RedSib: a Smart-M3 Semantic Information Broker Implementation

Francesco Morandi, Luca Roffia, Alfredo D'Elia, Fabio Vergari,
T. Salmon Cinotti
Alma Mater Studiorum – Università di Bologna
Bologna, Italy
{fmorandi, lroffia, adelia, fvergari, tsalmon}@arces.unibo.it

Abstract

Smart-M3 is an open source middleware solution originally released by Nokia as a prototype reference infrastructure to support context-aware ontology-driven smart applications. This paper proposes a renewedSmart-M3Semantic Information Broker implementation with increased performance and usability levels. In the proposed solution many features have been added or modified, preserving compatibility with the previous release. The major changes are related to the subscription mechanism, the RDF store and the supported encodings for information representation and query. SPARQL query language replaces Wilbur. This paper enlightens the analysis carried out on the original implementation and discusses the choices made to increase its maturity level. The work done is a step forward towards a stable and efficient open interoperability platform for the emerging market of smart space based services.

Index Terms: Information Interoperability, Smart Space, Triple Store, Smart-M3, RDF, Redland, Semantic Web, SIB, Semantic Information Store.

I. INTRODUCTION

In the smart environments market, the need for an open interoperability platform to support the deployment of innovative services is increasing day by day. In order to fully respond to multi-domain application requirements, this platform should handle heterogeneous devices (i.e. ranging from personal computers, servers and clusters to mobile devices and sensor nodes) and heterogeneous data (i.e. ranging from raw sensor data to user profiles). To have an impact on the market, this platform has also to be scalable, reliable, secure, reactive and easily accessible [1, 2]. The Smart-M3 interoperability platform [3] is a candidate solution to meet these requirements. It is based on the smart space abstraction and it inherits the semantic web data model [4-7] to provide an interoperable machine interpretable representation of data: the analysis of Smart-M3principles and its application design approach are also enlightened in [7]. Smart-M3 was released as open source platform at the NoTA Conference on October 1, 2009 [8] in San Jose, it was adopted by the ARTEMIS project SOFIA as baseline information interoperability component [9] and it has been further developed within several communities including the FRUCT and the SOFIA communities [10, 11], the Open-M3 community [12], and the Smart-M3 Lab community at the University of Bologna [13]. Soon after its first release, the Smart-M3 potential was understood and applied in other European projects, e.g. on eHealth [14] and eMobility[15]. Furthermore EIT ICT Labs, an Innovation Factory for ICT Innovation in Europe, included the smart spaces among its innovation action lines [16] and, under the name of M3, the platform

was adopted by the *Smart Space infrastructure* recently established at the Helsinki Node of the EIT ICT Labs [17].According to the Semantic Web theory [4] and in order to deal with heterogeneous data, the Smart-M3 platform was designed to store and retrieve information represented as a set of RDF triples [18]. Smart-M3 client agents (i.e. *Knowledge Processors* or KPs) communicate with the server part(i.e. the *Semantic Information Broker* also known as SIB) using an XML based protocol called SSAP (*Smart Space Access Protocol*) [19]. The Smart-M3 platform offers a subscribe-notify mechanism for context reactivity and several "Knowledge Processor Interface" (KPI) (in popular programming languages including C, C#, Java, PHP, JavaScript and Python) to simplify the communication with the SIB by abstracting from the low-level protocol details. The authors testedSmart-M3 in several projects [20-25]in order to better understand its strengths, weaknesses, opportunities and threats. As a result it was decided to replace the core part of the SIB (i.e. the no more supported Piglet RDF store [3]) with the currently supported Redland RDF store [26]. Additional contributions addressed the RDF store operating mode, the subscription handling mechanism and the SSAP messages parsing.

Like the Piglet SIB, also the proposed implementation, named *RedSib* in this paper, is available from SourceForge [27] under the name of Smart-M3-B.

This paper is organized in three main sections: in section II the motivations for a new SIB implementation are presented. In section III we describe the details of this new implementation. In section IV experimental results are reported. In section V conclusions are drawn and future work is envisioned.

II. Analysis and Motivations

In this section we analyze the architecture of the original SIB (released by Nokia and named *Piglet SIB* in this paper as Piglet is the name of its RDF store) [5, 3] to underline some critical points and to motivate the implemented changes. We identified a set of features to be preserved and a set of issues to be solved (see section II.C).

A. Architecture overview

Fig. 1 presents a schematic overview of the Piglet SIB architecture.

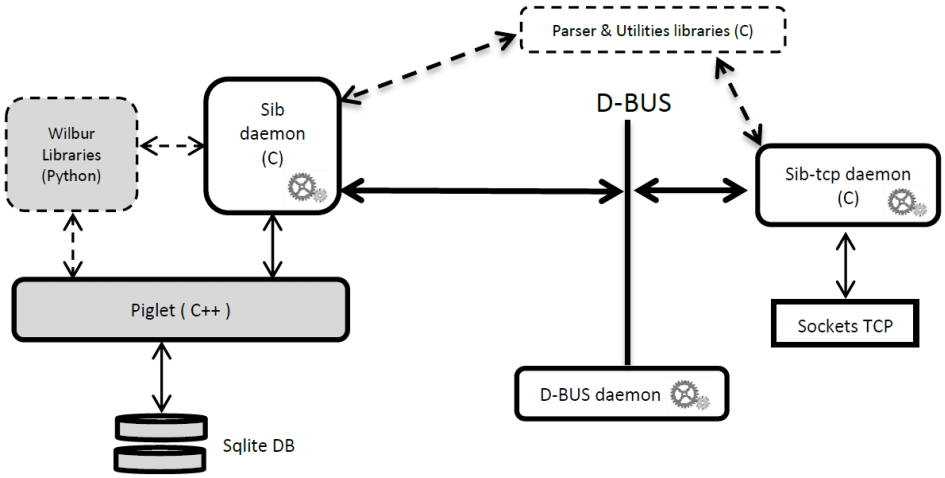

Fig. 1. Piglet SIB architecture overview

The *Piglet SIB* is composed of two daemons(i.e. *sib-daemon* and *sib-tcp*, *sib-nota* is not considered due to its still prototyped implementation) and a set of libraries (e.g. Piglet RDF store, Wilbur query engine [5, 28]).The daemons are seen from the operating system (i.e. a Linux based system) as independent processes that exchange content through the D-Bus [29].

The *sib-daemon* handles the messages coming from the D-Bus and it serializes them into a sequential scheduler; the overall synchronization is ensured by a mutex system.

The *sib-tcp* is responsible for managing the connections with the clients (i.e. the KPs) through TCP sockets. It translates SSAP requests into D-Bus messages, and it implements a bidirectional bridge between TCP sockets and the *sib-daemon*. All the D-Bus and parser support libraries are written in C.

The Piglet RDF store is a C++ library instantiated by the *sib-daemon*. Piglet operates on a SQL Lite persistent database and it natively supports forward chaining reasoning based on RDF++ semantics [28].

Queries on the Piglet RDF store can benefit from the Wilbur query engine (i.e. a Python library used by the *sib-daemon*). The Wilbur query engine can be used both for instant queries and for persistent queries (i.e. subscriptions).

B. The sib-daemon internal structure

Fig. 2 shows the internal architecture of the *sib-daemon*.

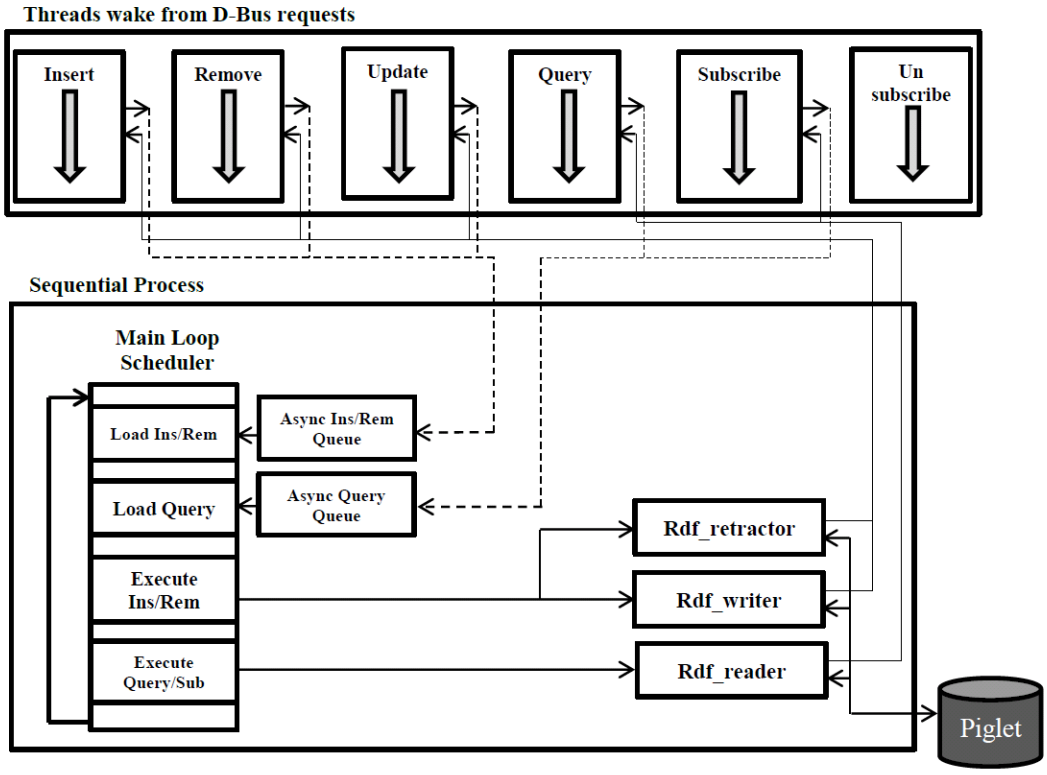

Fig. 2. Sib-daemon architecture

The *sib-daemon* is implemented as a multithreading application where the scheduler is the main loop thread. Every time a request operation (i.e. remove, insert, update, query, subscribe, unsubscribe) comes from the D-Bus, a new thread is allocated. Every thread puts its own requests in an asynchronous queue (see *Async Ins/Rem Queue*, *Async Query*

Queue in Fig. 2) and it waits for a wake-up signal. The scheduler executes the requests in the following order: remove (see *rdf_retractor* in Fig. 2), insert (see *rdf_writer* in Fig. 2) and query operations (see *rdf_reader* in Fig. 2).In this architecture, subscriptions are implemented as queries that constantly reload themselves into the query queue until the corresponding unsubscription occurs. When a waiting thread receives a wake up signal, it checks if its operations have been completed. If yes, the result is sent to the D-Bus and the thread is terminated, otherwise the thread waits for the next wake-up signal.

C. Analysis results

As result of our analysis the following aspects of *Piglet SIB* have been identified as the features to be preserved:

- **V1: C core implementation**: native C implementation of daemons and support libraries can be considered as one of best solutions to optimize performances and memory usage, especially if compared with more computationally expensive high level languages (e.g. Java or Python). C binaries are low weight and do not need a virtual machine or an interpreter (i.e. the SIB could be run also on embedded devices).
- **V2: Separate processes as coordinated modules**: the D-Busacts as an inter-process communication (IPC) mechanism designed for the X Window System for exchanging messages between applications. Even if it has been designed originally for a different purpose, it is anyway an efficient way to exchange information between independent processes. Moreover, it can run easily on every Linux system even without the graphical environment (i.e. just in a shell). The D-Bus mechanism allows a *sib-daemon* to be connected to an arbitrary number of *sib-tcp* processes.
- **V3: RDF++ materialization:** starting from the asserted triples and applying the rules defining the RDF++ semantics [4] it is possible to infer new information. The inferred triples are automatically inserted into the store, augmenting the knowledge available for clients. Our aim has been to maintain this reasoning feature but making it optional (i.e. it can be enabled/disabled).
- **V4: Synchronization at triple level:** the need to control concurrent access on shared RDF sub-graphs is an important feature in a multi agent scenario. In the Piglet SIB implementation a prototype of access control was implemented [30] and so we decided to maintain and improve it.

  The main issues to face can be considered:
- **I1: Diverging DB size and performance**: Piglet RDF store was an experimental software still considered in beta version when its development ended. For this reason some bugs or not optimized code were still present, like a continuous growing of the DB size. If a KP randomly inserts and removes triples from the Piglet RDF store, the dimension of the DB diverges. Furthermore, the Piglet RDF store only offers a persistent SQL Lite implementation and this can represent a limit when high performances are needed (i.e. using volatile memory instead of a persistent DB). In modern triple stores, like Redland [26] or Jena [31], modifications to the store can be done in RAM and then are committed to disk improving performances.
- **I2: Absence of SPARQL support:** Wilbur queries are not yet maintained and the only implementation is a Python based library that works strictly connected with

the Piglet RDF store. Furthermore, the W3C recommended query method for RDF since 2008 is SPARQL [32] and implementing SPARQL in Piglet would require a huge programming effort because the library has not been originally built for it.

- **I3: Lack of support for RDF/XML triple encoding:** RDF/XML is one of the most common syntaxes for serializing RDF knowledge and OWL ontologies. According to the Semantic Web stack, the ontology layer is based on RDF and on XML (i.e. each ontology is a valid RDF document that can be serialized in XML). Supporting RDF/XML encoding of triples represents a great improvement and it is functional for inserting whole ontologies in the knowledge base, starting from the output files of well-known editors like Protégé[33].
- **I4: Subscription performance impact and stability:** subscriptions are a powerful mechanism offered by Smart-M3 to provide software agent reactivity in context aware applications. Subscriptions are also widely used as an efficient synchronization method between multiple cooperating KPs and they reduce the overall SSAP traffic by providing a way to inform KPs of relevant contextual changes, avoiding the continuous polling of the SIB. Subscription handling on the client side is deeply analyzed in [34]. In this paper we focus on the server side. Subscriptions are managed by the SIB as recursive queries on the triple store (see Section II.B); when the triple store content changes (i.e. when an insert, remove or update occurs) all the queries related to active subscriptions are executed. The query results are compared with previously cached ones, and, if differences exist, notifications are sent to the subscribed KPs. This mechanism strongly affects performances because, for every insert/remove/update operation, multiple queries (i.e. multiple accesses to disk) are performed. Moreover, in order to be notified, every subscribed KP maintains a socket opened for each subscription. The SIB does not detect lost connections (e.g. due to network fluctuations or other reasons) and it continues to perform useless queries, thus dramatically reducing the overall performance.

III. THE NEW SIB IMPLEMENTATION

In this section we describe the new SIB implementation. To overcome the issues *I1*, *I2* and *I3*,we decided to replace the Piglet RDF store with Redland [35] (see Section III.B). Concerning the issue *I4*, we implemented a new subscribe/notify mechanism as described in Section III.C.

A. Redland RDF store overview

Redland is a set of free C libraries for the RDF. Redland natively supports SPARQL (1.0 fully and 1.1 partially) and it offers two storage options: persistent and volatile memory. Redland provides its own APIs for C (i.e. these have been used instead of the Piglet API to do every basic operation on the triple store). Redland works with two support libraries:

- Raptor is a free software / Open Source C library providing parsers and serializers for RDF triples. Raptor was designed to work closely with the Redland RDF library but it is an entirely separate module [36] (e.g. it has also been used in the Piglet implementation).

– Rasqal [37] is a free software / Open Source C library handling RDF query and responses. SPARQL Query 1.0, many features of SPARQL Query 1.1, and the Experimental SPARQL extensions (LAQRS) are supported. Rasqal was designed to work closely with the Redland RDF library and the Raptor RDF Syntax Library but it is entirely separate from both of them.

B. Redland integration into the SIB

To implement a Redland based SIB, part of the *sib-daemon* code was rewritten in order to replace the calls to the Piglet library with calls to the Redland C API. This was a simple process thanks to the SIB modular architecture. The parsing and the composition of the internal data structure for the SSAP primitives was not affected by the introduction of the Redland RDF store, therefore the *sib-tcp* process was not involved in this phase of the integration process. The *sib-daemon*, instead, was quite strictly coupled with the Piglet module, therefore all basic sequential methods (i.e. insert (*rdf_writer*), remove (*rdf_retractor*) and query (*rdf_reader*)) have been re-implemented using the Redland API methods. This integration solved the issue *I1*. Several benchmarks verified this assertion. The memory space usage during these tests, analysed with Valgrind [38], turned out to be stable.

Fig. 3 gives an overview of the new implementation, and it shows that the most affected part is the *sib-daemon* (renamed as *RedSib-daemon* in the figure).

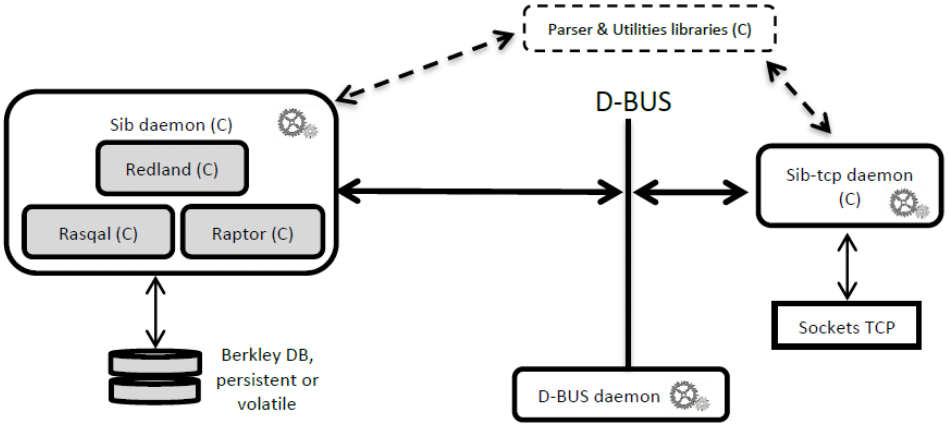

Fig. 3. RedSib architecture overview

As Redland natively supports SPARQL queries and RDF/XML insert operations, to solve issues *I2* and *I3*, it was necessary to extend or re-implement other parts of the SIB (i.e. the *sib-tcp*, the parser library and the utilities libraries) to proper manage the corresponding SSAP messages.

C. The new subscription mechanism implementation

To solve the issue *I4*, a different, more programmatic approach was required.

To improve the efficiency of the subscription mechanism it has been necessary to detect the weaknesses of the previous algorithm. It can be observed that, in most cases, subscriptions are not fired by KP operations, and, if yes, only some of them correspond to effective notifications to be sent. The need of doing many queries to the disk every time an insert or remove primitive is performed was not acceptable. Our implementation overcomes this problem by caching in RAM the triple patterns related to each

subscription. For each subscription a string based matching is done only on added or removed triples instead of doing continuously the same query on the whole triple store. Any triple inserted that matches with at least one of the subscription patterns should be notified as a *new_triple*, and any triple removed that matches with at least one of the patterns should be notified as an *obsolete_triple*.

The *RedSib-daemon* architecture is reported in Fig. 4. The new subscription mechanism has been implemented adding a new method (*Persistent query emulation*) and it is based on the concept of state buffer. We define a *state buffer* for a single subscription as the set of triples matching with the triple patterns of that subscription. We also introduced three volatile triple stores:

- The first one stores all the state buffers of the active subscriptions (*Context Status TS*)
- The second one stores all the inserted triples (*Temp Ins TS)*in a single scheduler cycle.
- The third one stores all the removed triples(*Temp Rm TS*) in a single scheduler cycle.

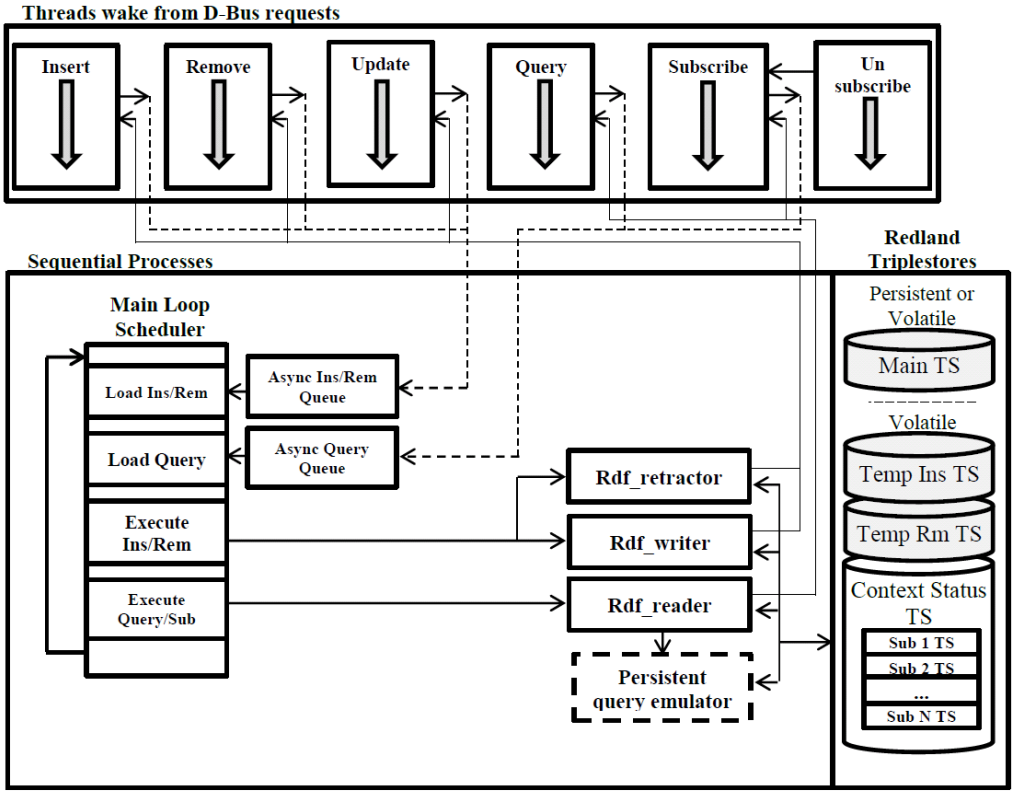

Fig. 4. RedSib-daemon architecture (new blocks are enclosed in dashed boundaries)

The new algorithm acts as follows:

- When a subscription request is received (i.e. it is scheduled in the sequential process as a common query in the query queue), a new thread is created. Each thread maintains a local copy of its corresponding RDF query patterns, along with a copy of the current state buffer (i.e. it is obtained querying the triple store). The

state buffer is notified as a query result and stored into the *Context Status* triple store.

- When an insert, remove or update operation is received, the added and removed triples are inserted in the correspondent temporary stores (respectively *Temp Ins TS* and *Temp Rm TS*) and the Context Status triple store is consequently updated. Every subscription thread, if involved, compares its local copy of the state buffer with that contained in the Context Status. If the values are different it is possible to infer which triples have to be notified. At the end, the subscription thread updates its local state buffer.
- When an unsubscription request is received the state buffer is cleaned and the subscription thread is terminated.

Considering issue *I4*, the overall SIB stability requires a garbage collector mechanism for cleaning "broken" subscriptions. A conjunct mechanism between *sib-daemon* and *sib-tcp* is required to properly remove unused subscriptions. A single subscription, in fact, requires two running threads, one in *sib-tcp*, to manage sockets, and one in *sib-daemon* to interact with the store.

In the original SIB version, an uncaught socket exception causes the respective subscription thread in the *sib-daemon* to continue running useless. When the number of running threads reach the maximum allowed by the operative system, the *sib-daemon* crashes obliging to restart.

To solve this problem we decided to apply the natural way of closing subscriptions in the SIB, i.e. unsubscriptions. When a notification is necessary, the *sib-tcp* tries to send it checking for any exception in the socket status. If the socket exception occurs, the *sib-tcp* catches it and sends back to the D-Bus a nun subscription message containing the same ID of the disconnected KP. In this way, the *sib-daemon* receives an unsubscription request exactly as it was received from the corresponding disconnected KP. This mechanism guarantees to terminate the thread inside the *sib-daemon* whenever a subscription indication cannot be delivered.

The described procedure revealed itself vulnerable to a common phenomenon: broken subscriptions which are not fired by any notification are not cleaned. To solve this additional issue, the implemented solution periodically tests socket status transmitting a "space" character (US-ASCII decimal 32) on all subscription sockets. The default interval value is 10 seconds, but it is configurable as parameter of the *sib-tcp*. This feature allows the *sib-tcp* to understand which sockets are really "broken", even in case there are no subscription indications to deliver. If exceptions are catched an unsubscription message is sent to the D-Bus exactly as previously described. With these extensions on *sib-tcp*, which have been tested on all client libraries, the massive use of subscriptions required in smart environment scenarios becomes more reliable.

The new *sib-tcp* module is absolutely compatible with *Piglet SIB*-daemon and libraries; it can be therefore considered a potential new release.

D. SSAP extensions

Issues *I2 and I3* required to extend the SSAP protocol with new messages (see Section II.B). The corresponding SSAP primitives have been implemented in the parser utilities to enable the following functionalities:

- SPARQL query request and response.
- RDF/XML insert, remove and update.

E. RDF++ support

RDF++ reasoning [6] has been implemented as a software module that can be loaded optionally. This extension works applying the set of rules defined in RDF++, on incoming inserted triples. Below are reported the rules implemented in the RDF++ module. If a rule is satisfied, the corresponding inferred triples are added to the triple store, increasing the semantic knowledge.

(1) $holds(p,s,o) \Rightarrow property(p)$

(2) $holds(rdfs:subClassOf, x, y) \Leftrightarrow class(x) \wedge class(y) \wedge \left[\forall z : type(z,x) \Rightarrow type(z,y)\right]$

(3) $holds(rdfs:subPropertyOf, x, y) \Leftrightarrow property(x) \wedge property(y) \wedge [\forall o, v : holds(x,o,z) \Rightarrow holds(y,o,v)]$

(4) $holds(rdfs:domain, p, c) \Rightarrow property(p) \wedge class(c) \wedge [\forall x, y : holds(p,x,y) \Rightarrow type(x,c)]$

(5) $holds(rdfs:range, p, c) \Rightarrow property(p) \wedge class(c) \wedge \left[\forall x, y : holds(p,x,y) \Rightarrow type(y,c)\right]$

The following rules have not been included in the RDF++ reasoner because, in our opinion, they would introduce to much overhead in the reasoning process, without adding a significant value:

(6) $holds(owl:sameAs, x, y) \Leftrightarrow x = y$

(7) $holds(p,x,z) \wedge holds(p,y,z) \wedge type(p, owl:InverseFunctionalProperty) \Rightarrow holds(owl:sameAs, x, y)$

F. Synchronization at triple level

Data access control at triple level is a feature implemented in the *Piglet SIB*[30] and all the functionalities have been maintained on the *RedSib*. Furthermore, the *RedSib* also support this mechanism with the new RDF/XML triples encoding.

IV. EVALUATION

To validate the proposed subscription algorithm some performance comparison between the *RedSib* and the *Piglet SIB* were carried out. Both SIBs were hosted by a Fit-PC2, i.e. an embedded PC based on a1600 Mhz, dual core Intel® Atom™Processor with 1 GB of Central Memory [39]. The operating system was a 32 bit Linux Mint distribution [40].Results of several *Piglet SIB* performance tests are available in the literature, most of them carried out in persistent execution, i.e. with the RDF store located in persistent memory [41, 42]. However, as the *RedSib* RDF store is partially allocated in RAM (cached) when the amount of triples is limited, we decided to compare these two implementations with their RDF store entirely located in non-persistent memory (RAM).This choice is consistent with all applications currently of interest for our research. Furthermore this choice enlightens the "*architectural implementation latency*", and masks out the database specific access latency. In order to locate the Piglet store in volatile memory the *sib-daemon* was launched in a specific folder mounted with *tmpfs* file system. The test - performed by a KP working with the Python KPI libraries [27] - has the following cyclic behavior: at any iteration (until a maximum of 1000 iterations)a subscription to a single random triple is added, and several (in our case 50)*inserts* and *removes* of single random triples which do not trigger any notificationare repeated. Every time a new triple is inserted or removed both SIBs check, each one using its own implementation, if some notification messages have to be delivered. In this way the

average insert (and remove) time as a function of pending subscriptions may be estimated. This test was repeated twice for each SIB, once with an initially empty store and once with nearly 15000 triples sitting in the RDF store. The results are shown in Fig. 5.

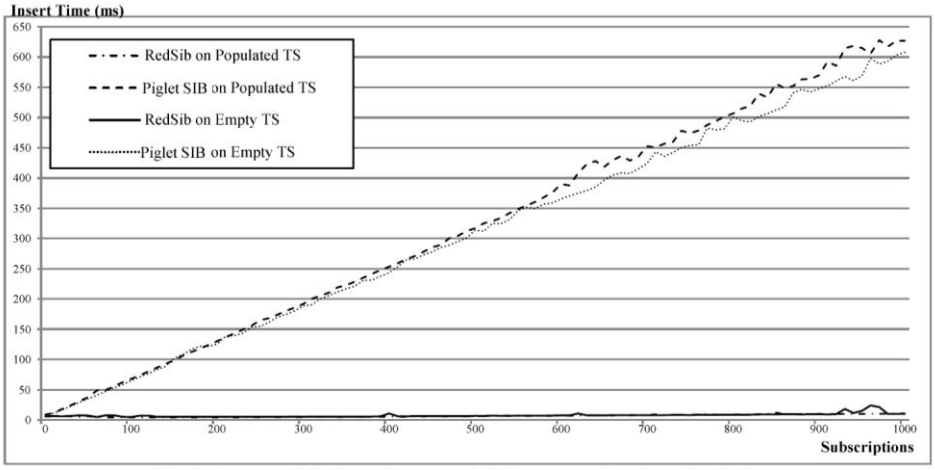

Fig. 5. Average triple insert time goes with the amountof active subscriptions:
this time increases by less than a factor of 2whenclose to 1000 pending subscriptions in the RedSib

Fig. 5 shows that, with no active subscriptions, both SIBs have quite similar performances, as expected, as each of them has its store on fast and non-persistent main memory (5.8 ms for the *RedSib*, 8.9 ms for the *Piglet SIB*). As the number of subscriptions increases, a linear growth can be observed, slightly noised by some unpredictable OS overhead. In both implementations the variability of the database access latency in volatile memory is negligible as the insert operation latencies are very similar in both cases of empty and populated stores. On the other side, the significant insert latency difference between the *RedSib* and the *Piglet SIB* may be motivated by the different implementations of the persistent query emulation algorithm. When a triple is inserted or removed from the *RedSib* the persistent query emulation algorithm "detects" on a fast access table if one or more subscriptions are involved: in our test no subscription has to be notified, therefore all the threads hold their sleep state. Even if not clearly visible in Fig. 5, the insert operation increases by less than a factor of 2 its latency value from 5.8 to 10.3msaswe go up to 1000 active subscriptions. On the other side, in the *Piglet SIB* the trend sare much higher because a query on the entire database is performed for each active subscription whenever an insert operation occurs and all the threads are woke up to check if their own notifications duty has to be carried out. Similar trends can be observed with remove and update operations.

We can conclude that with the proposed subscribe/notify implementation the impact of the amount of active subscriptions on operation latency is kept marginal, at least up to an order of magnitude of 10^3 active subscriptions. This result extends the range of applications that may benefit from the Smart-M3 architecture, and, particularly, it opens up to applications with many mobile clients subscribed to the SIB.

V. CONCLUSION

Smart-M3 is a middleware component to enable the exchange of data and events among heterogeneous devices not originally designed to co-operate. As such SMART-

M3may become a key component of a wide range of environment related co-operative ecosystems. Its mission critical role, though, requires many non-functional qualities to be in place, most of them related to its maturity level. The proposed Smart-M3 revised implementation is intended to be a step forward in this direction. Some preliminary results show a significant performance increase, both in terms of response time and number of active subscriptions. The subscription handling mechanism has now an improved performance profile and it is more stable in presence of unexpected situations, like abrupt client disconnection. Features addressing platform usability were also added, such as, for example the ability to upload ontologies serialized in RDF/XML.

On the other side there are still open issues to be considered. Security, privacy and data integrity solutions were proposed by several authors, particularly within the SOFIA project but they are not yet applied to the proposed implementation, so that they are still left to the application level [43].

Scalability needs to be further investigated, particularly with big data sets, keeping in mind the Semantic Web scenario. SPARQL support, along with the current development of web based APIs (i.e. PHP and JavaScript) are steps in this direction. The SSAP protocol performance level is currently critical and it should be enhanced [44]. Solutions like compressed binary format and connectionless sockets are currently under investigation.

Last but not least, we have to keep in mind that the SIB is only one of the components of the proposed solution. To have a real benefit from the platform, new tools should be implemented to manage and supervise the smart space.

With this paper the authors encourage all Smart-M3 communities worldwide to join their effort and increase the Smart-M3 maturity to the level required by the emerging industry of environment related, co-operative smart space based services.

ACKNOWLEDGMENT

The work described in this paper was carried out within the framework of three projects of the European Joint Undertaking on Embedded Systems ARTEMIS: SOFIA (2009-11), coordinated by NOKIA, CHIRON (2010-13) coordinated by FIMI and Internet of Energy for Electric Mobility (2011-14) coordinated by SINTEF. All of these projects are co-funded by the EU and by National Authorities including MIUR, the Italian Central Authority for Education and Research. Within each project features were added to the Open Source Interoperability Platform and its maturity level was increased. The authors are grateful to ARTEMIS JU, MIUR and all partners for their support to the progress of the addressed open source interoperability platforms. The authors are also grateful to the Helsinki node of EIT ICT Labs, to Juha-Pekka Soininen and his research team on smart spaces for supporting Research Visits to VTT within the framework of EIT ICT LABs Researcher Exchange Program and for hosting the comparison tests presented in section IV. Last but not least, the authors wish to thank Jukka Honkola for his support with fruitful discussions and suggestions.

REFERENCES

[1] S. Balandin and H. Waris, “Key properties in the development of smart spaces,” in Proc. 5th Int'l Conf. *Universal Access in Human-Computer Interaction*. *Part II: Intelligent and Ubiquitous Interaction Environments*(UAHCI ’09). Springer-Verlag, pp. 3–12, 2009.

[2] D. G. Korzun, S. I. Balandin, V. Luukkala, P. Liuha, and A. V. Gurtov, “Overview of Smart-M3 principles for application development,” in Proc. *Congress on Information Systems and Technologies* (IS&IT'11), Conf. *Artificial Intelligence and Systems* (AIS'11), vol. 4. Moscow: Physmathlit, Sep. 2011, pp. 64–71.

[3] J. Honkola, L. Hannu, R. Brown, and O. Tyrkko, "Smart-M3 information sharing platform," *IEEE Symposium on Computers and Communications (ISCC)*, pp. 1041-1046, 2010.
[4] T. Berns-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, vol. 284, pp. 34-43, 2001.
[5] O. Lassila, "Taking the RDF Model Theory Out for a Spin", in: *Ian Horrocks & James Hendler* (eds.): "The Semantic Web - ISWC 2002," Lecture Notes in Computer Science 2342, pp.307-317, Springer Verlag, 2002.
[6] O. Lassila, "*Programming Semantic Web Applications: A Synthesis of Knowledge Representation and Semi-Structured Data*," PhD thesis, Helsinki University of Technology, November 2007.
[7] E. Ovaska, T. Salmon Cinotti, A. Toninelli, "The Design Principles and Practices of Interoperable Smart Spaces," In *Advanced Design Approaches to Emerging Software Systems: Principles, Methodology and Tools*. Liu Xiaodong, Li Yang Eds, 2011.
[8] NoTA Conference, www.vtt.fi/nota2009
[9] SOFIA Project: Smart Objects For Intelligent Applications, http://www.sofia-project.eu/
[10] FRUCT: Finnish-Russian University Cooperation in Telecommunications, http://www.fruct.org/
[11] SOFIA community, http://www.sofia-community.org/
[12] Open-M3 community, http://www.open-m3.org/
[13] L. Roffia, A. D'Elia, F. Vergari, D. Manzaroli, S. Bartolini, G. Zamagni, T. S. Cinotti, and J. Honkola, "A Smart-M3 lab course: approach and design style to support student projects," *8th FRUCT Conference of Open Innovations Framework Program FRUCT*, S. Balandin and A. Ovchinnikov, Eds. Lappeenranta, Finland: Saint-Petersburg State University of Aerospace Instrumentation (SUAI), pp. 142 – 153, 2010.
[14] CHIRON project : Cyclic and person-centric Health management, http://www.chiron-project.eu/
[15] IOE project: Internet of Energy, http://www.artemis-ioe.eu/
[16] EIT ICT LABS, http://www.eitictlabs.eu/action-lines/smart-spaces/
[17] EIT-smartspace, http://www.open-m3.org/EIT-smartspace
[18] RDF Vocabulary Description Language 1.0: RDF Schema, http://www.w3.org/TR/rdf-schema/
[19] Deliverable 5.22: Logical Service Architecture. Architecture for Sofia Interoperability Platform, http://www.sofia-project.eu/system/files/SOFIA_D5-22-LogicalServiceArchitecture-v1-2011-01-02_0.pdf
[20] A. D'Elia, L. Roffia, G. Zamagni, A. Tonielli, and P. Bellavista, "Smart Applications for the Maintenance of Large Buildings: How to Achieve Ontology-based Interoperability at the Information Level," *IEEE symposium on Computers and Communications (ISCC 2010), First International Workshop on Semantic Interoperability for Smart Spaces (SISS 2010).* Riccione, Italy, pp. 1072–1077, 2010.
[21] F. Vergari, T. S. Cinotti, A. D'Elia, L. Roffia, G. Zamagni, and C. Lamberti, "An integrated framework to achieve interoperability in person-centric health management," *International Journal of Telemedicine and Applications*, vol. 2011, pp. 10, 2011.
[22] F. Vergari, S.. Bartolini, F. Spadini, A. D'Elia, G. Zamagni, L. Roffia, T. Salmon Cinotti, "A Smart Space Application to Dynamically Relate Medical and Environmental Information," *Proceedings Design, Automation & Test in Europe Dresden*, Germany March 8-12, 2010.
[23] R. Gazzarata, F. Vergari, J-M. Verlinden, F. Morandi, S. Naso, V. Parodi, T. Salmon Cinotti, M. Giacomini, "The Integration of e-health into the Clinical Workflow – Electronic Health Record and Standardization Efforts," *ICOST 2012, LNCS 7251*, M. Donnelly, C. Paggetti, C. Nugent, M. Mokhtari (Eds.), Springer-Verlag Berlin, Heidelberg 2012, pp. 107-115, 2012.
[24] D. Manzaroli, L. Roffia, T. S. Cinotti, E. Ovaska, P. Azzoni, V. Nannini, and S. Mattarozzi, "Smart-M3 and OSGi: The Interoperability Platform," *IEEE symposium on Computers and Communications (ISCC 2010), First International Workshop on Semantic Interoperability for Smart Spaces (SISS 2010).* Riccione, Italy, pp. 1049–1054, 2010.
[25] S. Pantsar-Syväniemi, E. Ovaska, S. Ferrari, T. S. Cinotti, G. Zamagni, L. Roffia, S. Mattarozzi, and V. Nannini, "Case Study: Context-aware Supervision of a Smart Maintenance Process," *Second International Workshop on Semantic Interoperability for Smart Spaces (SISS 2011)*. Munich; Germany: IEEE computer society, pp. 309-314, 2011.
[26] D. Beckett, "The design and implementation of the Redland RDF application framework," *Computer Networks*, vol. 39, Issue 5, pp. 577-588, 5 August 2002.
[27] Smart M3 SourceForge: http://sourceforge.net/projects/smart-m3/
[28] O. Lassila, "Enabling Semantic Web Programming by Integrating RDF and Common Lisp," *Proceedings of the First Semantic Web Working Symposium (SWWS'01)*, Stanford University, July 2001.
[29] D-BUS, http://www.freedesktop.org/wiki/Software/dbus
[30] A. D'Elia, J.Honkola, D.Manzaroli, T.Salmon Cinotti, "Access Control at Triple Level: Specification and Enforcement of a Simple RDF Model to Support Concurrent Applications in Smart Environments", *Smart Spaces and Next Generation Wired/Wireless Networking*. (pp. 63-74), Sergey Balandin, Yevgeni Koucheryavy, Honglin Hu. ISBN: 978-3-642-22874-2. LNCS 6869, presented at ruSMART11 (4th Conference on Smart Spaces, St. Petersburg, August 2011). Heidelberg: Springer, 2011.
[31] APACHE JENA, http://jena.apache.org/
[32] SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/
[33] Protégé Ontology Editor, Stanford University School of Medicine, http://protege.stanford.edu/

[34] A. Lomov, D. Korzun, "Subscription Operation in Smart-M3," Proc. *10th Conf. Open Innovations Association FRUCT and 2nd Finnish-Russian Mobile Linux Summit*, Tampere, Finland, 7-11 Nov. 2011, pp.83-94, 2011.
[35] Redland RDF, http://en.wikipedia.org/wiki/Redland_RDF_Application_Framework
[36] Raptor RDF library, http://librdf.org/raptor/
[37] Rasqal RDF query library, http://librdf.org/rasqal/
[38] Valgrind, http://valgrind.org/
[39] Fit-PC2, http://www.fit-pc.com/web/fit-pc2-models/
[40] Linux Mint, http://linuxmint.com
[41] M.Etelapera, J. Kiljander, K. Keinanen, "Feasibility Evaluation of M3 Smart Space Broker Implementations", *Applications and the Internet (SAINT)*, 2011 IEEE/IPSJ 11th International Symposium on Digital Object Identifier, Munich, Bavaria, 2011.
[42] S. Marchenkov, P. Vanag, D. G. Korzun, "Evaluation of the Smart Space Approach in Mobile Data Processing". *Proc. 11th Conf. Open Innovations Association FRUCT*, St.-Petersburg, Russia, 23-27 Apr. 2012. pp.194-195, 2012.
[43] Secure Smart Chat Project, http://www.open-m3.org/?q=node/73#overlay-context=Projects%3Fq%3DProjects
[44] J. Kiljander, F. Morandi, J-P. Soininen, "Knowledge Sharing Protocol for Smart Spaces," *Int. Journal of Advanced Computer Science and Applications*, Vol. 3, No. 9, 2012.