Design of Semantic Information Broker for Localized Computing Environments in the Internet of Things

Ivan V. Galov, Aleksandr A. Lomov, Dmitry G. Korzun
Petrozavodsk State University (PetrSU)
Petrozavodsk, Russia
{galov, lomov, dkorzun}@cs.karelia.ru

***Abstract*—Emerging communication technologies of the Internet of Things (IoT) make all the devices of a spatial-limited physical computing environment locally interconnected as well as connected to the Internet. Software agents running on devices make the latter "smart objects" that are visible in our daily lives as real participating entities. Based on the M3 architecture for smart spaces, we consider the problem of creating a smart space deploying a Semantic Information Broker (SIB) in a localized IoT-environment. SIB supports agent interaction in the smart space via sharing and self-generating information and its semantics. This paper proposes a renewed SIB design with increased extensibility, dependability, and portability. The research done is a step towards an efficient open interoperability platform for the smart space application development.**

I. Introduction

The Internet of Things (IoT) refers to the connection of physical objects. IoT technologies make all the devices of a spatial-limited physical computing environment interconnected as well as connected to the Internet. This ability leads to the consideration of notion of localized IoT-environments [1], [2], which now appears in many places of everyday life. Software agents running on devices turn the latter into "smart objects" that are visible in our daily lives as real participating entities. The next generation of software applications (smart applications) can be deployed in localized IoT-environments based on the smart spaces paradigm [3]. It allows creating a smart environment that "is able to acquire and apply knowledge about its environment and to adapt to its inhabitants in order to improve their experience in that environment" [4].

We consider the M3 architecture (multi-device, multi-vendor, multi-domain) for smart spaces [5], [6], [7], [1]. It utilizes the blackboard and publish/subscribe communication models to support interaction of agents via sharing information of the environment, rather than making agents to send messages to one another directly. The information and its semantics are collected in a smart space using ontological representation models of the Semantic Web and forming a knowledge base for interoperable sharing among participants. A smart space is maintained by a Semantic Information Broker (SIB). The latter is deployed on a host accessible by any device of the localized IoT-environment. SIB maintains an RDF triplestore, which represents the smart space content and acts as an informational hub relating many data sources. SIB provides access and reasoning primitives for agents to operate over the collected information. The content is generated by agents themselves.

There are several SIB implementations available: the first official Smart-M3 SIB [8], RIBS [9] for resource limited devices, OSGi SIB [10] for Java-based systems, and RedSIB [11] as evolution of the Smart-M3 SIB. They have grown from the SEDVICE system [12], where the M3 architecture was first implemented and experimented. The existing SIB implementations clearly showed certain elegant properties of the M3 architecture. On the other hand, the application development needs consideration of many technical aspects in order to achieve satisfactory operation in real-life settings.

In this paper, we propose a renewed SIB design with increased extensibility, dependability, and portability. We consider this design as further evolution of RedSIB. The modular approach is applied for high extensibility, similarly to [13]. Interprocess communication in SIB is optimized along the path from agent request arrival to SIB till the result is formed for delivering to the agent which results in higher dependability. The pool of access operations is systematized based on the original Smart Spaces Access Protocol (SSAP) and its variants as well as extended with mechanisms previously introduced in Knowledge Sharing protocol (KSP). The C/C++ programming language and Qt framework are used for the SIB implementation for higher portability.

The rest of the paper is organized as follows. Section II presents existing space-based approaches that exploit Semantic Web technologies for interoperable information sharing. Section III introduces the M3 architecture for SIB-based smart spaces with indirect information exchange in agent interaction. Section IV describes our architecture of the renewed SIB, which is based on the original Smart-M3 SIB and its successor RedSIB. Section V considers access operations to smart space content, which are basic elements for programming agent interaction. Section VI summarizes the key properties of the proposed SIB design.

II. Related work

Several software platforms are available for creating semantic-oriented and data-centric computing environments based on the paradigm of triple space computing. The paradigm applies the blackboard model for networked communication and employs Resource Description Framework (RDF) of the Semantic Web for information representation of shared

content. The platforms have mostly the status of research prototype; they focus on feasibility study of the paradigm and on proof-of-the-concept elaboration of possible solutions for application development.

In the TripCom project [14], an RDF triple space distributes its content in an overlay network made up of many kernels. The kernels form a distributed hash table, each kernel stores a subset of triples. A client knows one of the kernels to address any query despite in which kernel the triples are stored. The TripCom design achieves high scalability, enabling to store huge amount of RDF triples. Nevertheless, TripCom software clients are oriented to run on powerful machines.

In the TSC framework [15], a triple space represents its content as RDF graphs made up from triples and applies semantic matching algorithms for searching in such graphs. TSC describes transactional context and provides operations for publishing and retrieval. In the publish/subscribe mechanism, any user can subscribe using a template and receive notification whenever an update matches the query. Nevertheless, TSC does not support knowledge inference and does not allow expressive querying for searching the space content. The TSC middleware focuses on large scale architecture, preventing the use of low capacity devices.

In the SENS system [16] (Semantic Event Notification Service), a triple space employs a reasoning engine. The latter makes knowledge inference after every publication. The inference is based on the whole content the space stores, in contrast to methods with query operation for information subset. Such a triple space stores directly all inferred knowledge at the price of the performance of publish operation.

In SemWebSpaces [17] (Semantic Web Spaces), a triple space employs matching based on RDFS reasoning capabilities. Space content is a set of RDFTuples. Each tuple consists of the four fields: subject, predicate and object of an RDF statement and an identifier for the tuple based on tuplespace ontology. SemWebSpaces uses RDF reasoners, and RDFTuples introduced to the space can be checked for ontological conformance. SemWebSpaces also defines an ontology for describing the space itself. SemWebSpaces is a lightweight platform aiming at minimal system footprint and simplicity/flexibility of code. The platform does not support security mechanisms and has issues with scalability.

In Conceptual Spaces [18] (CSpaces), a space is a finite set of ontologies, their instances, and mapping and transformation rules. All these elements are represented using a common formal language that allows the ontologies to be enriched with rules. CSpace consists of raw sub-space and reasoning sub-space. The former stores imported or local data, including ontologies and mappings. The latter provides a compact representation of an associate raw sub-space, increasing the reasoning performance. Each reasoning sub-space is periodically regenerated with the latest version of the raw sub-space. Dedicated machines—CSpaces servers—store the content published in CSpaces, provide an access point for CSpace clients, include reasoning services for evaluating complex queries, implement subscription mechanisms, and balance workload.

The above triple space platforms allow constructing various computing environments to process information and implement semantic-aware services. The ideas of information sharing via triple space and subscriptions encouraged the development of M3 architecture for smart spaces [12] and its particular implementation—Smart-M3 platform [8].

III. Smart Spaces

In wide sense, smart spaces form a programming paradigm for a certain class of ubiquitous computing environments [4], [19], [5]. Consider localized IoT-environments; each is associated with a physical spatial-restricted place (office, room, home, city square, etc.) and, in addition to local networking, has access to the Internet. Such an environment is equipped with variety of devices: sensors, data processors, actuators, consumer electronics, personal mobile devices, multimodal systems, etc.

Deployed in an IoT-environment, the smart space makes the environment "smart" to handle the number of devices and the amount of information to be processed [12], [3]. Cooperation of devices is supported by establishing a shared view of resources in the environment. Software part of a smart environment includes two sides: "agents" and an information "hub". Participation of a device is determined by its software agent. Each agent produces its share of information and makes it available to others via the hub. Similarly, an agent consumes information of its own interest from the hub. That is, a hub is a server that realizes a shared information space for the required cooperation.

We focus on a particular approach for creating smart spaces—the M3 architecture [5], [6], [7], [1]. It aims at information-level compatibility, rather than promoting the compatibility within one specific service-level solution. Collaboration between different producers and consumers of information happens on a more abstract level. Agents interact on the semantic level, utilizing (potentially different) existing underlying services. The key architectural component is Semantic Information Broker (SIB) that implements an information hub for agents of a given environment. SIB maintains an RDF triplestore forming a common knowledge base [20]. Agents are also called knowledge processors (KPs). Communication between them is indirect, it is implemented through exchange of triples via SIB. High device interoperability is achieved since KPs share information via SIB, rather than have the underlying devices explicitly send messages to one another. Information interoperability problem is solved due to RDF representation model.

Interaction between agents is based on operations (smart spaces access primitives) with shared content. They are inherited from semantic space computing [21]; Table I provides a summary. Operations publish and read are traditional database-like operations. Operation take retrieves and then deletes the data fragment. Blocking operations (with prefix b-) wait until at least one result is returned. Operation subscribe is a persistent query tracking changes in a specified part of the content. In such indirect interactions, network communication between a KP and its SIB follows specialized protocols. The original implementation is Smart Spaces Access Protocol (SSAP) [8] and its variants [22] such as Knowledge Sharing Protocol [23] for smart spaces.

When considering localized IoT-environments, the following properties of SIB must be primarily taken into account.

TABLE I. CONTENT ACCESS OPERATIONS IN TRIPLE SPACE COMPUTING

	TripCom	TSC	SENS	SemWebSpaces	CSpaces	*Smart-M3 (SSAP)*
publish	out	create	publish	outr, claim	write	insert
read	rda, rd, rdg without timeout parameter	read	tryReceive	endorse, extract (non-blocking mode)	read	query
b-read	rda, rd, rdg with timeout parameter	waitToRead	receive	endorse, extract (blocking mode)	waitToRead	{not available}
take	ina, in, ing without timeout parameter	take	tryDelete	in (non-blocking mode)	take	remove (only delete)
b-take	ina, in, ing with timeout parameter	waitToTake	delete	in (blocking mode)	waitToTake	{not available}
subscribe	subscribe	subscribe	subscribe	{not available}	subscribe	subscribe

Simplicity: SIB architecture is easy to elaborate, evolve and understand by third-party developers.
Extensibility: SIB architecture provides an easy way of adding functionality.
Dependability: SIB operation is stable and can run continuously without failures for a long period of time. In case of failures SIB is able to recover its working state.
Portability: Various computing devices can be hosts for SIB, including Linux and Windows based systems as well as such embedded systems as OpenWrt on routers.

There exist several SIB implementations that follow the M3 architecture. None of them fully satisfies the above properties. Our proposal for new design and implementation of SIB is presented further in Section IV.

Smart-M3 SIB [8] is the first official SIB reference implementation. Its open source code is written in C language, with strong dependence on the glib library and the Piglet triplestore to operate with RDF data. Smart-M3 SIB provides support for the original SSAP protocol and for extended search queries in Wilbur query language. The SPARQL query language is not supported. There are problems with performance, especially with subscription when it is actively used.

Plug-in SIB approach [13] is used to extend the Smart-M3 SIB functionality. Each plug-in aims at additional processing of the smart space content.

RIBS [9] is a secure and fast SIB implementation, which is proprietary. RIBS targets for resource limited devices. Pure ANSI C code is used with minimal system dependencies. Bitcube triplestore is employed, which provides random access lookup operation with constant latency in lieu of cubical memory consumption.

OSGi SIB [10] is a Java-based SIB implementation. The OSGi technology is utilized to achieve a high level of modularity and portability. Jena framework with Pellet is utilized to effectively operate with RDF data. OSGi SIB supports SPARQL queries and reasoning rules over RDF data. This SIB implementation, however, is resource-demanding due to usage of Java. Besides software agents from other SIB implementations are incompatible with this SIB version.

RedSIB [11] is evolution of the original Smart-M3 SIB implementation. The Redland library is employed to maintain a triplestore and SPARQL support. The subscription mechanism is refactored for better performance. RedSIB (as well as Smart-M3 SIB) consists of two subsystems (sib-daemon and sib-tcp). D-BUS communication system is used for exchanging data between these subsystems. Since D-BUS is not intended for transferring big amounts of data the efficiency essentially suffers.

IV. THE SEMANTIC INFORMATION BROKER

Our SIB implementation is based on Smart-M3 SIB and RedSIB. We improve modularity and portability of the SIB design. These two properties are borrowed from OSGi SIB and RIBS, respectively. Based on the RedSIB source code, we employ the Redland library, for representing smart space content in an RDF triplestore and for implementing operations over the content (read&write, search query, reasoning).

The Smart-M3 SIB architecture is shown in Fig. 1, where the main subsystems are Network part, Operation logic, Triplestore. Network part handles network requests and forwards them to the operation logic. Connection between Network part (e.g., sib-tcp) and Operation logic (sib-daemon) uses D-BUS inter-process communication system. Operation logic processes operation request (SSAP operation or SPARQL queries) and performs corresponding changes in Triplestore (Redland library). Redland provides common interface for operating with triplestores of different kinds (e.g., memory, files, sqlite).

The RedSIB implementation has complicated code structure, which makes difficult to modify and improve SIB functions. The main difficulty is a multithread implementation where several thread pools and asynchronous queues are used. After receiving a request, sib-tcp forwards it to sib-daemon using D-BUS. In sib-daemon, a new thread is created for every request and then it is pushed to the asynchronous queue. Scheduler processes the requests in round-robin cycle

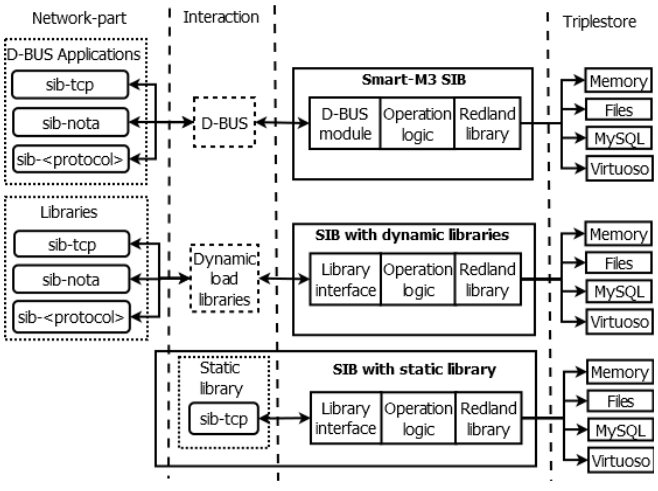

Fig. 1. The Smart-M3 SIB implementation is essentially based on D-BUS. In our architecture, SIB interface with network is implemented using dynamic or static libraries

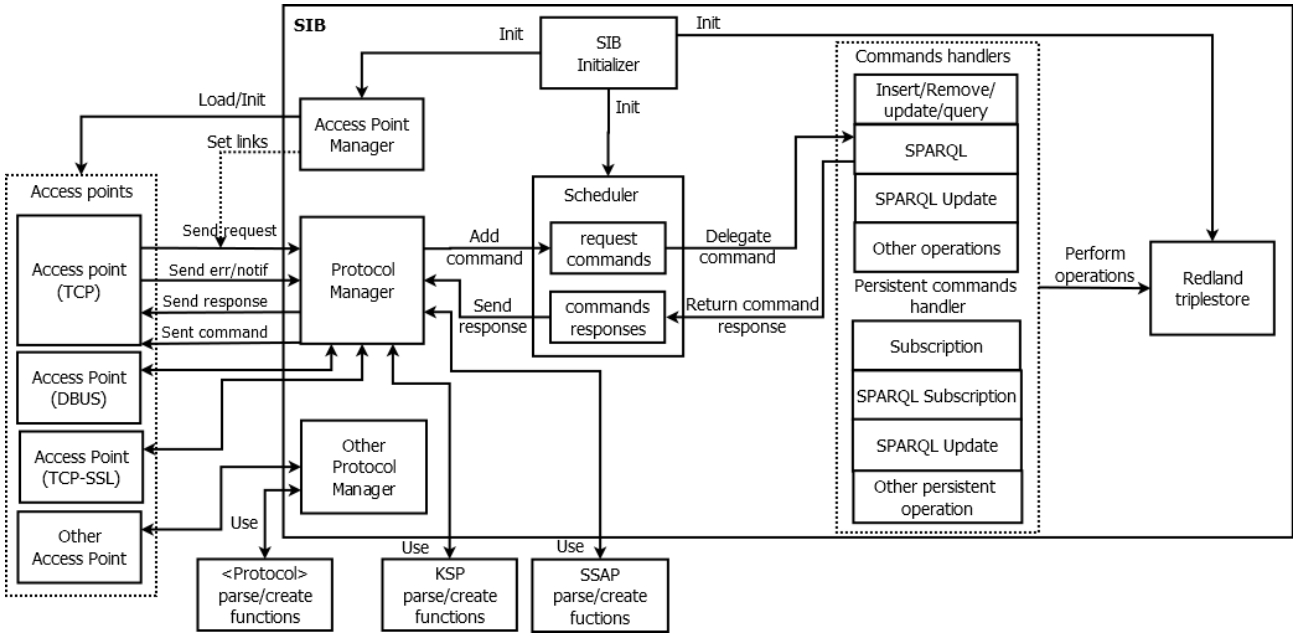

Fig. 2. The proposed Plug-ins based SIB architecture: achieving the extensibility of SIB functionality

in a separate thread and performs corresponding operations in Triplestore. Sib-daemon does not have independent modules to process data exchange in effective representation format (e.g., it processes messages directly in the D-BUS format). The implementation of subscription operation makes the SIB processes structure even more complicated. For every subscription, two threads are created: one in sib-tcp and another in sib-daemon. When subscription indication is needed it initiates a long chain of function calls in several threads. Our goal is to simplify the SIB architecture to make it more extensible and customizable for further improvements and understandable for third-party developers.

Our first direction of SIB development is eliminating D-BUS. One reason is that D-BUS is used only in Linux systems, thus preventing the use of 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. We propose the SIB architecture without these two processes communicating via D-BUS.

The new SIB architecture was depicted in Fig. 1 above. SIB communication modules for various network protocols (e.g., TCP, Nota) become plug-ins. They can be loaded/unloaded from the main SIB program (SIB with dynamic libraries). When higher portability is needed, such plug-ins can be integrated to SIB using static compilation (SIB with static library). 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.

This D-BUS elimination does not affect the functional SIB capabilities. In the case with D-BUS, sib-daemon and sib-tcp are standalone applications, where sib-daemon does not control sib-tcp. Indeed, the flexibility is that sib-tcp can be started or stopped manually in any moment. If Network part is implemented as plug-ins then SIB loads and controls plug-in lifecycle (loading/initialization/unloading). This way, Network part becomes easier for developing and testing. SIB can track network errors and restart/reinitialize network plug-ins in case of errors. Consequently, the new SIB architecture achieves higher dependability.

Our second direction of SIB development is to simplify the architecture in order to achieve higher extensibility. The architectural details are presented in Fig. 2. This SIB architecture is modular, allowing inclusion/exclusion of certain modules in compilation phase or in runtime. The feature affords to customize SIB functionality for given host device and IoT environment. The main module is SIB initializer. It prepares SIB for operation and initializes other SIB modules according to configuration from configuration file or command line parameters. The module initializes access points, scheduler, and Redland triplestore.

Network part is implemented as a pool of access points, each is an external module for SIB. Access point binds to particular network protocol (TCP or UDP) and port. Each access point receives KP requests and sends corresponding responses. Protocol manager interacts with a specific access point and performs request parsing and response generation. Access protocol (such as SSAP or KSP) is implemented as a separate module, which parses request messages and creates response messages. Access point manager initializes its access point and sets protocol managers. One access point is associated with a certain protocol manager module. There can be several access points of the same type (e.g., several TCP listeners running on different ports). SIB can be run with several protocol managers to handle additional access methods. For example, it is possible to implement SPARQL over HTTP to access SIB as a common SPARQL access point.

Each access point sends two types of messages to its

protocol manager: requests from agents and network errors/notifications (e.g., agent is disconnected). Similarly, each protocol manager sends two types of messages to access point: responses to agents and commands (e.g., shutdown, disconnect agent). While responses implement messages of a particular access protocol, commands are targeted only for access point and are used to control network connection.

Access points can be loaded/unloaded on the fly, thus it is possible to enable/disable different protocols support in SIB without shutting it down. An access point is a separate library, making difficult or even impossible to load such libraries for some embedded devices. To overcome this problem architecture allows to compile SIB with certain access points internally as was described above for SIB with static library (Fig. 1).

The plug-ins approach of [13] allows adding plug-ins into the Smart-M3 SIB implementation. They introduce additional processing of RDF-represented content. In our SIB architecture, such functionality can be added using handlers. They are used for basic functionality of SIB (triplestore operations, subscription). A handler can be statically compiled with SIB, which is a default way to implement handler for basic functionality. These modules can be chosen in compilation time of SIB. Other handlers, which extends SIB functionality, can be implemented for loading in runtime.

Scheduler module controls processing of commands among modules. It processes commands with KPs requests/responses and internal notifications (to control runtime of other modules). After receiving a command the scheduler delegates it to an appropriate command handler. Three command handlers can be distinguished from the RedSIB version: base operation handler (insert/remove/update/query), SPARQL handler, SPARQL Update handler. Similarly, three persistent command handlers are: base subscription handler, SPARQL subscription and SPARQL Update. Persistent command contains persistent operation (subscription or SPARQL). Persistent operation is always stored on the SIB side (continuous in time) and a response is generated whenever a specified event occurs.

Our SIB architecture is designed with the purpose of hosting SIB on various devices and platforms. In addition to Linux-based platforms, the new SIB implementation targets Windows OS machines and embedded devices (such as routers with OpenWrt firmware). The architecture assumes the ability to include/exclude particular modules on compilation phase and in runtime mode. It becomes possible to produce limited SIB executables to run on restricted devices for the cost of reduced functionality.

Let us summarize the distinguished properties of the proposed SIB design. As a result of the D-BUS elimination, the interprocess communication has simpler structure. SIB can be easier ported to other platforms. The modular approach facilitates evolution of the SIB implementation. Based on the plug-ins approach, the SIB architecture supports many access protocols, including the original SSAP.

V. Agent Interaction in a Smart Space

SSAP [8] is the first official protocol for operation-based network communication between an agent (KP) and its SIB. On the other hand, KSP protocol is oriented to use on an agent which is hosted on a constrained device [22], [23]. According to the plug-ins based architecture, SIB can support many different access protocols. Moreover, the access can be extended with SPARQL queries over HTTP such that SIB acts like a SPARQL endpoint.

TABLE II. System of SIB operations

Type		Operations
Basic operations	Session management	Join, Leave
	Content access and management	Instant Query, Insert, Remove, Update
	Persistent operations	(Un)Subscribe
Extended operations		Persistent Insert, Remove, Update
		SIB configuration rules
SPARQL operations	SPARQL	SELECT, CONSTRUCT, ASK, DESCRIBE
	SPARQL Update	INSERT, DELETE, INSERT DATA, DELETE DATA

Table II systemizes all operations that our SIB design supports. Basic operations include the original SSAP operations.

1) Session management (joining & leaving the smart space).
2) Content access and management (reading & writing & maintaining the RDF triplestore).
3) Persistent operations (querying in time period).

Session management contains operations Join and Leave. To access a smart space its agent establishes an SSAP session with SIB using Join. A session provides a time-lengthy substrate for the agent to communicate with SIB. After successful Join the agent performs operations with the content (manipulate with triples, subscriptions). To terminate its session the agents calls Leave, and all information associated with the session on the SIB side is removed, including closing all agent's subscriptions. Session-based access is optional; agents on restricted devices may prefer one-time access operations with SIB.

Sessions support management of agent's privileges on the SIB side. Let us distinguish public and private actions in the smart space. A public action does not require any special privileges to perform the operation and rights to access the needed content from the smart space. For example, action "getting current temperature" is public if the "query" operation has no special rights for execution and the temperature value (encoded in RDF triples) has no access restrictions. Otherwise, such an action is private. For private actions the agent can establish a session for the authentication (a smart space participant) and authorization (to perform the action).

Content access and management operations manipulate with triples in the RDF triplestore (see also Table I). Basic operations are read&write operations of SSAP: search Query, Insert, Remove, and Update. They are instant, i.e., in traditional "one query–one result" style. SSAP does not support take operation and blocking operations. Update implements a transaction composed of subsequent Insert and Remove.

In the RedSIB implementation, SSAP was enhanced with support for SPARQL and SPARQL Update queries (due to Redland library). The implementation supports 1) SELECT, CONSTRUCT, ASK, and DESCRIBE query of SPARQL and 2) INSERT, DELETE, INSERT DATA, and DELETE DATA

queries of SPARQL Update. Note that these SPARQL possibilities generalize the basic read&write operations of SSAP.

For a persistent operation, the agent registers its operation in SIB. The latter associates this operation with its agent and continuously keeps the operation state on the SIB side. SIB activates the operation whenever an appropriate action occurs in the smart space. Activation can happen many times until the agent terminates the operation. After termination, SIB removes the operation state. SSAP supports only one persistent operation—subscription. Its registration can use triple templates or SPARQL SELECT query to specify the content of agent's interest. The agent receives a subscription notification whenever the specified part of content is changed (e.g., due to activity of other agents).

The set of persistent operations can be extended with persistent Insert and persistent Remove [22]. In these operations, SIB, on behalf of the agent, performs updating and removal of the content whenever SIB detects specified conditions. A persistent operation can be extended with SPARQL Update. Such extensions were proposed in KSP.

The difference of KSP and SSAP is emphasized below.

- Binary format (important for low capacity devices). The binary format is compact and faster to parse, but not as versatile as the XML format used in SSAP.
- KSP transactions are based on the SPARQL 1.1 (and SPARQL UPDATE) only. In SSAP, SPARQL is one of the three query formats.
- KSP does not require join and leave operations. Access control parameters can be added explicitly to every KSP message when session-based communication is not needed.
- KSP allows agents to define the maximum size for SIB response. Hence restricted devices can limit consumption with partial content found in the smart space.
- In addition to Subscription, KSP defines other persistent operations, which continuously change the smart space content (on behalf of their agents).

Persistent operations can be used internally in SIB to implement configuration of the automatic content maintenance and transformation process. This way, a SIB administrator can define actions that SIB performs over the collected content. Examples are information consistency recovery or rights check of joining agents. Otherwise, this process should be implemented as a dedicated agent detecting appropriate changes in the content and reacting correspondingly [24]. SIB configuration rules are persistent operation bindings to some events. The main difference is that rules are set by a SIB administrator while persistent operations are called by agents. Note that SPARQL is not the only way to define such rules. Another promising approach is answer set programming [25], when SIB is enhanced with artificial reasoners for the content.

In our SIB architecture (see Fig. 2 above), operations implementation is structured with modules. Session management operations are implemented in Protocol Manager. Content access and management operations are implemented in command handlers (basic operations, SPARQL operations, etc.). Subscription and other persistent operations are implemented in the persistent command handlers. The proposed modularity allows turning on/off modules to deploy SIB with a selected tradeoff between required functionality and available capacity.

Now consider the sequence of processing steps on the SIB side for any SSAP operation in our SIB architecture. The sequence is modeled in Fig. 3. The agent (KP) sends an SSAP (or another) request. The latter is delivered to SIB using some network transport protocol (e.g., TCP). A network listener—a particular Access Point—receives the request. Network requests can be processed either in one thread or in several threads. This is an issue of further studies. Access Point forwards the request to Protocol Manager for parsing (is implemented in separate module—Message Manager—which parses messages of particular protocol) and converting the operation into a scheduler command. Then command goes to Scheduler, which receives commands in multiple threads. According to command type (insert, remove, SPARQL request, subscription, etc.), Scheduler forwards the command to corresponding Command Handler. The latter module performs particular operation in the RDF triplestore. Note that our SIB design preserves the property of processing each command in a separate thread, which is important for serving several commands of multiple agents in parallel.

When an operation has been completed in the RDF triplestore, Command handler forms a response command with operation result. Scheduler forwards the response command to corresponding Protocol Manager, which constructs a response message for Access point. Then the response is sent and delivered over the network to the agent.

The most complicated processing is related to subscription. Our SIB design is based on the RedSIB subscription mechanism, which is shown in Fig. 4. When Scheduler forwards a subscription command to Subscription handler, the latter performs a query request, creates a separate RDF store (subworld, to store associated triples), and runs a separate thread for this subscription. Then, whenever some Command handler performs changes in the triplestore, this handler notifies Subscription handler. Every subscription thread checks the changes for matching. Correspondingly, subscription indication responses are sent to the agents. New SIB implementation fully borrows subscription mechanism from RedSIB although it can be modified or replaced on another mechanism in the future.

New version of SIB will be implemented using Qt framework and provide functionality similar to the RedSIB implementation. So it will support all operations from original SSAP implementation although pool of commands can be extended with new operations (for example, blocking operations from Table I). SIB architecture presented in this paper is quite simple to extend and change by third-party developers. All new functions (including new operations) are added as modules. Due to such modularity SIB extensibility is achieved. Thus at the deployment phase SIB can be configured to settings of a given IoT environment. New SIB implementation will rely mostly on rich capabilities of Qt framework (network connections processing, threads, modules interaction and plugins). Well-testes and optimized Qt mechanisms contribute SIB dependability and portability that allows to run it on different platforms.

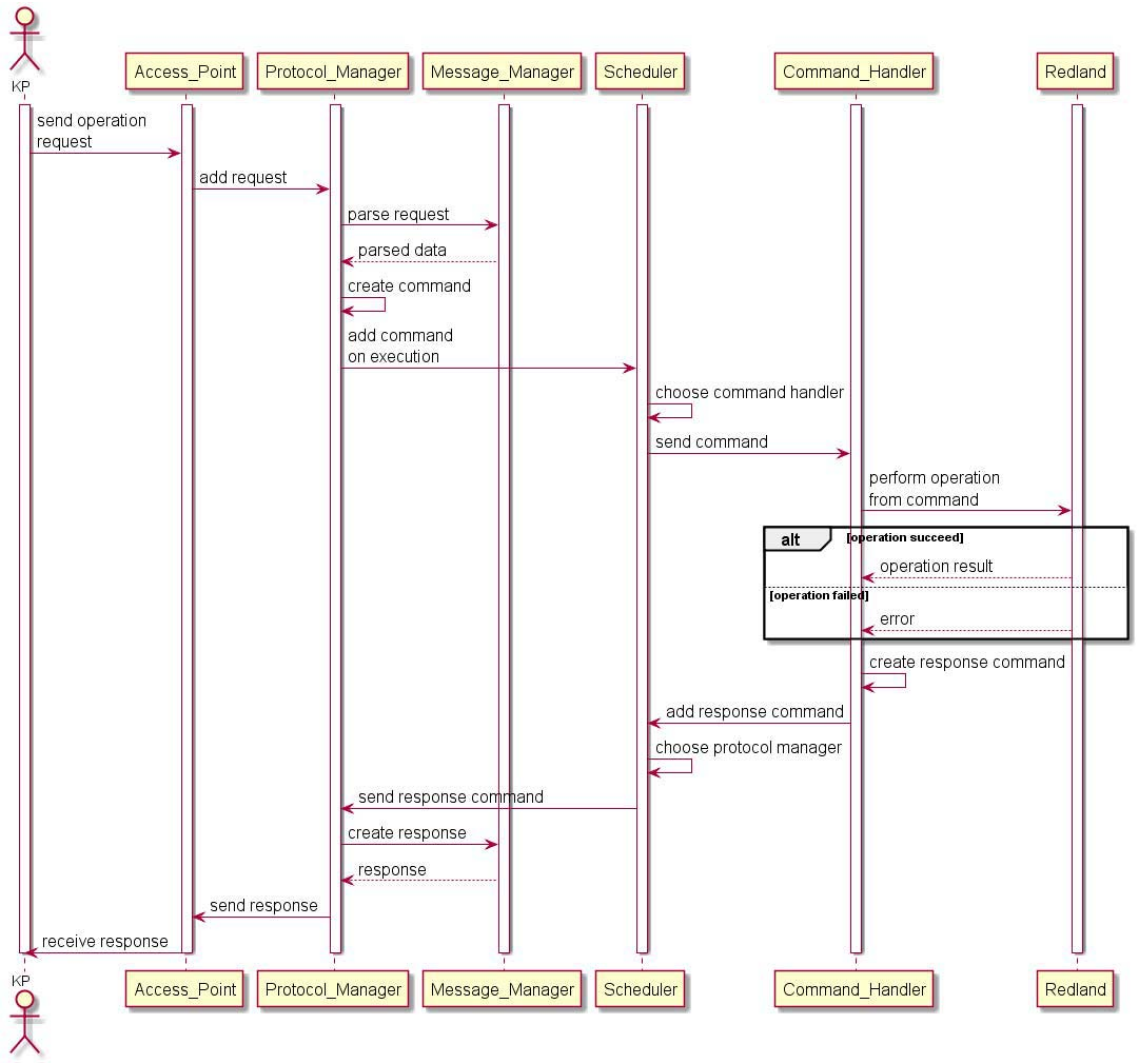

Fig. 3. Sequence diagram for operation processing in SIB

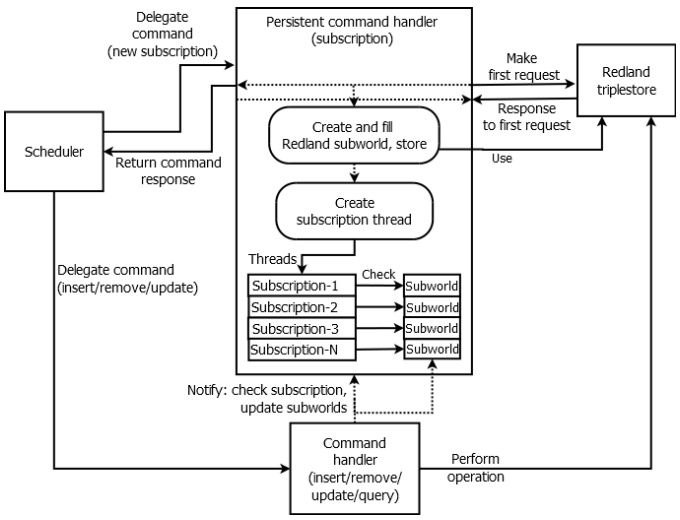

Fig. 4. Scheme of subscription processing in RedSIB

VI. Conclusion

This paper provided the renewed SIB design, which can be considered a step towards an efficient open interoperability platform for the smart space application development. The design takes into account such properties as extensibility, dependability, and portability, which are fully satisfied in none of the existing SIB implementations. On one hand, the compatibility with SSAP is preserved, making an ease transition from the Smart-M3 SIB or RedSIB for the large set already developed applications. On the other hand, new opportunities for application development appear, especially related to advanced smart space access operations.

Acknowledgment

This research is financially supported by the Ministry of Education and Science of Russia within project # 648-14 of the basic part of state research assignment for 2014–2016. The reported study was supported by the Russian Foundation for Basic Research, research project # 14-07-00252. The authors are grateful to all anonymous reviewers for their valuable comments. We would like to thank Sergey Balandin, Alexey Kashevnik, and Kirill Krinkin for their feedback and expertise.

REFERENCES

[1] 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*, S. Balandin, S. Andreev, and Y. Koucheryavy, Eds. Springer-Verlag, Aug. 2013, pp. 48–59.

[2] B. Vlist, G. Niezen, S. Rapp, J. Hu, and L. Feijs, "Configuring and controlling ubiquitous computing infrastructure with semantic connections: A tangible and an AR approach," *Personal Ubiquitous Comput.*, vol. 17, no. 4, pp. 783–799, Apr. 2013.

[3] D. Korzun, "Service formalism and architectural abstractions for smart space applications," in *Proc. 10th Central & Eastern European Software Engineering Conference in Russia (CEE-SECR 2014)*. ACM, Oct. 2014.

[4] D. J. Cook and S. K. Das, "How smart are our environments? an updated look at the state of the art," *Pervasive and Mobile Computing*, vol. 3, no. 2, pp. 53–73, 2007.

[5] 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 (UAHCI '09). Part II: Intelligent and Ubiquitous Interaction Environments, LNCS 5615*, C. Stephanidis, Ed. Springer-Verlag, 2009, pp. 3–12.

[6] J. Kiljander, A. Ylisaukko-oja, J. Takalo-Mattila, M. Eteläperä, and J.-P. Soininen, "Enabling semantic technology empowered smart spaces," *Journal of Computer Networks and Communications*, vol. 2012, pp. 1–14, 2012.

[7] E. Ovaska, T. S. Cinotti, and A. Toninelli, "The design principles and practices of interoperable smart spaces," in *Advanced Design Approaches to Emerging Software Systems: Principles, Methodology and Tools*, X. Liu and Y. Li, Eds. IGI Global, 2012, pp. 18–47.

[8] 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, Jun. 2010, pp. 1041–1046.

[9] 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.

[10] D. Manzaroli, L. Roffia, T. S. Cinotti, E. Ovaska, P. Azzoni, V. Nannini, and 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.

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

[12] I. Oliver and J. Honkola, "Personal semantic web through a space based computing environment," *Computing Research Repository (CoRR)*, vol. abs/0808.1455, Aug. 2008.

[13] P. Bellavista, V. Conti, C. Giannelli, and J. Honkola, "The Smart-M3 semantic information broker (SIB) plug-in extension: Implementation and evaluation experiences," in *IEEE International Conference on Green Computing and Communications (GreenCom)*. IEEE, 2012, pp. 704–711.

[14] E. Simperl, R. Krummenacher, and L. Nixon, "A coordination model for triplespace computing," in *Coordination Models and Languages*. Springer, 2007, pp. 1–18.

[15] D. Fensel, R. Krummenacher, O. Shafiq, E. Kuehn, J. Riemer, Y. Ding, and B. Draxler, "TSC–triple space computing," *e & i Elektrotechnik und Informationstechnik*, vol. 124, no. 1-2, pp. 31–38, 2007.

[16] M. Murth and E. Kühn, "Knowledge-based coordination with a reliable semantic subscription mechanism," in *Proceedings of the 2009 ACM symposium on Applied Computing*. ACM, 2009, pp. 1374–1380.

[17] L. Nixon, E. P. B. Simperl, O. Antonechko, and R. Tolksdorf, "Towards semantic tuplespace computing: the semantic web spaces system," in *Proc. 2007 ACM symp. Applied computing (SAC '07)*. ACM, 2007, pp. 360–365.

[18] F. Martín-Recuerda, "Towards Cspaces: A new perspective for the Semantic Web," in *Proc. 1st IFIP WG12.5 Working Conf. on Industrial Applications of Semantic Web*, ser. IFIP — The International Federation for Information Processing, M. Bramer and V. Terziyan, Eds., vol. 188. Springer, 2005, pp. 113–139.

[19] I. Oliver, "Information spaces as a basis for personalising the semantic web," in *Proc. 11th Int'l Conf. Enterprise Information Systems (ICEIS 2009)*, May 2009, pp. 179–184.

[20] C. Gutierrez, C. A. Hurtado, A. O. Mendelzon, and J. Pérez, "Foundations of semantic web databases," *J. Comput. Syst. Sci.*, vol. 77, no. 3, pp. 520–541, May 2011.

[21] M. Murth and E. Kühn, "Knowledge-based interaction patterns for semantic spaces," in *Proc. the 2010 Int'l Conf. on Complex, Intelligent and Software Intensive Systems (CISIS '10)*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1036–1043.

[22] J. Kiljander, A. D'elia, F. Morandi, P. Hyttinen, J. Takalo-Mattila, A. Ylisaukko-oja, J.-P. Soininen, and T. S. Cinotti, "Semantic interoperability architecture for pervasive computing and internet of things," *IEEE Access*, vol. 2, pp. 856–874, Aug. 2014.

[23] J. Kiljander, F. Morandi, and J.-P. Soininen, "Knowledge sharing protocol for smart spaces," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 3, pp. 100–110, 2012.

[24] S. Balandin, I. Oliver, S. Boldyrev, A. Smirnov, A. Kashevnik, and N. Shilov, "Anonymous agents coordination in smart spaces," in *Proc. 4th Int'l Conf. Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010)*, Oct. 2010, pp. 242–246.

[25] T. Janhunen and V. Luukkala, "Meta programming with answer sets for smart spaces," in *Proc. 6th Int'l Conf. Web Reasoning and Rule Systems (RR 2012)*. Springer, 2012, pp. 106–121.