Kiljander, A. D'Elia, F. Morandi, P. Hyttinen, J. Takalo-Mattila, A. Ylisaukko-Oja, T. Cinotti, "Semantic interoperability architecture for pervasive computing and Internet of Things", *IEEE access*, vol. 2, 2014, pp. 856–873.

[46] A. D'Elia, F. Viola, L. Roffia, T. Cinotti, "A Multi-Broker Platform for the Internet of Things", *In Conference on Smart Spaces*, Springer, 2015, pp. 34–46.

[47] H. Hamza, E. Ashraf, A. Nabih, M. Abdallah, A., Gamaleldin, A. D'Elia, A. Attallah, "Design and Implementation of an Interoperable and Extendable Smart Home Semantic Architecture using Smart-M3 and SOA", *In The Tenth International Conference on Networking and Services, ICNS*, 2014, pp. 48–53.

[48] F. Viola, A. D'Elia, L. Roffia, T. Cinotti, "Performance Evaluation Suite for Semantic Publish-Subscribe Message-oriented Middlewares", *The Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2016)*, 2016 (submitted for publication).

[49] M. Schmidt, T. Hornung, G. Lausen, C. Pinkel, "SP^ 2Bench: a SPARQL performance benchmark", *In 2009 IEEE 25th International Conference on Data Engineering*, IEEE, 2009, pp. 222–233).