Subscription Operation in Smart-M3

Aleksandr Lomov, Dmitry Korzun
Petrozavodsk State University
Petrozavodsk, Russia
{lomov, dkorzun}@cs.karelia.ru

Abstract

Smart spaces provide a shared view of dynamic resources and context-aware services within a distributed application. Smart-M3 is an open-source platform that implements the smart space concept. It inherits the tuplespace and publish/subscribe asynchronous communication models, and agents (Smart-M3 knowledge processors, KPs) cooperate sharing the common smart space. A Smart-M3 semantic information broker (SIB) maintains the smart space in low-level ontological terms of triples, based on resource description framework (RDF). Each SIB supports the subscription operation; it allows storing data in a synchronized state at several KPs or to notify KPs about changes. In this paper we consider the Smart-M3 subscription operation. First, we study its low-level version with RDF triples as parameters. This version is implemented in KPI_low interface; other low-level KP interfaces employ similar schemes. Second, we introduce our high-level version of Smart-M3 subscription operation, which operates with such ontological concepts as class, property, and individual. This version is implemented in SmartSlog ontology library.

Index Terms: Smart spaces, Smart-M3, Subscription, SmartSlog, RDF, OWL.

I. Introduction

Smart spaces provide an environment for heterogeneous devices and programmable agents to share their resources and services [1], [2]. An information subsystem provides permanent robust infrastructure for storing and retrieving the information of different types from the multitude of environment participants. Smart-M3 is an open-source interoperability platform for information sharing [3]. Smart-M3 implements information subsystems of smart environments [4], [5]. A semantic information broker (SIB) maintains its smart space information store and serves incoming queries from the application side.

A Smart-M3 application consists of knowledge processors (KPs)—distributed agents running on various computers and participating in the smart spaces. A smart space stores information in a shared RDF triple store: a collection of triples, where a triple is *object–predecate–subject*. A participating KP understands a non-exclusive set of triples and assumes that they form an RDF graph. The latter usually allows a higher-level ontological data representation model such as OWL, i.e., a graph of individuals (instances of ontology classes), each individual has data properties and individuals are interlinked with object properties.

Each KP connects to a SIB using the Smart Space Access Protocol (SSAP). The developers of the KP logic use a knowledge processor interface (KPI) to smart spaces. Supported operations are insert, remove and update queries, and subscribe and unsubscribe. A subscription operation is a special case of query—a persistent query [3], [6]. It realizes the publish/subscribe communication model [7] in smart spaces. Consequently, a change in the space triggers actions from participating KPs.

This paper focuses on the Smart-M3 subscription operation. A KP subscribes to information in the space for synchronizing its local knowledge with the shared space or for receiving notifications about recent changes.

We analyze the basic Smart-M3 subscription operation, where RDF triples are knowledge units. For a reference case we consider KPI_low[1], which directly relays the SSAP operations. This low-level type of subscription does not account the RDF graph to track relations between triples. As a result, KP logic code has to deal with a lot of triples that each subscription returns and the code has to check itself their RDF graph consistency, e.g., in terms of the *rdf:type* predicate.

To overcome these drawbacks we propose a high-level subscription scheme that operates in terms of such ontological entities as individuals, their data and object properties. In this scheme, a KP subscribes to certain properties of a specified set of individuals available in the smart space. From the point of view of KP logic, the result is a set of new values (whenever some properties have been updated) or property remove/insert notifications.

We implemented this scheme in SmartSlog ADK[2] [8], [9]. The latter maps an OWL ontology description (provided by KP logic developer) to auxiliary code (ontology library that forms a KP-specific KPI). Thus the developer abstracts the ontology and smart space access in the KP code, benefiting from a less expensive application development process.

SIB and SSAP support only triple-based operations. SmartSlog ontology library transforms internally all high-level ontological data structures to RDF triples and calls a low-level KPI such as KPI_low. The reverse information flow from SIB requires corresponding RDF-to-OWL transformations. We develop algorithms that reduce our subscription scheme to the basic Smart-M3 subscription operation.

The rest of the paper is organized as follows. Section II analyzes the basic Smart-M3 triple-based subscription operation, where KPI_low is used for the reference case. Section III describes our advanced subscription scheme and provides the algorithms of its implementation in SmartSlog. Finally, Section IV summarizes the paper.

II. The basic Smart-M3 subscription operation

The operation defines a set of triples a KP tracks in the smart space. A KP keeps its triples locally up to date and receives notifications about changes in the specified content in the smart space. The specification uses triple-patterns. A triple-pattern is a triple where some elements are masks (any value). As a result, a triple-pattern maps to a list of actual triples stored in the smart space. A list of triple-patterns is an argument of the subscription operation. Examples of triple-patterns and their mapping to triple store content are shown in Figure 1.

From the point of view of the KP logic the subscription scheme consists of the following steps. The logic sets in the code a list of triple-patterns that describes a required part of the smart space content. Then the KP calls the subscribe function with the list as an argument. A network connection between the KP and SIB is established (e.g., a TCP connection) for the subscription session.

The SIB responses with all triples available in the smart space and matched to a triple-pattern in the given list. Whenever a change has happened in the specified content of the smart space KP's network socket receives the notification data. They include two lists of all the triples that have been affected; one list with old values and the other with new values. Figure 2 shows an example of the data flow in a subscription session.

Updates to the space are from participating KPs. Note that the same KP may subscribe to triples and modify them itself in parallel; the SIB response is the same as in the case of another KP. To close its subscription session, the KP calls the unsubscribe function.

[1]Open-source code is at http://sourceforge.net/projects/kpilow/.

[2]Open-source code is at http://sourceforge.net/projects/smartslog/.

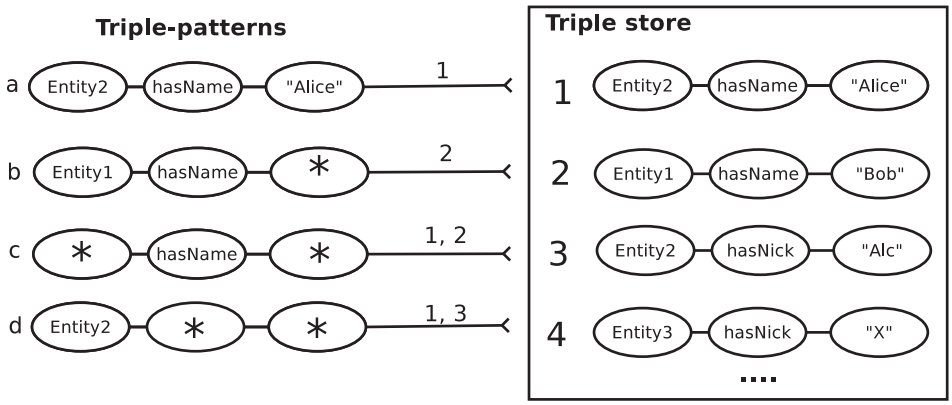

Fig. 1. Example of triple-patterns a, b, c, d and their mappings to the triple set $\{1, 2, 3, 4\}$ in the triple store at SIB. Triple-pattern a maps to triples $\{1\}$; b maps to $\{2\}$; c maps to $\{1, 2\}$; d maps to $\{1, 3\}$.

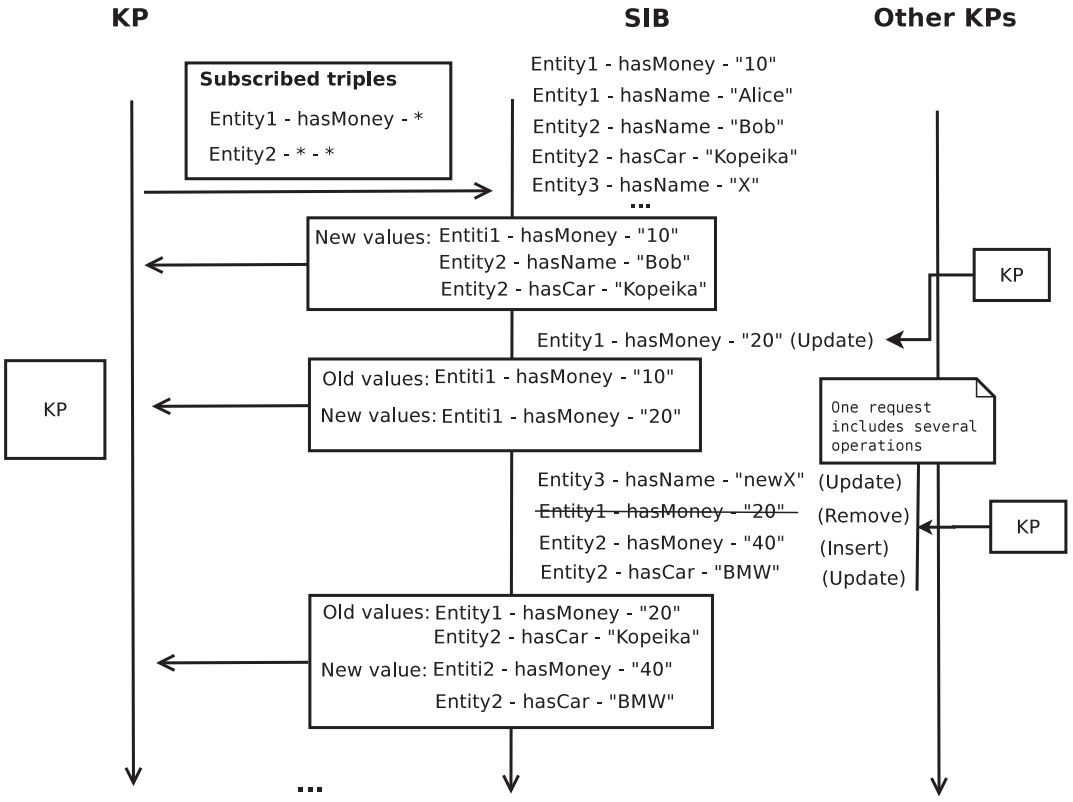

Fig. 2. Subscription notifications for triples: KP (on the left) subscribes to triples at SIB (in the middle) and waits for notifications. Other KPs (on the right) change data in the space and SIB sends the notifications.

There are three actions: insertion, update, and deletion of a triple or a set of triples. Notification data from SIB are two lists: with old and new values of the affected triples, respectively. A KP has to analyze these lists to determine which triples have been affected with which action, see Table I.

This basic subscription scheme interprets triples in the lists independently each on another. In terms of RDF graph, a set of triples can represent an ontological object, for example an instance of *man*, who has a name, age, friends, etc. To represent an object an additional triple with the predicate *rdf:type* is used to define the object type.

TABLE I
REFLECTION OF ACTIONS FROM PARTICIPATING KPS IN THE SUBSCRIPTION NOTIFICATION

Action	Criterion	Example from Figure 2
Update	The triple appears in both lists.	Old values: Entity1 - hasMoney - ”10” New values: Entity1 - hasMoney - ”20”
Insertion	The triple appears only in the list with new values	Old values: no New values: Entity2 - hasMoney - ”40”
Deletion	The triple appears only in the list with old values	Old values: Entity1 - hasMoney - ”20” New values: no

Consider an example.

```
id1 – rdf:type – man
```

The object identifier is used to assign other Sergey’s properties,

```
id1 – name – "Sergey", id1 – age – "23", ...
```

Another object can be added to the space, e.g., Anna, who is a woman,

```
id2 – rdf:type – woman, id2 – name – "Anna", id2 – age – "22", ...
```

Assume that a certain KP subscribes for Anna’s and Sergey’s age using the triple-patterns

```
id1 – age – *, id2 – age – *
```

To decide who is the object, a man or a woman, the KP analyzes the RDF graph. The KP has to either query the SIB for the augmented triples after the subscription notification or maintain the triples for types and identifiers locally. In both cases, additional code is required.

Further, objects can be linked to other objects. For example, a man has friends; they are either men or women. Subscription to Sergey’s friends uses the triple-pattern

```
id1 – hasFriend – *
```

The subscription notification returns triples with a new identifier. Based on the identifier further information about the object can be received from the smart space. If the object type is not additionally checked in the KP logic, then one object can be used incorrectly. In Figure 3, one KP sets a triple that incorrectly links to a car object as to a friend.

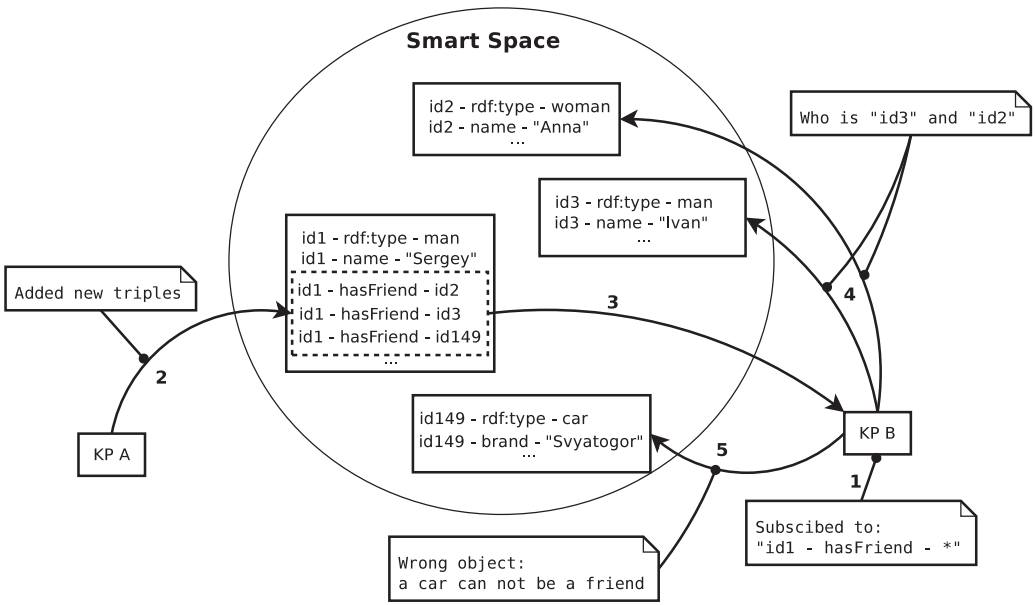

Fig. 3. Subscription to triples for object properties: 1 – KP B subscribes to Sergey’s friends; 2 – KP A add new triples; 3 – KP B receives the added triples; 4 – KP B checks the types; 5 – KP B determines a wrong type of the object.

TABLE II
CODE EXAMPLES OF KPI_LOW SUBSCRIPTION.

Code	Description

`ss_triple_t *triple_rqst = NULL;`
`ss_add_triple(&triple_rqst,`
`"entity", "hasName",`
`SS_RDF_SIB_ANY, SS_RDF_TYPE_URI,`
`SS_RDF_TYPE_URI);`
`ss_add_triple(&triple_rqst,`
`"entity", "hasSurname'',`
`SS_RDF_SIB_ANY, SS_RDF_TYPE_URI,`
`SS_RDF_TYPE_URI);`
	Setting a triple-pattern for data. SS_RDF_SIB_ANY is special constant that means any value.
	SS_RDF_TYPE_URI is type of subject or object.
	This triples are pattern for data:
	entity - hasName - *
	entity - age - *

`ss_triple_t *triple = NULL;`
`ss_subs_info_t subs_info;`
`if(ss_subscribe(ss_info, &subs_info,`
`triple_rqst, &triple) > 0) {`
`printf("Failed to subscribe.");`
`...`
`}`
`ss_delete_triples(triple_rqst);`
	Trying to subscribe. If subscription is successful then delete requested triples and handle a list with triples (triple), it contains all triples that matched to the triple-pattern list (triple_rqst).

`ss_triple_t *n_val = NULL;`
`ss_triple_t *o_val = NULL;`
`int status;`
 	
`while(1) {`
`status =`
`ss_subscribe_indication(ss_info,`
`&subs_info, &n_val, &o_val, 1000);`
`if(status == 1) {`
`print_triples(o\_val);`
`print_triples(n\_val);`
`ss_delete_triples(n_val);`
`ss_delete_triples(o_val);`
`n_val = NULL; o_val = NULL;`
`} ...`
`}`
	Waiting notifications. If notification is received then old and new values are processed and a new iteration begins.
	Notification's data always contains two triples with old and new values of triples that changed in the SIB and matched with the subscription's triple-pattern. If some triples has not changed they are not send to a KP.
	In this example only one status is handled but in real applications it is necessary to handle other statuses that can be returned by the ss_subscribe_indication function.

`ss_unsubscribe(ss_info, &subs_info);`	When subscription is no longer needed it is necessary to unsubscribe from triples.

KPI_low is one of the popular low-level triple-based KPI for Smart-M3 applications. It is oriented to low-performance devices, like embedded devices. KPI_low straightforwardly implements the basic Smart-M3 subscription. It support only a synchronous mode when KP logic blocks waiting for subscription notification form the SIB. A code example for subscription with KPI_low is given in Table II.

Although the triple-based subscription is simple to use it brings unnecessary complexity for the developers. They have to divert effort for managing triples instead of concentrating on the application logic. The next section considers a subscription scheme where KP logic operates with ontological objects, leading to more compact and tractable code.

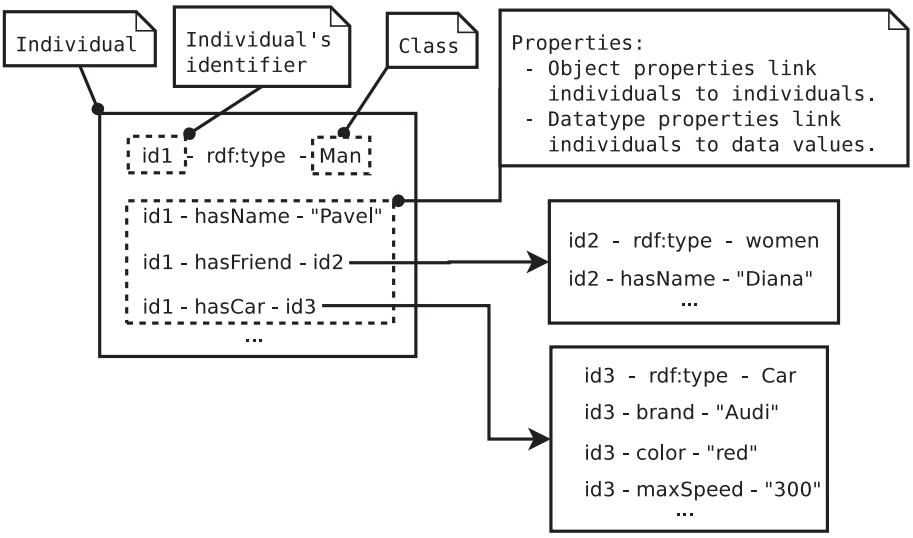

Fig. 4. Individuals and their properties consist of triples, forming a subgraph in the RDF graph.

III. Subscription to ontology properties

In KP development, it is easier to consider knowledge in the smart space as following a high-level ontology model defined in the problem domain. Thus data structures and operations in KP logic are in terms of this ontology.

SmartSlog ADK provides this high-level abstraction mapping an OWL ontology description to code (ontology library). KP code manipulates with ontology classes, individuals (objects as instances of classes), and properties instead of actual underlying triples, although each object corresponds to a subgraph of the RDF graph, see example in Figure 4. In this case, KP code should subscribe to ontological objects, not to RDF triples.

When KP subscribes to a property of an individual, it internally means the basic Smart-M3 subscription to a set triples. Nevertheless, (de)composition (to)from triples is hidden from a KP developer in the ontology library, see Figure 5.

For data properties the SmartSlog subscription operation can be thought as synchronization of local data with recent smart space content. Instead of a list of triples, a subscription container is used. It contains an arbitrary set of individuals; for each individual a set of its properties are included. A container as an argument for the operation leading to a subscription for all the properties within the container. A subscription session is established for a given container. SmartSlog ontology library includes a module (subscriber) that handles the subscription.

Subscription consists of the following steps (see Figure 6).

1. Subscription initialization starts from setting correspondence between individuals and their properties to subscribe for. This correspondence is defined with a subscription container. KP adds into subscription container pairs: *individual — list of properties*.

2. Call subscribe function with the container as an argument. The container can be subscribed synchronously and asynchronously. For each container one subscription session is established regardless how many properties and individuals are in the container.

3. If the container was subscribed successfully, then the first synchronization phase and initialization have been passed. The container is placed into one of the lists, depending on the subscription mode. Now all the properties of the given individuals are tracked for changes.

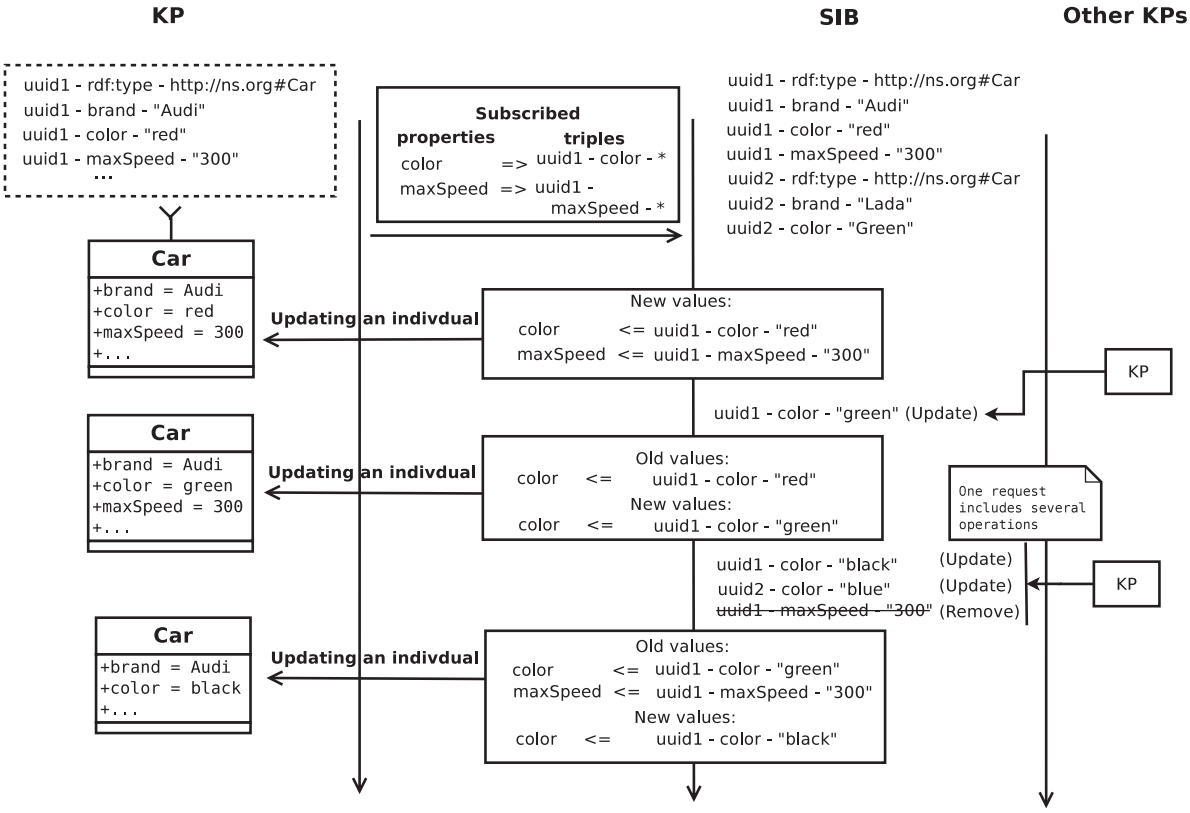

Fig. 5. Subscription notifications for ontological properties.

From this step a subscription session starts.

4. The further process depends on the mode: synchronous or asynchronous.

4-a. In the synchronous mode, the container is expected to be called with a function to check the container. The current thread is blocked and KP waits for data from the smart space. Whenever data are received the function updates the properties in the container and KP continues with new data. This step is iterative, it is repeated depending on the KP logic.

4-b. In the asynchronous mode, the container is updated in background with additional thread. Hence the KP implicitly manipulates with the most recent data from the smart space.

5. Unsubscribe the container, so the subscription session is closed.

Figure 7 shows that whenever a new triple with regular data is added into the smart space then the KP creates locally a new data-property and assigns it to the individual. The case of object-properties needs some more discussion.

Whenever a new triple with reference data is added into the space, then the KP creates locally a new object-property to refer to a new individual. Its identifier is taken from the triple. Importantly that SmartSlog checks a class of the individual and classes which can be set for the property. If the property can not be assigned to an individual of this class then the property is not created.

Synchronous subscription uses a special function that waits for new data, thus blocking the current thread (KP logic). When new data have been received and the container is updated and the function returns the control to the KP logic. Synchronous subscription does not create any other threads, it runs in the thread from which it was called.

Figure 8 shows subscription session of the synchronous subscription. In the initialization

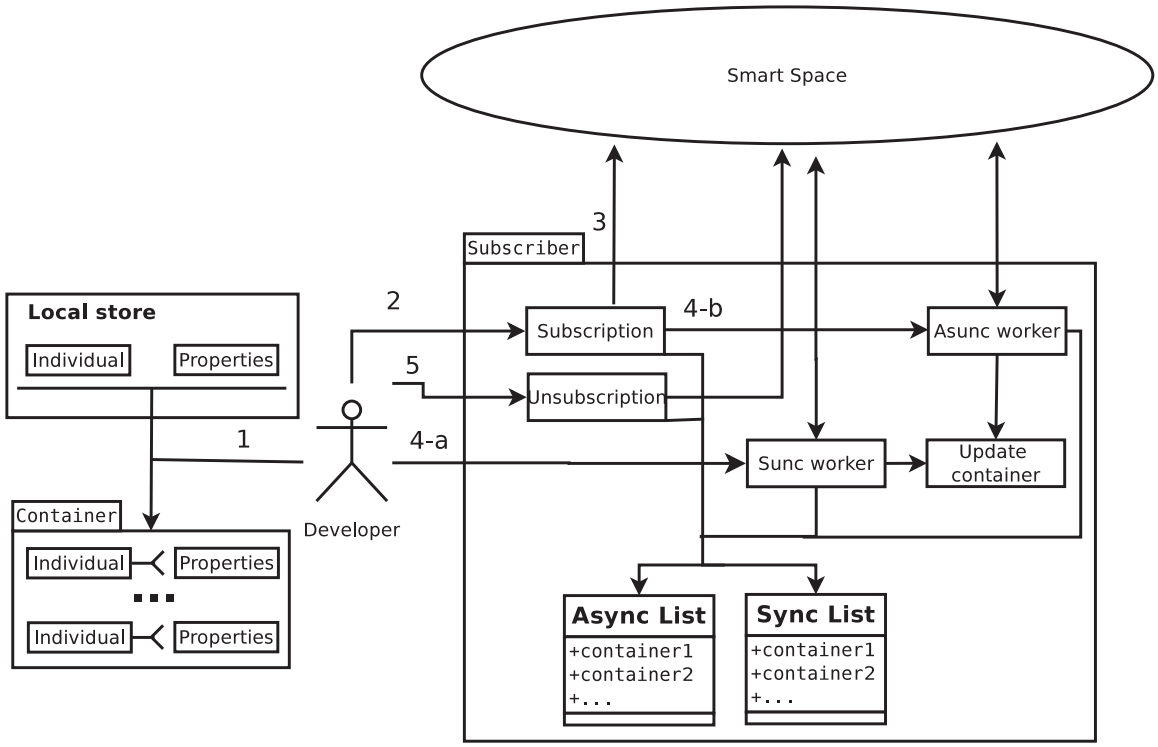

Fig. 6. High-level subscription scheme.

phase, data are subscribed. If the subscription is successful then subscription session starts and a KP can receive notifications from the smart space.

It is possible to get several notifications before a KP have checked them. During the Δ_1 period the smart space sends two notifications, and then the KP checks them all calling a special function to wait a new data (notification), but in this case the KP doesn't wait new data, it handles two packets that have arrived and returns control to a KP logic. The next Δ_2 is the period when nothing has happened. After this the KP again calls the function to get a new data and while it is waiting the KP Logic is blocked (Δ_3). When the KP receives a notification it processes it and returns the control to the KP Logic. At the end of Δ_4 the KP unsubscribes from the data, and the subscription session is closed.

Asynchronous subscription runs in a separate thread, thus the KP logic is not blocked (Figure 9). All containers that were subscribed as asynchronous are updated in the background process. To explicitly track changes callbacks can be used. A callback function is set for a subscription container. When new data have been received and container has been updated then the callback function is called. All changed data are passed to the callback function.

Code examples of SmartSlog subscription are given in Table III. SmartSlog supports ontological libraries in ANSI C and C#, and we provide examples for both.

IV. Conclusion

This paper studied the Smart-M3 subscription operation that implements persistent queries from a KP to its smart space. We analyzed the subscription scheme of low-level Smart-M3 KPIs, which is based on RDF triples, and identified its drawbacks. Then we proposed a high-level subscription scheme that overcomes the triple-based subscription drawbacks. Table IV summarizes the subscription types we consider.

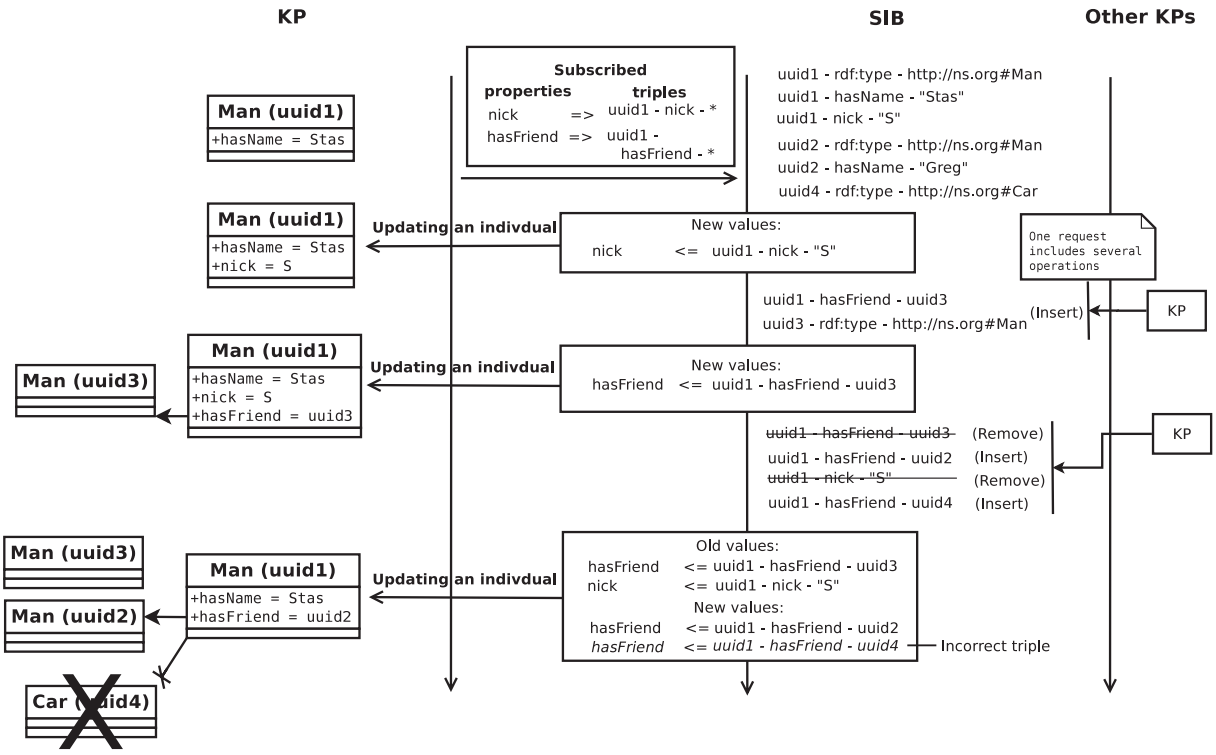

Fig. 7. Data and object properties

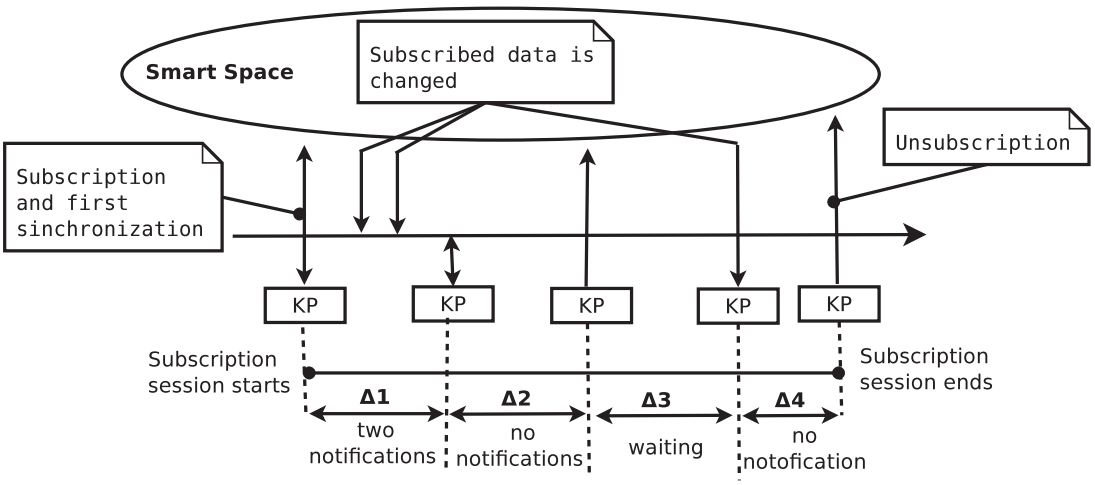

Fig. 8. Time diagram of the synchronous subscription

Our subscription scheme allows synchronous and asynchronous modes; they can be supplied with callbacks. The KP programmer uses an OWL ontology description to represent structurally the smart space content. Instead of a set of RDF triples, KP logic forms a list of individuals and selects a subset of properties, either a data or object one, for every individual from the list. Then KP logic subscribes for all these properties to receive updates from its smart space. Updates come in form of new property values or property remove/insert notifications.

We implemented this scheme in the SmartSlog ADK. The programmer does not need to deal

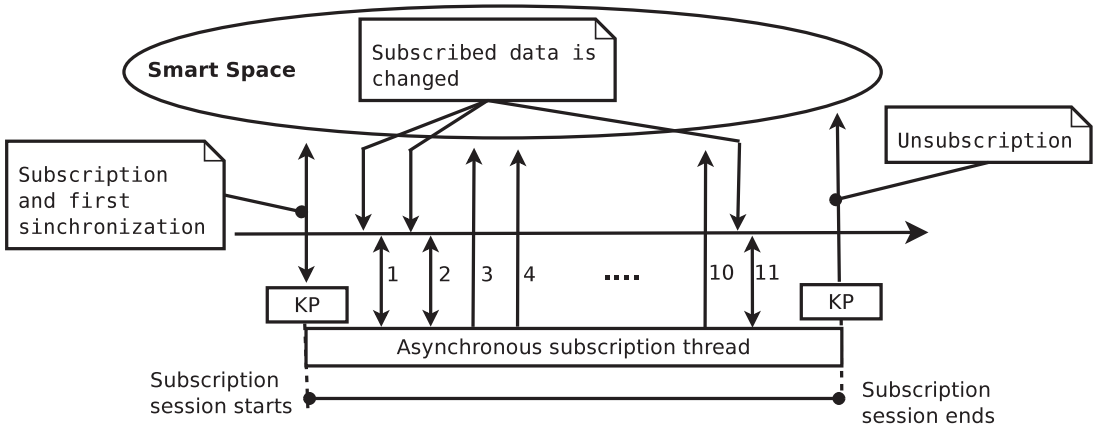

Fig. 9. Time diagram of the asynchronous subscription. The subscription is processed in a background: 1,2,11 – new data are received and processed; 3,4 ... 10 – no new data.

TABLE III
CODE EXAMPLES OF SMARTSLOG SUBSCRIPTION.

ANSI C	C#
Creating a subscription container	
`subscription_container_t *container = new_subscription_container();`	`SubscriptionContainer container = new SubscriptionContainer();`
Adding individuals and properties to the container	
`list_t *properties = list_get_new_list(); list_add_data(property, properties); add_individual_to_subscription_container( container, individual, properties);`	`List<Property> properties = new List<Property> { NameProp, SurnameProp }; container.Add( individual, properties);`
Synchronous subscription of the cantainer	
`if (ss_subscribe_container(container, false) != 0) { printf("Can not subscribe"); ... }`	`if (node.Subscribe(container) == false) { Console.WriteLine( "Can not subscribe"); ... }`
Tracking the subscription notifications synchronously	
`while(...){ wait_subscribe(container); ... }`	`while(...) { node.WaitSubscribe(container); ... }`
Unsubscribe the container	
`ss_unsubscribe_container(conatiner);`	`node.Unsubscribe(container);`

TABLE IV
SUBSCRIPTION TYPES POSSIBLE IN SMART-M3

	Type	Description
1	Triple-based subscription	Subscription based on triple-patterns and manipulations with triples.
2.1	Property-based subscription: data properties	This type for individual's set of properties that holds data values.
2.2	Property-based subscription: object properties	This type for individual's set of properties that links current individual with other individuals.
2.3	Property-based subscription: data properties, object properties	It is possible to subscribe both on data-properties and object properties in the same time.
SmartSlog plans for future		
3	Constraints to value of data property	It is possible to subscribe to triples with concrete values (objects) using triple-based subscription, but there is no method to use subscription for setting constrains of values.
4	Subscription to a class	This subscription type means that you get all individuals of some class from the smart space.

with a multitude of RDF triples in her KP code. All OWL $\leftrightarrow$ RDF transformations are done internally in the SmartSlog ontology library. Content inconsistencies can be automatically detected on the OWL level (e.g., an object property refers to an individual of incorrect class). Note that when the programmer writes the code on the RDF triple level such inconsistencies appear because of the absence of a type-checking mechanism. Consequently, defective actions from KPs can easily result in a set of RDF triples forming an incorrect RDF graph.

The scheme can be enhanced with other advanced subscription types such as subscription to events when updated property values satisfy certain constraints or subscription to OWL classes when insertions, updates, and removals of individuals of a given class are detected. We leave this topic to our further research.

ACKNOWLEDGMENT

Authors would like to thank Finnish-Russian University Cooperation in Telecommunications (FRUCT) program for the provided support and R&D infrastructure. We would also like to thank Sergey Balandin, Iurii Bogoiavlenskii, and Vesa Luukkala for providing feedback and guidance.

REFERENCES

[1] 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.

[2] 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.

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

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

[5] D. G. Korzun, S. I. Balandin, V. Luukkala, P. Liuha, and A. V. Gurtov, "Overview of Smart-M3 principles for application development," in *Proc. Int'l Conf. Artificial Intelligence and Systems (AIS 2011)*, Sep. 2011.

[6] S. Boldyrev, I. Oliver, R. Brown, J.-M. Tuupola, A. Palin, and A. Lappeteläinen, "Network and content aware information management," in *Proc. 4th Int'l Conf. Internet Technology and Secured Transactions*, ser. ICITST 2009. IEEE, 2009, pp. 1–7.

[7] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, pp. 114–131, June 2003.
[8] D. Korzun, A. Lomov, P. Vanag, J. Honkola, and S. Balandin, "Generating modest high-level ontology libraries for Smart-M3," in *Proc. 4th Int'l Conf. Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010)*, Oct. 2010, pp. 103–109.
[9] A. Lomov, P. Vanag, and D. Korzun, "Multilingual ontology library generator for Smart-M3 application development," in *Proc. 9th Conf. of Open Innovations Framework Program FRUCT and 1st Regional MeeGo Summit Russia–Finland*, Apr. 2011, pp. 82–91.