Knowledge Sharing Protocol for Smart Spaces

Jussi Kiljander
VTT Technical Research Centre of Finland
Oulu, Finland

Francesco Morandi
ARCES, University of Bologna
Bologna, Italy

Juha-Pekka Soininen
VTT Technical Research Centre of Finland
Oulu, Finland

***Abstract*— In this paper we present a novel knowledge sharing protocol (KSP) for semantic technology empowered ubiquitous computing systems. In particular the protocol is designed for M3 which is a blackboard based semantic interoperability solution for smart spaces. The main difference between the KSP and existing work is that KSP provides SPARQL-like knowledge sharing mechanisms in compact binary format that is designed to be suitable also for resource restricted devices and networks. In order to evaluate the KSP in practice we implemented a case study in a prototype smart space, called Smart Greenhouse. In the case study the KSP messages were on average 70.09% and 87.08% shorter than the messages in existing M3 communication protocols. Because the KSP provides a mechanism for automating the interaction in smart spaces it was also possible to implement the case study with fewer messages than with other M3 communication protocols. This makes the KSP a better alternative for resource restricted devices in semantic technology empowered smart spaces.**

Keywords- Semantic Web; SPARQL; Ambient Intelligence; Ubiquitous Computing; embedded system; M3.

I. INTRODUCTION

Smart spaces are realizations of ubiquitous computing (ubicomp) [1] and ambient intelligence (AmI) [2] visions. A typical smart space consists of a large amount of devices which in co-operation provide services for users. In order to provide relevant services in the right situations the devices need to share knowledge about the smart space with each other. Fortunately, a lot of knowledge representation (KR) technologies have been developed for the emerging Future Internet paradigm, called the Semantic Web [3] that could be also exploited in ubicomp/AmI domain. This has also been proposed by Lassila [4], and Chen [5], for example. The M3 concept [6] is a recent example of ubicomp interoperability framework which utilizes semantic technologies for knowledge representation.

Many of the devices in smart spaces are resource restricted in terms of memory, processing capacity, and energy. Additionally, typical communication technologies in smart spaces such as the 6LowPAN [7], and Bluetooth low energy (BLE) [8], for example, possess limited capabilities when compared to the technologies used in the Web. On the other hand, the current technologies enabling KR in the Semantic Web such as Resource Description Framework (RDF) [9], RDF Schema (RDFS) [10], Web Ontology Language (OWL) [11], and SPARQL [12] use either Extensible Markup Language (XML) based or human readable[1] syntax. These formats both require a large amount of memory and are slow to process in low capacity computing platforms. As a result they are not as such feasible for real-life smart spaces. Binary XML formats such as Efficient XML Interchange (EXI) [13] and X.694 [14] provide feasible solutions for compressing XML, but cannot be used with non-XML based semantic technologies such as the SPARQL, for example. Another interesting approach for Semantic Web based KR in resource restricted devices is the Entity Notation (EN) [15]. As a lightweight KR notation the EN is a good alternative for typical RDF serialization formats such as the RDF/XML, Turtle, and N-Triple, but it does not provide SPARQL-like mechanisms to query and update knowledge in smart spaces.

In M3 applications the problem with resource restricted computing platforms has been typically solved by utilizing gateways which transform the proprietary format data from low capacity devices to semantic format. However, this approach complicates the system unnecessarily as for each new device a new gateway is needed (or new interface to existing gateway needs to be added). If all the communication between smart space agents would be based on common knowledge sharing protocol instead, the smart spaces would be much easier to develop and maintain. Additionally, since the common knowledge sharing protocol would enable also resource restricted devices to access the information published by other devices it would be easier to develop context-aware embedded systems capable of providing relevant services for users.

In this paper we present a novel Knowledge Sharing Protocol (KSP) for M3-like semantic interoperability frameworks. The KSP provides all kind of agents from low capacity embedded systems to high end personal computers with SPARQL-like mechanisms for accessing and manipulating the knowledge in smart spaces. Unlike normal SPARQL, however, the KSP is designed to be suitable for smart spaces. Instead of the sparse human readable syntax used in SPARQL the KSP uses a compact binary format which allows significantly shorter messages to be created. Features such as the persistent update and max request size option make it also easier to exploit semantic technologies in resource restricted devices and networks. Additionally, to support the

[1] By human readable syntax we refer to notations designed to be used by developers as such. These kinds of formats are used, for example, in SPARQL, Notation3, and Turtle.

heterogeneous nature of smart spaces the KSP defines various bindings for typical communication and networking technologies used in smart spaces. In the paper bindings for the most typical transports User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) are presented.

The rest of the paper is structured as follows. In the section 2 we present short overview of the SPARQL and M3 concepts as necessary background information for the paper. The section 3 describes the KSP in high detail. In the section 4 we illustrate the KSP with practical example and compare it with existing M3 communication protocols (M3CP). In the section 5 conclusions and future work directions are presented.

II. BACKGROUND

A. SPARQL

SPARQL provides the standard way to access and manipulate RDF data in the Semantic Web. The importance of SPARQL to the Semantic Web is formulated by the W3C Director Tim Berners-Lee as follows: "Trying to use the Semantic Web without SPARQL is like trying to use a relational database without SQL." [16]

SPARQL 1.1 defines four types of query forms: SELECT, CONSTRUCT, ASK and DESCRIBE. All these query forms use solutions obtained via pattern matching to form result sets or RDF graphs. The SELECT query returns the bound variables as such. The CONSTRUCT query returns an RDF graph constructed by substituting variables in a set of RDF triple patterns with bound variables obtained from the pattern matching. ASK query returns a Boolean value indicating whether the query has a solution or not. The DESCRIBE query returns a single RDF graph that contains data about the requested resource.

In addition to the query language the SPARQL 1.1 specification defines an update language for manipulating RDF data. The update operations are classified into two groups: graph management and graph update. Management type operations provide mechanisms for creating, destroying, moving and copying named graphs, or adding the contents of one graph to another. In contrast to the management operations used for operating on the graph level the update operations are used for modifying triples inside the graphs. The update operations are: INSERT DATA, DELETE DATA, DELETE/INSERT, LOAD and CLEAR. The DATA format operations operate on concrete RDF triples meaning that the use of variables is prohibited. Blank nodes are also not permitted in the DELETE DATA operation. The DELETE/INSERT operation is used to modify triples in the graph store based on bindings obtained via query pattern matching. It is possible to omit either DELETE or INSERT part of the operation in which case only the remaining part of the operation is executed. LOAD operation is used to insert triples from a RDF document specified by an URI into the graph store. CLEAR operation is used to remove all triples from the specified graphs.

SPARQL provides also many features that can be used to modify the outcome of query and update operations. These features include, for example, OPTIONAL graph pattern, FILTER, BIND, subquery, GROUP BY, and various solution sequence modifiers. The OPTIONAL graph pattern is used to declare a graph pattern which does not need to match in order for the query pattern match to succeed. FILTER keyword provides a way to restrict the solution to those in which the FILTER expression evaluates as true. The BIND feature allows a variable to be bound with a new value. The subquery feature makes it possible to combine results of multiple queries by allowing queries to be put inside of queries. The GROUP BY keyword allows sets of data to be grouped together so that aggregate functions such as average, minimum, maximum, sum, and count can be performed on the individual groups.

In SPARQL the results of the query operations can be modified using modifiers such as ORDER BY, LIMIT, OFFSET, and DISTINCT. The ORDER BY keyword can be used to sort the results of the query to either ascending or descending order. The LIMIT keyword allows the number of results to be restricted. The OFFSET keyword provides a way to skip a number of results. The DISTINC feature can be used to remove duplicates from the results set.

B. M3 Concept

The M3 aims to combine Semantic Web technologies with publish/subscribe-based blackboard [17] architecture to provide multi-device, multi-domain and multi-vendor solution to semantic level interoperability in smart spaces. Like in typical blackboard systems the idea in M3 is that knowledge is shared between various knowledge sources via common knowledge base.

The main advantage of the blackboard architecture is flexibility. In blackboard system the knowledge sources are not tied together in any way. The knowledge sources do not in fact have to know anything about each other – they just see the information published to the knowledge base. This makes it possible to add new, or remove existing knowledge sources without the need to reconfigure other knowledge sources. The blackboard system is also independent of any particular application domain. This means that the blackboard architecture can be used for sharing any kind of information and the shared information can be used in all kind of applications. Because of these features the blackboard architecture is ideal for smart spaces where it is not possible to *a priori* determine all the components or features needed in the future.

The difference between M3 and typical blackboard systems is that the knowledge in M3 is presented with semantic technologies such as RDF, RDFS, and OWL. There are many advantages in using the semantic technologies to present the knowledge in ubiquitous computing systems. First, because semantic technologies provide a natural way to describe any kind of knowledge as common machine-interpretable ontologies they make it easier to develop context-aware applications to smart spaces. Second, because of the flexible data model of RDF it is possible to add new information and create links between the information available in the knowledge base without breaking the existing applications. Third, the ontologies described with OWL and RDFS make it possible for knowledge sources to learn new

concepts much in the same way as humans use encyclopedias to describe unfamiliar words.

In M3 the blackboard is called Semantic Information Broker (SIB). The Knowledge Processor (KP) is the name for the knowledge sources. M3 applications are created by KPs who provide services for end-users by interacting with each other via the SIB. The Smart Space Access Protocol (SSAP) defines the rules for KP-SIB interaction.

There are currently two serialization formats for the SSAP available: SSAP/XML and SSAP/WAX. Both formats define eight operations: *join*, *leave*, *insert*, *remove*, *update*, *query*, *subscribe*, and *unsubscribe*. The SSAP/XML version provides three types of query operations: Triple, Wilbur, and SPARQL 1.0. The M3 implementation using the SSAP/XML is called Smart-M3 [18]. A Word Aligned XML (WAX) version of the SSAP was introduced to better support the exploitation of M3 in low capacity devices [19]. In SSAP/WAX encoding scheme each XML tag is 32 bit long. This allows more compact messages to be constructed. The SSAP/WAX is used in RIBS version of the M3 [20]. Wilbur queries are not supported in the RIBS.

III. KNOWLEDGE SHARING PROTOCOL

A. Overview

The objective of the KSP is to define the methods and syntax for knowledge sharing in AmI/ubicomb domain. The idea was to develop similar protocol to Semantic Web enhanced ubicomb systems, as the Constrained Application Protocol (CoAP) [21] is to the embedded web (i.e. the idea was to do the same for SPARQL/HTTP, as the CoAP has done for the HTTP). Initially we even planned to implement the KSP solely on top of CoAP. However, we soon understood that the heterogeneous nature of smart spaces requires the KSP to be usable also directly with various lower level transport technologies. In this paper bindings for TCP and UDP transports are presented. The principal model of KSP stack is illustrated in the fig. 1.

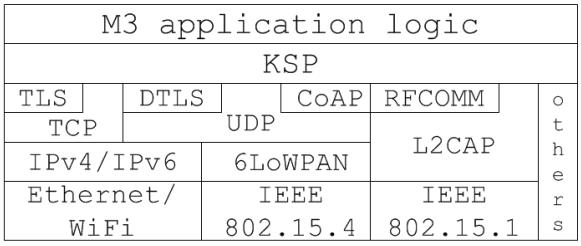

Figure 1. Knowledge Sharing Protocol stack

The different requirements of various transport technologies is managed in the KSP header presented in the section C. The KSP messages can contain also transport independent data and options fields. The structure for these fields is presented in sections D and E respectfully. All KSP messages are encoded in little endian format.

There are five major differences with the KSP and existing M3CPs (namely the SSAP/XML and SSAP/WAX). First, to provide a feasible solution for low capacity devices a binary format for the messages is used in KSP. The binary format is more compact and faster to parse, but not as versatile as the XML format used in the SSAP. Second, all the transactions in KSP are based on the SPARQL 1.1 whereas in SSAP the SPARQL is only one of the three query formats. Third, KSP does not require join and leave operations because the access control parameters can be added to any KSP message when needed. This makes the basic KP-SIB interaction more scalable because the SIBs do not have to keep track of the state of KPs. Fourth, the KSP allows KPs to define the maximum size for SIB responses. This is very useful for low capacity devices because the memory requirements can be estimated at the compile time. The fifth main difference is that the KSP defines persistent format also for update operations (i.e. DELETE, INSERT, and UPDATE). The persistent type update operations work so that the data manipulation part of the operation is executed if a solution is found from the query pattern matching. Similarly to the query operation the persistent update operations are re-evaluated every time the content of the SIB change. By allowing KPs to create simple rules to the SIB the persistent update operations provide a way to reduce the traffic in resource restricted networks and lighten the load on low capacity devices. The persistent operation are terminated with TERMINATE operation.

B. Messaging Model

The KP always initiates the transaction by sending a request (REQ) message to the SIB. To support the needs of the wide range of communication technologies the KSP defines two types of requests: Non-confirmable (NON), and Confirmable (CON). With NON request the SIB sends a response (RES) message only if results are found or an internal error has occurred. The NON requests are useful for TCP-like transports that provide a reliable delivery of messages and do not therefore require extra control from the KSP. Additionally, the NON request are useful for reducing the network traffic when it is not required that every message is delivered to the SIB. For example, a low capacity accelerometer updating acceleration value in high intervals does not have to care if a message is lost because it would take more time to update the old value than to insert a new one. The NON requests are also useful when KSP is used in multicast/broadcast manner to discover the available SIBs in the network. With CON type requests the SIB always sends a response to the KP. Therefore, the CON type request is typically used with transport technologies such as the UDP that do not provide reliable delivery of messages. The RES message is always send to the same socket where the REQ message was received

In addition to normal RES messages, the KSP defines indication (IND) messages which are used to notify the KP about changes in the persistent operations. The IND messages are typically used when the results of a persistent query operation (i.e. subscribe) change. Indications are also used to inform KP when a persistent operation needs to be terminated for some reason. Because transports such as UDP do not provide reliable delivery of messages the KSP provides acknowledgement (ACK) messages to be used with unreliable communication protocols to notify the SIB when the IND message has been received. The decisions on how long to wait before retransmitting and how many times to retransmit the

IND message are left for various SIB implementations. It is even possible to implement a SIB that can dynamically adjust to various networks by making these values modifiable via a specific graph in the SIB. The fig. 2 presents a sequence chart which illustrates the message exchange between KP and a SIB in a scenario where a KP1 subscribes to a triple and KP2 modifies the triple using UPDATE operation. Confirmable and Non-confirmable type requests are used by KP1 and KP2 respectfully.

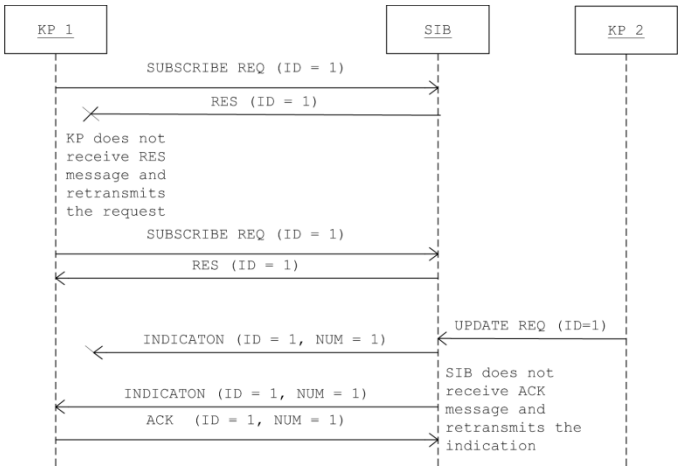

Figure 2. Example of message exchange with CON and NON requests

C. Message Format and Semantics: Header field

The fixed size header field contains parameters such as the version, transaction type, request type, and transaction identifier that are common for all transactions. The structure of the header field depends on the message type (REQ, RES, IND, or ACK) and the transport technology. The header size is eight bytes for TCP, four bytes for UDP REQ/RES/ACK messages, and six bytes for UDP Indications. Fig. 3 and fig. 4 illustrate the header formats for different message types with TCP and UDP transports.

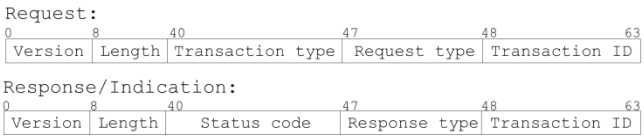

Figure 3. KPS header formats for TCP

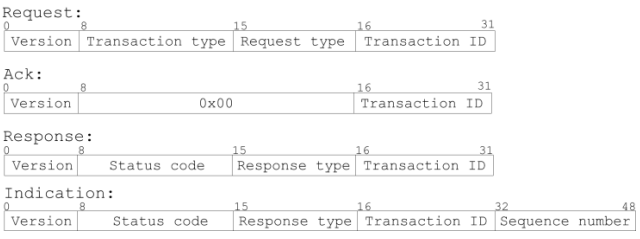

Figure 4. KSP header formats for UDP

The 8-bit *Version* field specifies the KSP version number. Value 0x01 is used for the version presented in the paper. With TCP the length of the KSP messages is defined in the 32-bit *Length* field. The length of the message is needed with TCP because the KSP message can be divided into multiple TCP segments. With UDP the *Length* field is not present and the whole KSP message must fit to a single UDP datagram. The maximum size for UDP datagram depends on the underlying protocols. For example, with 6LowPAN the maximum message size is 1024 bytes. In KP initiated messages the 7-bit *Transaction type* field specifies the type of the operation and the 1-bit *Request type* the type of the request (either NON or CON). Table 1 presents the code values for different transaction types. With connectionless transports the ACK message is identified by assigning value 0x00 for the *transaction type* and *request type* fields.

TABLE I. TRANSACTION TYPES

Transaction type	Code
DELETE DATA	0x01
INSERT DATA	0x02
UPDATE DATA	0x03
DELETE	0x04
INSERT	0x05
UPDATE	0x06
SELECT	0x07
ASK	0x08
CONSTRUCT	0x09
DELETE_PERSISTENT	0x0a
INSERT_PERSISTENT	0x0b
UPDATE_PERSISTENT	0x0c
SELECT_PERSISTENT	0x0d
ASK_ PERSISTENT	0x0e
CONSTRUCT_ PERSISTENT	0x0f
TERMINATE	0x10
RESET	0x11
CREATE	0x12
DROP	0x13
COPY	0x14
MOVE	0x15
ADD	0x16

In RES and IND messages the *Transaction type* and *Message type* fields are replaced with 7-bit *Status code* and 1-bit *Response type* fields (either RES or IND). The Status code specifies whether operation was successfully executed. Available status codes are illustrated in the table 2.

TABLE II. STATUS CODES

Status	Code
OK	0x00
ERROR: KSP version not supported	0x01
ERROR: Invalid transaction type	0x02
ERROR: Invalid message type	0x03
ERROR: Invalid data field format	0x04
ERROR: Invalid option type	0x05
ERROR: Invalid option format	0x06
ERROR: SIB internal error	0x07

In order to be able to pair RES and IND messages with the requests each KSP transaction is identified with 16-bit Transaction ID. UDP-like transports (no ordered delivery of messages) need also an additional identifier for the IND messages. The 16-bit *Sequence number* field is used for this purpose. The first indication for each persistent transaction must use a value 0x01 and the value is incremented by one for each following indication. A KP may discard an indication as outdated under the following condition:

$$\frac{-2^{16}}{2} < I_2 - I_1 < 0 \quad (1)$$

Where I_1 is the sequence number of the previous (not discarded) indication and the I_2 is the sequence number of the most recent indication. The right side of the equation checks if the sequence number for I_2 is smaller than for I_1. The left side of the equation checks whether the sequence number for the I_2 has wrapped around.

D. Message Format and Semantics: Data field

The KSP transactions can be divided into three categories: query, update, and terminate. In query and update operations the REQ message *Header* field is followed by the *Data* field which contains the transaction specific information. With TERMINATE operation the *Data* field is not needed because transaction identifier in the request header can be used to define the active persistent transaction to be terminated. In RES messages only query operations contain the *Data* field.

1) Encoding Format for RDF graph

Since the KSP is a query and update protocol for RDF the RDF graphs play a central role in almost all KSP messages. The common structure for *Graph* fields is illustrated in the fig. 5.

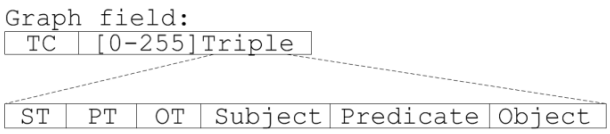

Figure 5. RDF graph field structure

The *Graph* field consists of 8-bit triple count (*TC)* field and a zero or more (maximum 255) *Triple* fields. Each *Triple* field starts with 3-bit *ST*, 2-bit *PT*, and 3-bit *OT* fields, which specify the content of the following *Subject*, *Predicate*, and *Object* fields respectfully. Possible types for these triple members (subject, predicate, and object) are presented in the table 3. The *Literal* type can be only used in objects.

TABLE III. TRIPLE MEMBER TYPES

Type	Code
Empty	0x00
URI	0x01
Reserved Word	0x02
Variable	0x03
Literal	0x04

The field structure for the various triple members is illustrated in the fig. 6. In *URI* field the 8-bit *Prefix index* field specifies the URI from the prefix list that is concatenated with the local URI to form the full URI (see prefix option). For example, with prefix index value 0x03 the third URI is selected form the prefix list. Prefix index value 0x00 is used for full URIs. The 8-16 bit *URI length* field specifies the length of the actual URI string field. The first bit of the *URI length* field denotes whether the length of the URI is presented with one or two bytes. This kind of length encoding allows not only compact messages, but also makes it possible to use longer URIs when necessary. The drawback is that the parser needs to perform extra work when decoding/encoding the messages. Similar length encoding is also used in other strings in the KSP.

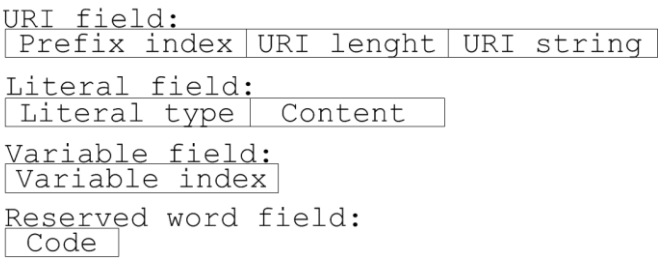

Figure 6. Field structure for triple member types

Literals in RDF can be either plain or typed. Only typed Literals are used in KSP however. The first eight bits in the Literal field is reserved for the type. The table 4 presents the supported Literal types in the KSP version 1.0.

TABLE IV. LITERAL TYPES

Type	Value
xsd:string	0x00
xsd:interger	0x01
xsd:float	0x02
xsd:dateTime	0x03
xsd:Boolean	0x04

With *xsd:string* type literal the first 8-32 bits is reserved for the length of the string. The two most significant bits specify the number of bytes used for the length field and following 6-30 specify the length of the *xsd:string*. The *xsd:integer* and *xsd:float* fields are 32-bit long. The IEEE 32-bit floating-point format is used for the *xsd:float* type. The *xsd:dateTime* field is 19 byte ASCII string. The *xsd:Boolean* field contains a 8-bit unsigned integer. Value 0x00 is reserved for "false" and 0x01 for "true".

Variable type triple member field contains 8-bit variable index which is used as an identifier for the variable in the KSP message. With *n* variables the largest variable index is *n-1*. In SELECT operation it is also required that the indexes for projected variables (i.e. variables whose bindings are returned) start from zero. For example, in SELECT query with six variables of which two are projected variables the variable indexes 0x00 and 0x01 are used for projected variables and indexes from 0x02 to 0x05 are reserved to other variables.

The target in KSP is to provide as compact messages as possible and therefore two triple member types are designed just for this purpose. With *Empty* type the field (subject, predicate, or object) is not present and the corresponding value from the previous triple is used. The purpose of the *Empty* type is to enable more compact messages to be constructed by grouping triples with common subjects, predicates, and objects. The basic idea is similar to the Predicate-Object and Object lists in SPARQL, but the *Empty* type is more versatile because it allows both predicates without common subject, and objects without common subject-predicate to be grouped. The *Reserved word* type is another mechanism for shortening KSP messages. The idea in reserved words is to present common vocabulary with eight bit unsigned integers. The code values

for various reserved RDF, XML Schema (XMLS), RDFS, and OWL words are presented in the table 5.

TABLE V. RESERVED WORDS

Word	Value
rdf:type	0x00
rdfs:Class	0x01
rdfs:subClassOf	0x02
rdfs:property	0x03
rdfs:subPropertyOf	0x04
rdfs:range	0x05
rdfs:domain	0x06
owl:TransitiveProperty	0x07
owl:SameAs	0x08
xsd:string	0x09
xsd:interger	0x0a
xsd:float	0x0b
xsd:dateTime	0x0c
xsd:Boolean	0x0d

2) Data field format for query operations

KSP defines six types of query operations. These operations include the transient and persistent formats for SELECT, ASK and CONSTRUCT. The persistent query operations in KSP are very similar to the SSAP equivalents. The only difference is that in KSP the SIB does not inform the KP about new and obsolete results, but just sends the current status of the query when the results have changed. The *Data* field format for transient and persistent query REQ and RES messages is presented in the fig. 7.

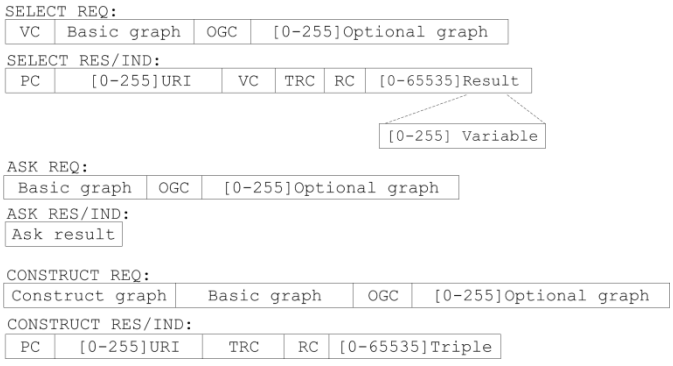

Figure 7. Data field format for query requests and responses

In SELECT requests the 8-bit VC field specifies the number of variables whose bindings are returned in the RES message. The following three fields are reserved for the query pattern which is matched against the RDF-database of the SIB. The query pattern consists of a Basic graph, and a number of Optional graph fields. The Basic graph field defines a triple pattern that must match in order for there to be a solution. The Optional graph fields contain graphs that do not reject the solution if no match for the graph pattern is found. The optional graphs are useful when a KP wants to receive some results even though not all the requested triples exist in the SIB. The 8-bit OGC field defines the number of optional graphs in the query pattern. The ASK request is otherwise identical to the SELECT request except there is no VC field. The CONSTRUCT request is also similar to the SELECT request expect the VC field is replaced by a Construct graph field. The Construct graph field defines triples to be constructed by replacing the variables with RDF terms obtained from the query pattern matching.

As already mentioned the query RES and IND messages are identical in KSP. This simplifies parser and SIB implementations when compared to the SSAP where the indications contain both new and obsolete results. The SELECT RES/IND messages start with 8-bit *PC* field which specifies the number of following URIs associated to the prefix indexes. The 16-bit *TRC* and *RC* fields specify the number of total results and results respectfully. The difference between these fields is that the total result count specifies the total number of results of the query operation whereas the result count defines the number of results in the RES message. These values can be different if all the results do not fit to a single message due the max response size. Each *Result* field in SELECT RES/IND message contains a number of bound variables. The exact number of variables (maximum 255) in a single *Result* field is defined by the 8-bit *VC* field. The first eight bits in the *Variable* field specify the type of the RDF term to which the variable is bound. Possible types for bound variables include *Empty*, *URI*, *Reserved Word*, and *Literal*. Same formats for these RDF terms fields are used as in the *Triple* field (see fig. 6). The CONSTRUCT RES/IND messages are otherwise similar to the SELECT messages expect there is no *VC* field and the *Results* field contains triples instead of RDF terms. The *Data* field in ASK RES/IND messages contains 8-bit unsigned integer defining whether solutions were found.

3) Data field format for update operations

In addition to the query operations the KPS provides various operations for manipulating the data in the SIB. These operations can be roughly divided into two groups: update and management. The difference between these operation types is that update operations are used to modify triples inside the graphs whereas the management operations provide mechanisms for creating, destroying, *etc.* complete graphs. Fig. 8 illustrates the data field formats for the data manipulation requests.

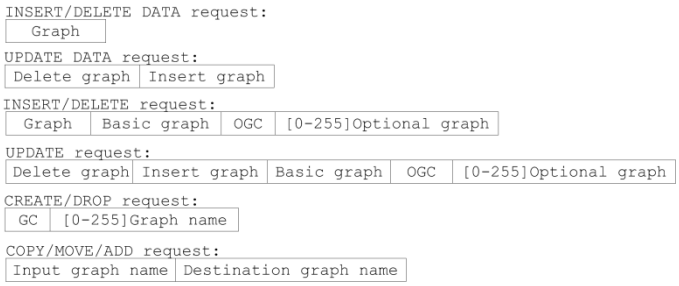

Figure 8. Data field structure for update request

The DATA format KSP update requests consist of various graphs which depending on the operation define either the triples to be deleted or/and inserted. The difference between plain and DATA type operations is that variables are not allowed in DATA operations. The advantage of DATA type operations is that large updates can be done without the need to first query bindings to the variables. The plain update

operations provide also many advantages over the DATA type operations, however, and it depends on the situation which type should be used. The plain update operations are especially useful in situations where a KP would need to first query information and then use the information for modifying triples in the SIB. It is also possible to create simple rules to the SIB by defining the update to be persistent. In persistent update operation a new query pattern match is executed always when information in the SIB is modified. If solutions are obtained from the query pattern matching the data manipulation part of the operation is executed.

With SSAP it is only possible to access a single graph in the SIB. Sometimes it is feasible to be able to create separate graphs for different purposes however (e.g. privacy management). A simple way to manage privacy in M3 is to create separate graphs for various user groups and make the graphs accessible only to specific users. In KSP the CREATE and DROP operations are used for creating and destroying graphs respectfully. The 8-bit *GC* field defines the number of following *Graph name* fields which specify the URIs for the graphs to be created or destroyed. The encoding of the *Graph name* field is identical to the *URI* field. In COPY, MOVE and ADD operations the *Source graph* and *Destination graph* fields specify the source and destination graphs of the operation respectfully. Same encoding is used as for the *Graph name* field.

E. Message Format and Semantics: Options field

One of the main advantages of XML based protocols is extendibility. In KSP options are a way to achieve a certain level of extendibility in a non-XML protocol. Another advantage of options is that because options are, as the name implies, optional they make it possible to create more compact messages by leaving out the parts that are not needed in the particular message. The *Options* field follows the *Data* field in the KSP REQ messages. The 8-bit *OC* field defines the number of options in the request. Eight bits are reserved for the type in each option field. The table 6 presents the code values for the various option types.

TABLE VI. OPTION TYPES

Option	Code
PREFIX	0x00
DELETE GRAPH	0x01
INSERT GRAPH	0x02
QUERY GRAPH	0x03
OPTIONAL PATTERN	0x04
FILTER	0x05
SOLUTION MODIFIER	0x06
BIND	0x07
MAX RESPONSE SIZE	0x08
CREDENTIALS	0x09

The fig. 9 illustrates the field formats for the various KSP options. One of the most important design guidelines for the KSP was to support low capacity computing platforms. There are two options specified for this purpose: *Prefix* and *Max response size*. The *Prefix* option works as the PREFIX keyword in SPARQL and it is useful for shortening URIs appearing multiple times in a message. In *Prefix* field the first eight bits specify the number of following *URI* fields. In KSP messages the order number (prefix index) of the URI is used in the same way as the prefix label in SPARQL. By allowing the maximum size for RES and IND messages to be specified in the REQ message, the 32-bit *Max response size* option field makes it easier for a KP to estimate the memory requirements at compile time.

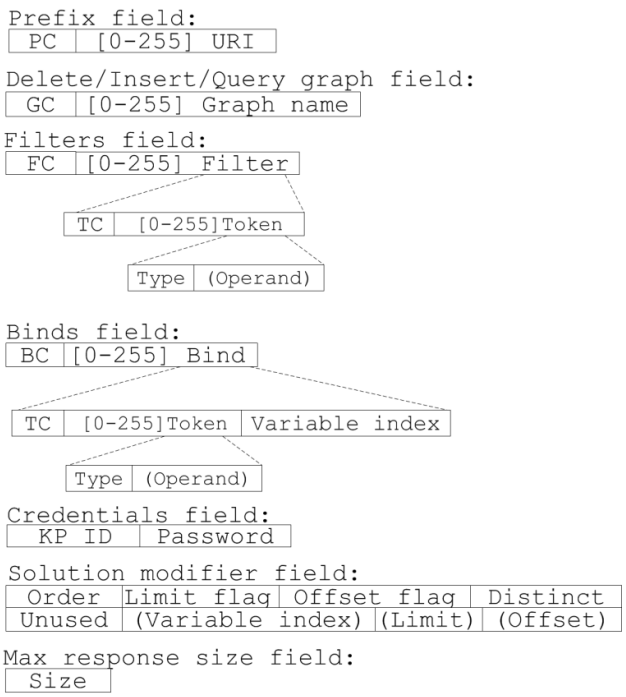

Figure 9. Field structures for options

Normally the query and update operations affect to all the graphs in the SIB (assuming KP has access rights). It is sometimes useful to make a KSP operation to affect only certain graphs in the SIB however. The *Delete graph*, *Insert graph* and *Query graph* options are used for this purpose. The first eight bits specify the number of named graphs. Same encoding is used for *Graph name* fields as for normal URI fields. The various named graph options are typically used together with credentials option which provides access control for KSP communication. Credentials are needed in situations where the SIB (or some graphs inside a SIB) can be accessed only by certain KPs. The *Credentials* field consists of 128-bit *KP ID* and *Password* fields which identify each KP uniquely. The first 8-bits in the *Password* field define the length of the following password. The *Credentials* option is typically only useful with transports such as Transport Layer Security (TLS) and Datagram TLS (DTLS) that provide encryption and trust for the authentication process. It should be noted that the KSP provides only a way to present the access rights and the actual access control management is out of the scope of the paper.

The KSP provides two options for modifying the solution produced in query pattern matching: *Filter* and *Solution modifier*. The *Filter* option can be used for limiting the solution to those which filter expression evaluates true. The first eight bits in the *Filters* field are reserved for filter count (FC). Each filter expression consists of a group of operand and

operator tokens presented in Reverse Polish notation (RPN). The RPN was chosen because it provides a compact notation and allows fast processing of the expressions. The 8-bit *TC* field defines the number of tokens in the expression. The type for each token is also presented with eight bits. Values from 0x00 to 0x06 are reserved for operands and values from 0x07 onwards for operators. The table 7 presents the code values for various token types. If the token is operand type the *Type* field is followed by the *Operand* field which specifies the value for the operand.

TABLE VII. TOKEN TYPES

Token type	Code
xsd:string	0x00
xsd:integer	0x01
xsd:float	0x02
xsd:dateTime	0x03
xsd:Boolean	0x04
variable	0x05
URI	0x06
=	0x07
!=	0x08
<	0x09
>	0x0a
<=	0x0b
>=	0x0c
+	0x0d
-	0x0e
*	0x0f
/	0x10
||	0x11
&&	0x12
regex	0x13

The *Solution modifier* option provides ways for altering the solution sequence. By default the solution sequence of query operations is unordered. The 2-bit *Order* field can be used to define the solution order (unordered, ascending, or descending). The 1-bit *Limit flag*, *Offset flag*, and *Distinct flag* fields define whether the *Limit*, *Offset*, and *Distinct* modifiers are used. These modifiers are identical to the SPARQL equivalents. The next 3-bits are unused in this version of the KSP. If the solution sequence is ordered the 8-bit *Variable index* field defines the variable based on which the order of the solution sequence is created. For example, alphabetic order for the solution sequence is used if the variable is bound to xsd:string type. If the *Limit* and *Offset* flags are set the 16-bit *Limit* and *Offset* fields specify the maximum number and the offset for the results respectfully. The *Limit* and *Offset* modifiers are especially useful in large queries because they allow the KP to request the results in smaller packets.

The last option type in KSP is the *Bind*. The *Bind* option allows new values for variables to be assigned. Unlike the BIND keyword of SPARQL (can be used inside the query pattern) the *Bind* option is always executed after the query pattern matching. The *Bind* option is especially useful in update operations. Certain scenarios are even quite difficult to execute without the bind. For example, let's consider a scenario where the KP needs to increment a certain literal by one. Without the *Bind* option the KP would first need to query the value, update it by one, and then insert the new value to the SIB. However, if another KP modifies the literal between the query and the update operations, the update operation does not work correctly. The structure of the *Bind* field is similar to the *Filter* field. The only difference is the 8-bit *Variable index* field specifying the variable to be bound.

IV. VALIDATION

In order to evaluate the KSP in practice we compared it with the other M3 communication protocols in a prototype smart space called Smart Greenhouse [22]. Of all the KPs in the Smart Greenhouse we chose to implement the Autocontrol KP because it is the most complex embedded system in that smart space. First, the Autocontrol KP was implemented with SSAP/XML and SSAP/WAX using as few and as compact messages as possible. In order to reduce the amount of messages the SPARQL queries were used instead of the Triple queries. We used both the Predicate-Object and Object list, as well as, minimum number of characters in variables to make the SPARQL queries as compact as possible. Then, to make direct comparison of the M3CPs possible we implemented the Autocontrol KP with KSP using the same operations used with SSAP. Finally, to demonstrate the full capabilities of the KSP we implemented the Autocontrol KP using persistent update operations.

In Smart Greenhouse the Autocontrol KP is responsible for modifying the status of the virtual actuators (LEDs, fans, and a water pump) available in the SIB. To do this it utilizes information about plant preferences and sensor measurements for humidity, temperature, and luminosity. For example, when the temperature is too high for a given plant the Autocontrol KP publishes information stating that the fans should be turned on. The information published by the Autocontrol KP is used by Actuator KP which modifies the physical actuators accordingly.

The operations needed for modifying the status for LEDs, fans, or the water pump when the luminosity, temperature, or humidity are out of the range of plant preferences are very similar. First, the Autocontrol KP needs to query the available actuators. Then to be aware when the state of an actuator needs to be modified the KP executes two ASK subscriptions. The first subscription informs when an actuator needs to be turned on and the second when an actuator needs to be turned off. The content of the subscriptions depends on the actuator type. After performing the ASK subscriptions the Autocontrol KP waits for IND messages from the SIB. Every time the SIB notifies the Autocontrol KP that the status of an actuator needs to be modified the KP updates a new status using the SSAP update operation. Again the content of the message depends on the actuator type.

The fig. 10 illustrates the message exchange between Autocontrol KP, Sensor KP, Actuator KP and the SIB. To make the sequence diagram as clear as possible we simplified it in two ways. First, only the messages related to the temperature and fans are illustrated and it should be noted that in reality similar message exchange is executed for other measurements and actuators as well. Second, only one of the two ASK subscriptions is illustrated in the sequence chart. It is also assumed that the static plant preference values for minimum (19.5 Celsius degree) and maximum (25.7 Celsius degree) temperature have been published into the SIB. The

Autocontrol KP uses CON type request while NON type request are used by the Sensor KP and the Actuator KP.

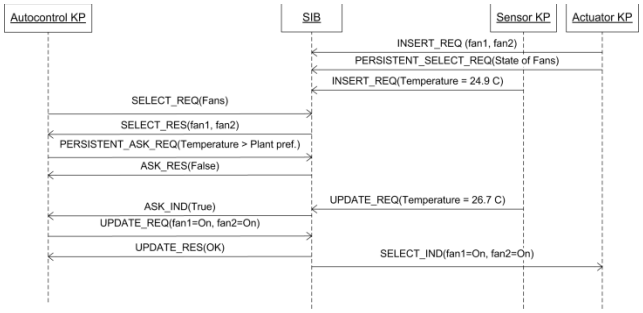

Figure 10. Sequence chart illustrating the message exchange between the KPs and the SIB in Smart Greenhouse

The average size of REQ messages sent by the Autocontrol KP is illustrated in the fig. 11. The sizes for RES and IND messages are presented in the fig. 12. TCP was used as the transport technology with all M3CPs. For clarity sake we did not include the *join* and *leave* messages needed with SSAP/XML and SSAP/WAX. The first column in the fig. 11 presents the average size of messages used to query the different actuators (fans, LEDs, or a water pump) from the SIB. In the fig. 12 the first column presents the average size of the RES messages to the actuator query. The second and third columns in fig. 11 and fig. 12 represent the ASK subscription requests and indications respectfully. In fig. 11 the fourth column presents the average size for the update message that is send every time a subscribe indication is received from the SIB. The average size for the update response message is illustrated in the fourth column of the fig. 12. As can be seen from the fig. 11, the size for KSP query/subscribe request is on average only 18.63% of the corresponding SSAP/XML and 43.58% of the corresponding SSAP/WAX requests. In the case of the update request the average size of KSP message size is 15.22% of SSAP/XML and 27.50% of the SSAP/WAX requests. In the RES and IND messages the difference between KSP and SSAP is even bigger. The KSP response/indication message size is on average 6.89% of the SSAP/XML and 19.06% of the SSAP/WAX RES/IND messages.

In the previous study we illustrated how implementing the Autocontrol KP with KSP leads to a significantly shorter message sizes than with other M3CPs even when the same operations are used. The KSP provides also more advanced ways to implement the Autocontrol KP however. By using the persistent update operation both the workload on the Autocontrol KP and the network traffic can be dramatically decreased. Only two persistent update operations are needed for each actuator type and once the Autocontrol KP has performed these operations it can enter to a sleep mode. The persistent update requests are the only messages the Autocontrol KP has to send and it is therefore practical to compare them with the largest messages needed with other M3CPs. The average size of the persistent update requests sent by the Autocontrol KP is 165 bytes which is only 14.27% and 25.78% of the largest SSAP/XML and SSAP/WAX requests respectfully.

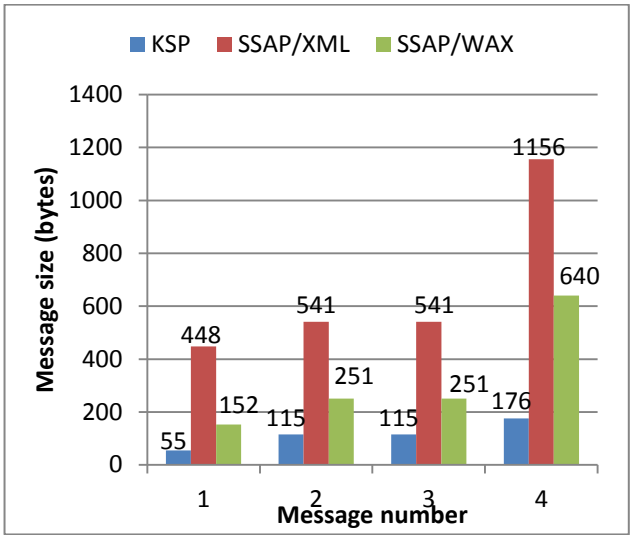

Figure 11. Average request size of Autocontrol KP with different M3 communication protocols

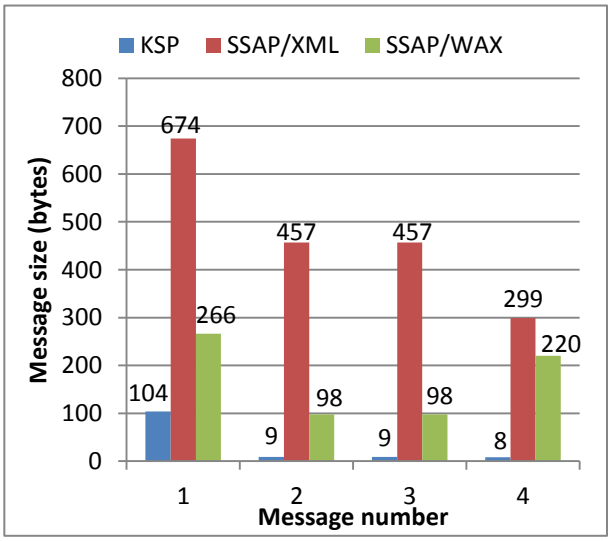

Figure 12. Average message size of the response and indication messages received by the Autocontrol KP with different M3 communication protocols

When CON type requests are used the size for each persistent update RESP message is four bytes (UDP assumed) which is only 0.59% of the largest SSAP/XML and 1.50% of the largest SSAP/WAX RES message. When used with TCP it is feasible to use NON type request in which case the persistent update response messages are not needed at all. The fig. 13 presents the message exchange between the KPs and the SIB in Smart Greenhouse when persistent update operations are used by the Autocontrol KP.

V. Conclusions and Future Work

We presented a novel knowledge sharing protocol for semantic technology empowered AmI systems. The KSP is designed for M3 applications but it can be also used with any other application that requires a compact protocol for knowledge sharing.

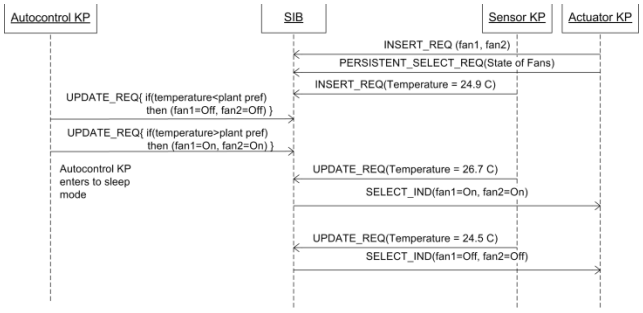

Figure 13. The message exchange between the KPs and the SIB when persistent update operations are used by the Autocontrol KP.

The main design guideline in the KSP was to implement SPARQL-like knowledge sharing mechanism in a compact binary format that is suitable for real-life smart spaces. In addition to the compact binary format, the feasibility of the KSP to resource restricted devices and networks is improved with mechanisms such as the persistent update, multi-transport support, and the max request size option, for example.

For evaluation purposes we implemented the Autocontrol KP of the Smart Greenhouse with different M3CPs. The KSP messages were on average 87.08% and 70.09% shorter than the SSAP/XML and SSAP/WAX messages respectfully. We also demonstrated how the Autocontrol KP implementation can be significantly simplified with persistent update operations. Only six (two for each actuator type) persistent update request were needed to fully implement the Autocontrol KP. It is evident that with fewer and more compact messages the KSP is more suitable for battery powered low capacity devices than the other M3CPs.

The KSP is designed to be compact and easily processable knowledge sharing protocol for ubicomb systems. This kind of format does not come without limitations however. The binary format limits both maximum amount and size of entities such as prefixes, graphs, triples, and results, for example. This may cause troubles in certain situations where huge amount of RDF triples need to be manipulated in a single operation. Another drawback in KSP is that unlike SPARQL it requires a good application programming interface (API) because the binary format is not suitable to be used by developers as such. We also choice not to implement some of the rarely used features of SPARQL 1.1 mainly because they would have made the KSP too complicated. The current version of the KSP does not support SPARQL 1.1 features such as DESCRIBE queries, Property paths, Aggregates and Subqueries, for example.

The adoption of semantic technologies in resource restricted devices is not only important for ubiquitous computing, but also for the Semantic Web. This is because, the Semantic Web is not going to get wider acceptance before there is enough data available to create meaningful Semantic Web applications. Therefore, in the future we are planning to exploit the KSP also in the field of IoT and Semantic Web. To this end we will further develop and evaluate the KSP. For example, new bindings for at least the BLE transport needs to be implemented. We are also considering whether some of the missing SPARQL 1.1 functionalities should be incorporated into the next version of the KSP.

In the future we will also need to make a more comprehensive study on the benefits and drawbacks of the KSP when compared to SPARQL/HTTP used in the Semantic Web. Because the KSP is a binary format with predefined places for parameters it is obviously much faster to parse than the SPARQL. However, the actual difference in parsing times needs to be measured with different workloads to justify the drawbacks caused by the binary format. The effect of persistent update operations on the performance of the SIB needs to also to be carefully analyzed in the future.

ACKNOWLEDGEMENTS

This work has been funded by the Merging IoT Technologies (MIoTe), the Device and Interoperable Ecosystem (DIEM), and Smart Objects for Intelligent Applications (SOFIA) projects.

REFERENCES

[1] M. Weiser, “The Computer for the 21st Century,” Scientific American, September 1991, pp. 94-100.

[2] E. Aarts, H. Harwing, and M. Schuurmans, “Ambient Intelligence,” The Invisible Future: The Seamless Integration Of Technology Into Everyday Life. Denning, P. (ed.), McGraw Hill, New York (2001).

[3] T. Berners-Lee, J. Hendler, and O. Lassila, “The Semantic Web,” Scientific American, May 17, 2001, pp. 34-43.

[4] O. Lassila, “Serendipitous interoperability”, The Semantic Web Kick-off in Finland – Vision, Technologies, Research, and Applications, HIIT Publications, University of Helsinki, 2002.

[5] H. Chen, An Intelligent Broker Architecture for Pervasive and Context-Aware Systems, doctoral dissertation, University of Maryland, Baltimore County, Department of Computer Science and Electrical Engineering, 2004.

[6] A. Lappeteläinen, J. Tuupola, A. Palin, and T. Eriksson, “Networked systems, services and information – The ultimate digital convergence,” First International NoTA conference, 2008.

[7] Z. Shelby, and C. Bormann, 6LoWPAN: the Wireless Embedded Internet, John Wiley and Sons, 2010, p. 244

[8] B. SIG, “Bluetooth specification version 4.0,” Available at http://www.bluetooth.org, August 2012.

[9] G. Klyne and J. J. Carroll. 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, 10 February 2004, URL: http://www.w3.org/TR/rdf-concepts/.

[10] D. Brickley and R.V. Guha. 2004. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation 10 February 2004, URL: http://www.w3.org/TR/rdf-schema/.

[11] W3C OWL Working Group. 2009. OWL 2 Web Ontology Language Document Overview. W3C Recommendation, 27 October 2009, URL: http://www.w3.org/TR/owl2-overview/.

[12] S. Harris and A. Seaborne. 2012. SPARQL 1.1 Query Language, W3C Working Draft, 5 January 2012, URL: http://www.w3.org/TR/sparql11-query/.

[13] J. Schneider and T. Kamiya, 2001. Efficient XML interchange (EXI) format 1.0. W3C Recommendation, 10 March 2001, URL: http://www.w3.org/TR/exi/

[14] International Telecommunication Union: X.694 (2004), http://www.itu.int/ITU/studygroups/com17/languages/x694.pdf

[15] X. Su, J. Riekki, and J. Haverinen, ”Entity Notation: enabling knowledge representations for resource-constrained sensors”, Personal and Ubiquitous Computing, 21 September, 2011, pp. 1-16

[16] B. DuCharme, Learning SPARQL: Querying and Updating with SPARQL 1.1, O’Reilly Media (2011)

[17] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, Pattern-Oriented Software Architecture: A System Of Patterns, West Sussex, England: John Wiley & Sons Ltd., 1996

[18] J. Honkola, H. Laine, R. Brown, and O. Tyrkkiö, ”Smart-M3 information sharing platform,” proc. ISCC 2010, pp. 1041 – 1046

[19] A. Ylisaukko-oja, P. Hyttinen, J. Kiljander, J. Soininen, and E. Viljamaa, ”Semantic Interface for Resource Restricted Wireless Sensors,” IC3K 2nd International Workshop on Semantic Sensor Web – SSW2011, October, 2011, Paris, France.

[20] J. Suomalainen, P. Hyttinen, and P. Tarvainen, "Secure information sharing between heterogeneous embedded devices," proc. ECSA 2010, pp. 205-212

[21] Z. Shelby, K. Hartke, C Bormann, and B. Frank. B. 2012. Constrained Application Protocol (CoAP). IETF Internet-Draft 09, URL: http://datatracker.ietf.org/doc/draft-ietf-core-coap/ (2012)

[22] J. Kiljander, M. Eteläperä, J. Takalo-Mattila, and J. Soininen, “Opening information of low capacity embedded systems for Smart Spaces”, Proc. WISES 2010, pp. 23-28

AUTHORS PROFILE

Jussi Kiljander is a research scientist at VTT Technical Research Centre of Finland. He received his M.Sc. (Technology) from University of Oulu in 2010. His current research and Ph.D. studies are focused on ubiquitous computing and device interoperability with semantic technologies. He has published more than 10 scientific papers and contributed to several projects related to semantic interoperability and pervasive computing.

Francesco Morandi was born in Lugo, Italy, the 26th of July 1984. He received the degree in Electronical Engineering in 2006 and specialistic degree in Telecommunication Engineering in 2009 both from University of Bologna. In 2010 he worked for Fondazione Ugo Bordoni in Pontecchio Marconi (Bologna) as consultant for the television transition to digital (DVB-T). From 2011 he's working as researcher for the University of Bologna in ARCES (Advanced Research Center on Electronic Systems for Information and Communication Technologies E. De Castro). His current research is focused on developing and improving interoperable platforms for healthcare, maintenance and energy smart grids.

Prof. Juha-Pekka Soininen is a research professor of computing and computer architectures at VTT Technical Research Centre of Finland. He received his MSc, LicTech and Doctor of Science (Technology) degrees from University of Oulu 1987, 1997 and 2004 respectively. He has been Research scientist at VTT since 1988, senior research scientist since 1996 and research professor since 2007. He has been the leading expert in various large research projects at VTT during 1993 - 2011. These projects include contract research projects, joint research projects and European Union research projects. His current research deals with ubiquitous and distributed computing, system architectures, platform-based design methodologies, system architecture evaluation methods, and system-level design methods. He has been a reviewer in several international conferences, journals and books. His has published more than 70 scientific publications and he is a member of IEEE.