Multilingual Ontology Library Generator for Smart-M3 Application Development

Alexandr A. Lomov, Pavel I. Vanag, Dmitry G. Korzun
Department of Computer Science
Petrozavodsk State University, PetrSU
Petrozavodsk, Russia
{lomov, vanag, dkorzun}@cs.karelia.ru

Abstract

A Smart-M3 application consists of knowledge processors (KPs) that share the smart space. OWL ontology allows structuring smart space content in high-level terms of classes, relations between them, and their properties. A Smart-M3 semantic information broker (SIB) maintains the smart space in low-level terms of RDF triples. A KP can use an ontology library to access the smart space using high-level ontology terms instead of low-level triples. This paper describes SmartSlog, an ontology library generator. It provides API to communicate with SIB and data structures to represent ontology entities in KP code. SmartSlog uses a multilingual generation scheme, reducing the dependence on particular programming languages. (Current release includes the support for ANSI C and C#.) SmartSlog follows the multi-ontology approach when several ontologies can be used in the same smart space. SmartSlog is oriented to tunable optimization of KP code size, system dependencies, device CPU/memory consumption, network load, and data synchronization.

Index Terms: Smart-M3, Ontology, Code Generator, Knowledge Processor.

I. Introduction

Smart-M3 is an open-source platform for information sharing [1]. It provides applications with a smart space infrastructure to use a shared view of dynamic knowledge and services [2]. Services are implemented as distributed agents (knowledge processors, KP) running on the various computers, including mobile and embedded devices. Knowledge is kept in a single RDF triple storage and is accessible via a semantic information broker (SIB).

SmartSlog is a Smart Space ontology library generator [3] for Smart-M3 applications. It maps an OWL ontology description to code (ontology library), abstracting the ontology and smart space communication for KP application logic. Initially SmartSlog has been developed targeting low-performance devices. It produces ontology libraries in pure ANSI C with minimal dependencies to system libraries, the property is essential in many embedded systems [4].

ANSI C programming is too low-level for some classes of modern devices. For example, although writing KP in ANSI C for the Blue&Me platform (Windows mobile for Automotive) is possible, it is indeed complicated. Many developers would prefer using .NET/C# language for this case. In this paper we introduce further development of SmartSlog aiming at support of multilingual ontology libraries. To validate the approach we implemented the next SmartSlog release that supports ontology libraries in ANSI C and C#.

The main contribution of this paper is our extension to the Smart-M3 family of ontology library generators. SmartSlog becomes a multilingual generator that implements ontology-specific modules in code templates and their handlers. Other modules are kept independent on programming language specifics. Although the syntax of ontology library API and the corresponding internal details depend on the particular language, the semantics is similar. It allows applying the same ontological and optimization methods for improving the KP development and code efficiency in the multilingual case.

The rest of the paper is organized as follows. Section II overviews the existing KP development tools with focus on SmartSlog as our reference use case. In Section III we introduce modifications to the SmartSlog generation scheme to support multilingual ontology libraries. Section IV analyzes the problems that are common for ontology library generators independently on target programming languages. Section V summarizes the paper.

II. Smart-M3 Ontology Libraries

Although Smart-M3 aims in following the ubiquitous computing vision, the existing KP development tools are language-specific and platform-dependent. In this section we overview available tools and motivate our multilingual approach for Smart-M3 applications with KPs written in different languages and running on different platforms. Our reference use case is SmartSlog [3], based on which we implement and validate our solution to multilingual ontology library generation.

The developers of the KP application logic use a KP interface (KPI) to access information in smart space. The content conform the ontological description. Low-level access requires the user code to operate with RDF triples (directly following the SSAP operations, where triples are basic exchange elements). In contrast, high-level KP development is based on an ontology library. It allows the user code to be written using high-level ontology entities (classes, relations, individuals); they implemented in the code with predefined data structures and methods. Table I shows available low-level KPIs and ontology library generators for several popular programming languages.

An ontology library simplifies constructing KP application logic proving the developer a programming language view to the concepts defined in a given ontology. The number of

TABLE I
KP Interfaces to Smart-M3 Smart Space

Library	Description
Low-level KP programming: RDF triples	
Whiteboard, Whiteboard-Qt + QML	Language: C/Glib, C/Dbus, C++/Qt. Network: TCP/IP, NoTA. BSD license. A part of the Smart-M3 distribution, http://sourceforge.net/projects/smart-m3/
KPI_Low	Language: ANSI C. Network: TCP/IP, NoTA. GPLv2. Primarily oriented to low-performance devices. VTT-Oulu Technical Research Centre (Finland), http://sourceforge.net/projects/kpilow/
Smart-M3 Java KPI library	Language: Java. Network: TCP/IP. University of Bologna (Italy) and VTT-Oulu Technical Research Centre (Finland), http://sourceforge.net/projects/smartm3-javakpi/
M3-Python KPI (m3_kp)	Language: Python. Network: TCP/IP. BSD license. A part of the Smart-M3 distribution, http://sourceforge.net/projects/smart-m3/
C# KPI library	Language: C#. Network: TCP/IP. University of Bologna (Italy).
High-level KP programming: OWL ontology	
Smart-M3 ontology to C-API generator	Language: Glib/C, Dbus/C. Network: TCP/IP, NoTA. BSD license. A part of the Smart-M3 distribution, http://sourceforge.net/projects/smart-m3/
Smart-M3 ontology to Python generator	Language: Python. Network: TCP/IP, NoTA. BSD license. A part of the Smart-M3 distribution, http://sourceforge.net/projects/smart-m3/
SmartSlog	Language: ANSI C, C#. Network: TCP/IP, NoTA. GPLv2. Petrozavodsk State University (Russia), http://sourceforge.net/projects/smartslog/

domain elements is reduced since an ontology entity consists of many triples. The library API is generic: its syntax does not depend on a particular ontology, ontology-related names do not appear in names of API methods, and ontology entities are used only as arguments. For example, creating an individual of lady Aino Peterson can be written in C as

```
individual_t *aino = new_individual(CLASS_WOMAN);
set_property(aino, PROPERTY_LNAME, "Peterson");
```

Figure 1 shows the SmartSlog ontology library structure. It consists of two parts: ontology-independent and ontology-dependent. The former is the same for any KP and implements generic API to access knowledge in the smart space. The latter is produced by SmartSlog CodeGen by a given OWL description (provided by the KP developer) and implements data structures for particular ontology entities. The library internally performs OWL-RDF transformations and calls a low-level KPI (e.g., KPI_Low) for data exchange with SIB. If the low-level KPI is in different language then wrappers are needed for corresponding calls.

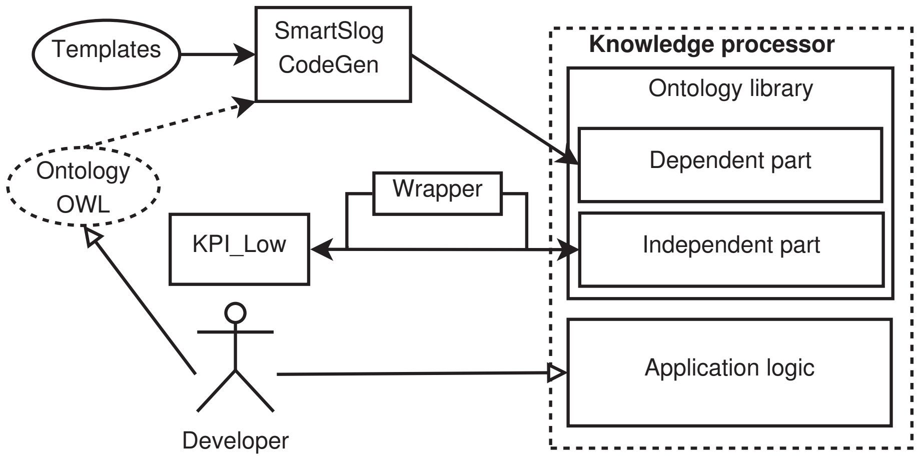

Fig. 1. The SmartSlog ontology library architecture: ontology-dependent and ontology-independent parts

The model of code generation is similar for all three ontology library generators from Table I. They use a common Jena-based back-end for analyzing the ontologies. SmartSlog API and Smart-M3 ontology to C-API share the same core [3]. In contrast, SmartSlog is more concerned with restrictions of low-end devices. It keeps dependencies to minimum and memory usage is predictable and bounded. Furthermore, SmartSlog originally has focused on efficiency optimization. For instance, search requests are written compactly by defining only what is needed for or known about the object to find in the smart space (even if the object has many other properties). In the multilingual case, this original focus is preserved, and more platform restrictions and abilities are taken into the consideration.

III. SMARTSLOG GENERATION SCHEME

In this section we extend SmartSlog to support multilingual ontology library generation. Figure 2 shows our generic scheme. It inherits many properties from the ANSI C version of SmartSlog [3]. We validated this scheme by extending SmartSlog with C# support.

The scheme defines two key steps a KP developer performs. First, the developer provides a problem domain specification as an OWL description. The generator inputs the description and outputs the ontology-dependent part of the ontology library. The latter is composed with

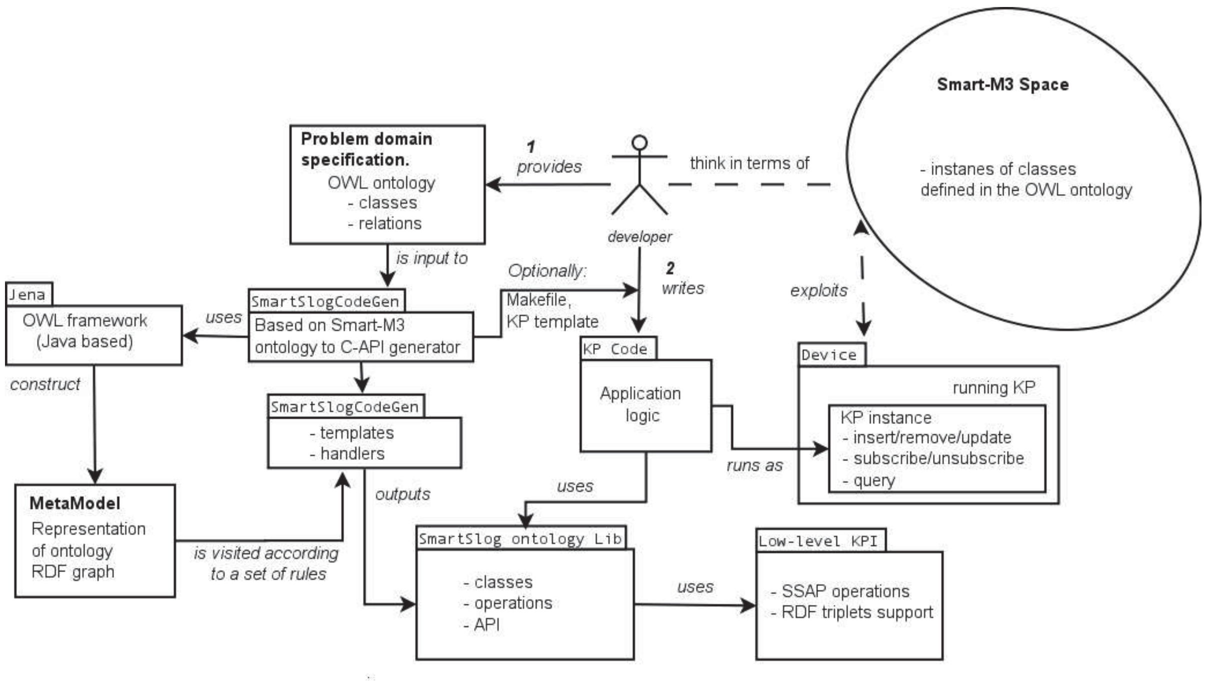

Fig. 2. Smart-M3 ontology library generation scheme.

the ontology-independent part forming the ontology library for the target language. Second, the developer directly uses the library in the KP code.

SmartSlog CodeGen is written in Java and implements generation of the ontology-dependent part of the library. The following static templates/handlers scheme is used. Code templates are “pre-code” of data structures that implement ontology classes and their properties. Each template contains a tag $\langle \mathrm{name} \rangle$ instead of a proper name (unknown in advance). A handler transforms one or more templates into final code replacing tags with the names from the ontology. Templates and handlers are language-specific.

This scheme belongs to a class of source code generators [5] where templates define an ontological model for the generation process and handlers implement template processors. The transformation follows the concept of automatic programming [6]. High-level objects (tags) are transformed to low-level elements (names in source code) by a set of logical applicability conditions (handlers). The scheme is horizontal transformation since only names of data structures and arguments in methods are affected.

SmartSlog CodeGen utilizes Jena toolkit [7] to construct an RDF ontology graph (Jena meta-model). The graph is iteratively traversed. When a node is visited its appropriate handlers are called to transform templates into final code. Optionally, a KP template (a code skeleton) can be generated, and the developer can easier start writing her KP.

The ontology-independent part implements API providing basic data structures/classes (for generic ontology class, property, and individual) and functions/methods for their manipulation. Internally it also implements all high-level ontology entity transformations to low-level RDF triples and vice versa. Calls to KPI_Low is used for communications with SIB. Since KPI_Low is written in C, the C# version needs a KPI_Low wrapper. Note that our scheme permits other low-level KPI, different from KPI_Low.

Based on this scheme, introducing a new language needs the following appropriate language-specific modules.

- Templates and handlers in the generator.
- Ontology-independent part of the library.
- Interface to the low-level KPI.

Our current implementation[1] of this scheme supports ANSI C and C# languages in SmartSlog.

IV. Design Features

The same ontological and optimization methods, which improves the KP development and code efficiency, can be applied in the multilingual case. Our solutions that have been already implemented in the ANSI C version of SmartSlog we straightforwardly ported to the C# version, see [3] for their description. In this section we discuss our most recent design solutions related to the adaptation of ontological and optimization methods in SmartSlog. Their implementation is in progress and will be available in further SmartSlog releases.

A. Ontology composition

Smart space content can be structured with a set of different ontologies instead of a single big ontology. Figure 3 shows that in this case the generator produces a common library for several ontologies.

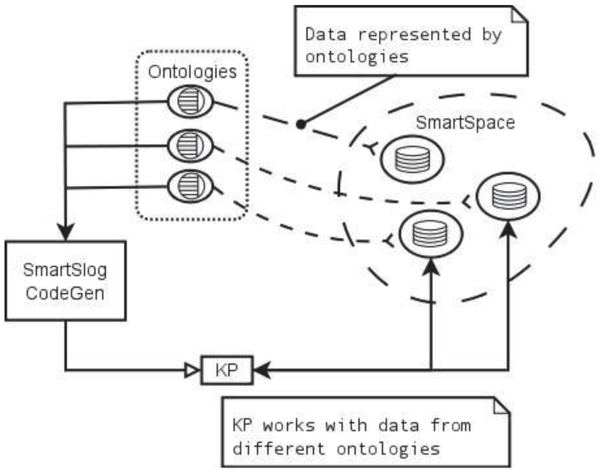
Data represented by ontologies
Ontologies
SmartSpace
SmartSlog CodeGen
KP
KP works with data from different ontologies

Fig. 3. Ontologies intergation schema.

1) Ontology integration: Integration is either complete or partial [8], [9]. Complete integration means that the multiple ontologies are treated as totally combined into a single one. Partial integration means that only some entities (classes, properties) are taken from each ontology. After integration the KP can work with knowledge structured in the smart space with different ontologies.

The KP can cooperate with other KPs even if they access the common smart space differently, e.g., each of them operates with a disjoint part of the space. Given a set of ontologies, the generator produces the library that allows KP to manipulates with entities from the ontologies. All namespaces, entity names are available in KP application logic and it can manipulate with several knowledge sets in the smart space via a single KP. Similarly to the previous SmartSlog design, the developer can select (or deselect) the ontology entities she needs (does not need).

[1]Open-source project, http://sourceforge.net/projects/smartslog/

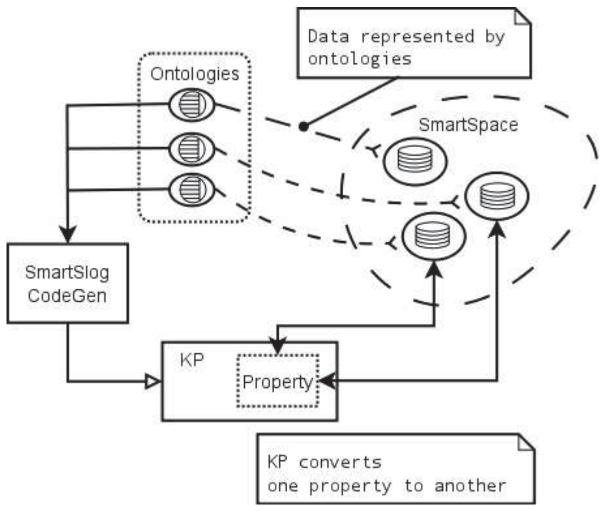
Data represented by ontologies
Ontologies
SmartSpace
SmartSlog CodeGen
KP
Property
KP converts one property to another

Fig. 4. Ontology composition when different ontologies refer to the same property

2) The same property in different ontologies: Figure 4 shows another scheme for composition of multiple ontologies. Assume that there is a mapping that defines what properties are the same in several ontologies of the given set. This mapping is represented with additional properties—bridge properties [10]. Values of such a multi-ontology property are stored in all corresponding parts of the smart space.

The same knowledge can be of different types due to different ontologies. In some cases the type is not important. For example, titles of books are available in different parts of the space. In one part the title corresponds to a printed book. In another part it corresponds to an electronic version of the same book.

The KP code can use the only active property for manipulating with all of the same properties. Active property links all other properties via the bridge property. The latter duplicates the request to corresponding parts of the smart space, and KP accesses values of all properties. Furthermore, bridge property can transform data to common format. For example, if the same properties are date then the bridge property converts the value to the format the KP requires.

3) KP controller: There are smart applications where access to the smart space is controlled by a dedicated KP. Smirnov *et al.* [11] suggested a KP for resolving the problem of simultaneous access to the smart space content. Luukkala and Honkola [12] introduced the same idea for coordinated access to devices. Korzun *et al.* [13] employed a KP mediator for two smart spaces: smart conference and blogosphere.

KP controller has to know several ontologies, see Figure 5. It controls ontology entities that are shared other KPs. The controller publishes control information to the smart space and receives information about KP states to decide further control actions. For example, many devices has the onoff state, it is described differently in different ontologies. If the application needs to turn off several devices, this function can be implemented using a KP controller.

B. KP code optimization

SmartSlog does not optimize its mediator library (KPI_low). Instead, SmartSlog optimizes local data structures, the (de)composition (to)from triples, and the way how the mediator library is used. Some of these optimizations are also usable for computers with no hard performance restrictions.

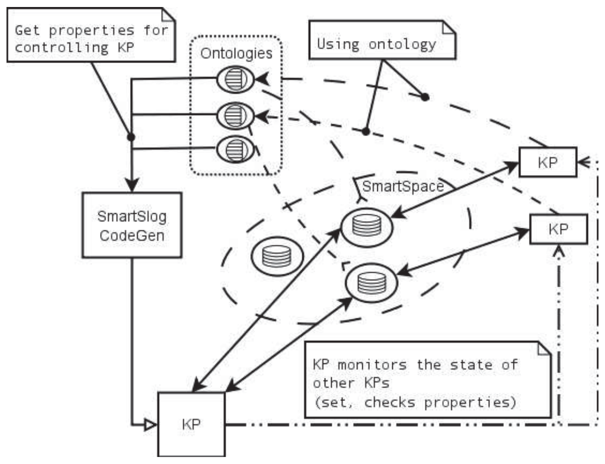
Get properties for controlling KP
Ontologies
Using ontology
KP
KP
SmartSpace
SmartSlog CodeGen
KP monitors the state of other KPs (set, checks properties)
KP

Fig. 5. KP controller schema.

1) Local data structures for ontology entities: Each ontology entity is implemented as a structure/class of constant size. For ontology with N entities the SmartSlog ontology-dependent part is of size $O(N)$.

In many problem domains, however, the whole ontology contains a lot of classes and properties. First, SmartSlog provides parameters (constants) that limits the number of entities, hence the developer can control the code size. Second, if the KP logic needs only a subset of the give ontology, then SmartSlog allows ontology entity selection/deselection.

2) Knowledge patterns: A data model that allows defining ontological objects. It is a base tool for searching and filtering the smart space content. Developer defines objects by composing a pattern for filtering locally available objects or for searching new objects in the smart space.

A pattern for an object contains only a subset of properties needed for filtering/searching. In filtering, these properties are compared with local objects. In searching, these properties are used to find appropriate objects. The result is an object that contains values only for the given properties. This way reduces the amount of data to keep, process, and transfer, even if the actual objects have many properties.

Figure 6 represents a particular case of the ontology instance graph. Objects A, B, C and D with datatype and object properties are related to each other using object properties. Instances of these objects are stored in the smart space and locally at the KP.

Our pattern-based approach admits both local (KP) and SIB implementation. Due to the recent restrictions of Smart-M3 SIB SmartSlog implements patterns only on the local side.

Filtering is used for transferring/delivering necessary parts of objects to/from the smart space. As a result, KP works locally with a subset of properties required by the KP logic at current time instance. There is no need to load/save all properties from/to the smart space. Searching is used to deliver (search) new objects, existing in SS.

A data model describes how to organize the structure of data and defines how to process them. In this way there are two criteria for pattern evaluation of patterns: correctness of defining objects and efficiency of processing.

For searching, patterns allow to define object by ontological class, UUID, and checked properties (properties that object should have). To determine object more intelligently patterns should be extended to support unchecked properties (properties that object should not have) and conditional properties (with relations like $\leq, \geq, \leq, \geq$). It also could be useful in filtering.

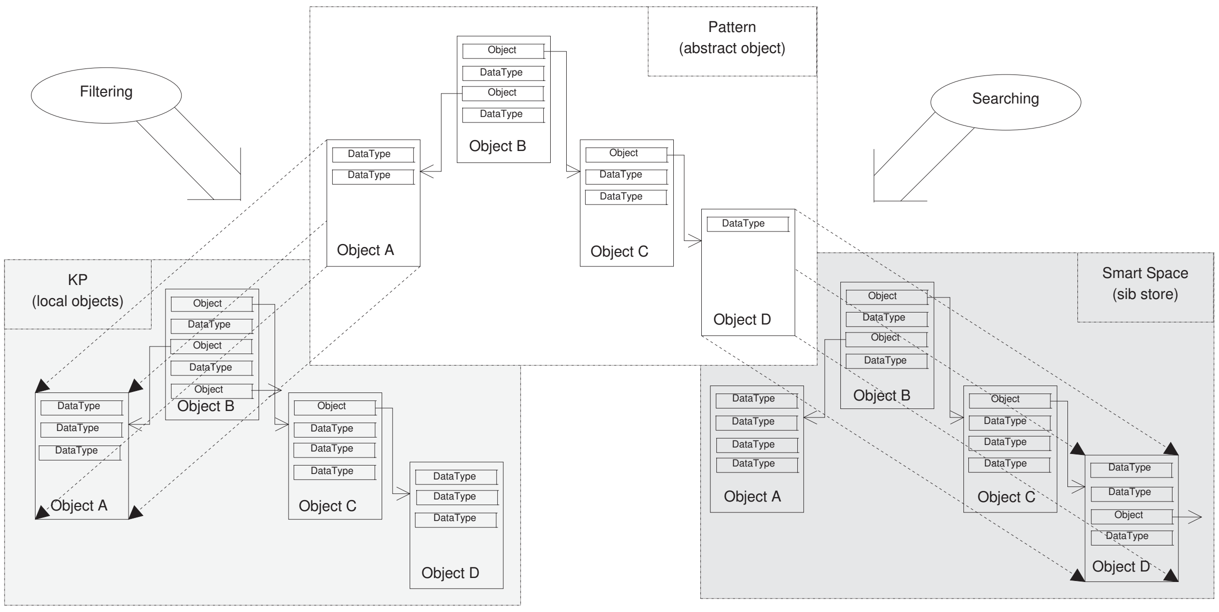

Fig. 6. SmartSlog patterns for filtration and search.

Currently Smart-M3 SIB does not support SPARQL. Therefore, searching lead to transferring a lot of triples and then to their iterative processing. In both cases, composing a pattern for searching is constructing logical query, which could be optimized. Also the access to properties could be optimized using hash tables. It leads to fast access to necessary properties.

3) Synchronization: SmartSlog supports both types of subscriptions: synchronous and asynchronous. The latter case requires threading. SmartSlog uses POSIX threads, available on many embedded systems [4]. Nevertheless, SmartSlog allows switching the asynchronous subscription off if the target device has no thread support.

SmartSlog provides direct access both to SIB and local data. If many KPs asynchronously change data in the smart space, the KP is responsible to keep the data in the actual state. Another way for data synchronization is subscription.

Consider the example in Figure 7. Let A be data to synchronize. After local manipulations A is transformed to A' on the local side. In the smart space it is still A. After synchronization both sides keep the same A'. Then A' is transformed to A'' on the SIB side while A' remains locally. After synchronization the same A'' is on both sides. Δ_1 is the period with stale data in SIB and Δ_2 is the period with locally stale data. To support data in actual state these periods should be minimized.

SmartSlog supports blocking and none-blocking synchronizations (synchronous and asynchronous subscriptions). Both require explicit defining of objects to synchronize. In some cases it could be too difficult or impossible to explicitly point objects to synchronize. Therefore, KP should track for changing of objects itself and keep them up to date.

When an object is changed locally then it is marked for future synchronization. When an object is changed in SIB then KP synchronizes with SIB in the none-blocking mode. As a result, the developer uses only local objects assuming that they are always up to date. There should be some options to control synchronization. For instance, the developer sets the data importance and SmartSlog library synchronizes them accordingly.

There should be a mechanism for determination of synchronization time moments. Let us

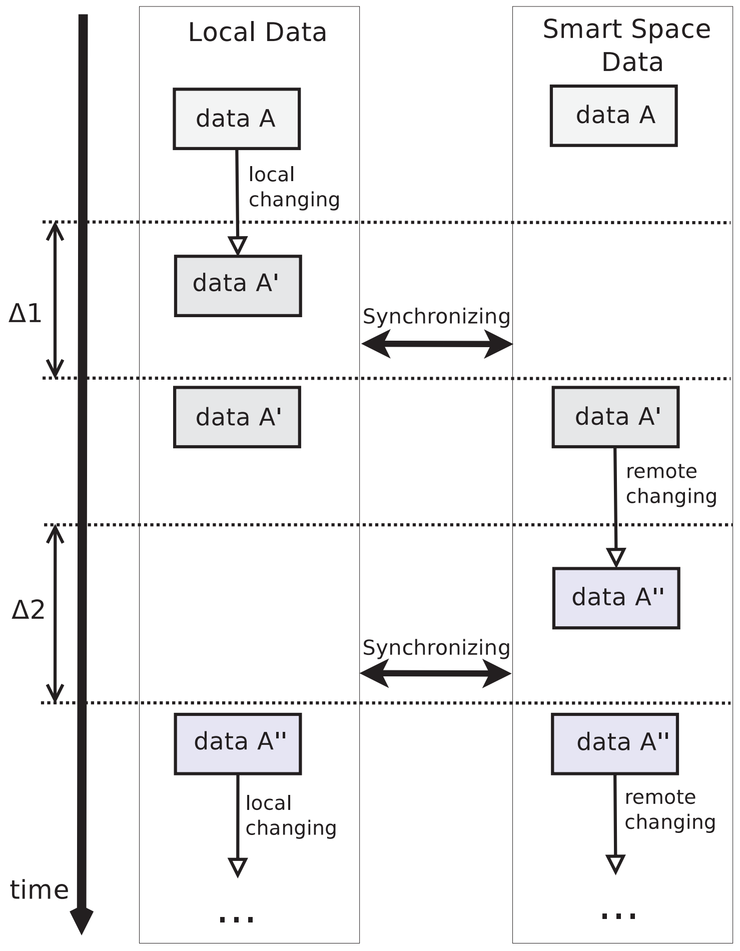

Fig. 7. Synchronization problem. δs are periods of desynchronization.

list parameters that affect this mechanism.

- memory usage (marking of changed objects leads to increasing memory usage, synchronize when memory limit reached),
- duration from last synchronization (synchronize at least one time per some duration),
- update frequency of object (if object changes rarely it could be synchronized immediately),
- network activity (reduce the network load),
- synchronization frequency (reduce the device load),
- device state (device could be busy by more prioritized task)

4) Local triple store: Smart-M3 CodeGen keeps a triple store, a local cache of smart space content. For large ontology it is expensive. In contrast, SmartSlog does not intend to store any triple for long time. Ontology entities are stored in own structures. When a triple is needed it is created and processed. Then the memory is freed immediately after the usage.

V. Conclusion

The addressed area of ontology library generation for multitude of devices is very important. The Smart-M3 platform aims at allowing any device to easily share information with other devices and architectures, large or small. SmartSlog is a tool that supports easy programming of such devices for participating in Smart-M3 applications.

The previous version of SmartSlog [3] was oriented to ANSI C, a particular programming language. This paper contributed a multilingual scheme of ontology library generation. A KP developer can choose among several programming languages for the ontology library. The proposed scheme is available in latest versions of SmartSlog. Its current implementation supports ontology libraries in ANSI C and C#. The operability was tested on Linux- and Windows- based platforms.

Similarly to the previous SmartSlog release for ANSI C, the generated code is compact due to high-level ontology style and portable due to the reduction of system dependencies. This paper described the further progress in this direction. Our design solutions allow adopting advanced ontological and optimization methods in SmartSlog. We showed that the SmartSlog generation scheme supports multiple ontologies if KP needs to access different parts of the smart space. We identified several points in the SmartSlog generation process where certain performance optimization methods can be applied for the problems of device CPU/memory consumption, network load, and data synchronization. Implementation of these ontological and optimization features as well as its experimental confirmation form a topic of our ongoing 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, Jukka Honkola, Vesa Luukkala, and Ronald Brown for providing feedback and guidance.

REFERENCES

[1] J. Honkola, H. Laine, R. Brown, and O. Tyrkkö, "Smart-M3 information sharing platform," in *The 1st Int'l Workshop on Semantic Interoperability for Smart Spaces (SISS 2010) in conjunction with IEEE ISCC 2010*, Jun. 2010.

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

[3] 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.

[4] M. Barr and A. Massa, *Programming Embedded Systems: With C and GNU Development Tools*. O'Reilly Media, Inc., 2006.

[5] K. Czarnecki and U. W. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[6] C. Rich and R. C. Waters, "Approaches to automatic programming," *Advances in Computers*, vol. 37, pp. 1–57, 1993.

[7] B. McBride, "Jena: A semantic web toolkit," *IEEE Internet Computing*, vol. 6, pp. 55–59, November 2002.

[8] N. Choi, I.-Y. Song, and H. Han, "A survey on ontology mapping," *SIGMOD Record*, vol. 35, pp. 34–41, September 2006.

[9] P. Bouquet, M. Ehrig, J. Euzenat, E. Franconi, P. Hitzler, M. Krötzsch, L. Serafini, G. Stamou, Y. Sure, and S. Tessaris, "D2.2.1 specification of a common framework for characterizing alignment," Knowledge Web Consortium, Tech. Rep., Feb. 2005. [Online]. Available: http://knowledgeweb.semanticweb.org/semanticportal/deliverables/D2.2.1v2.pdf

[10] A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz, "Ontologies for enterprise knowledge management," *IEEE Intelligent Systems*, vol. 18, pp. 26–33, March 2003.

[11] A. Smirnov, A. Kashnevik, N. Shilov, I. Oliver, S. Balandin, and S. Boldyrev, "Anonymous agent coordination in smart spaces: State-of-the-art," in *Proc. 9th Int'l Conf. Smart Spaces and Next Generation Wired/Wireless Networking (NEW2AN'09) and 2nd Conf. Smart Spaces (ruSMART'09)*, ser. Lecture Notes in Computer Science, vol. 5764. Springer-Verlag, 2009, pp. 42–51.

[12] V. Luukkala and J. Honkola, "Integration of an answer set engine to smart-m3," in *Proc. 2nd Conf. Smart Spaces (ruSMART'10) and 10th Int'l Conf. Next Generation Wired/Wireless Networking (NEW2AN'10)*, ser. ruSMART/NEW2AN'10. Springer-Verlag, 2010, pp. 92–101.

[13] D. Korzun, I. Galov, A. Kashevnik, K. Krinkin, and Y. Korolev, "Blogging in the smart conference system," in *Proc. 9th Conf. of Open Innovations Framework Program FRUCT*, Apr. 2011.