Multilingual Ontology Library Generator for Smart-M3 Information Sharing Platform

Dmitry G. Korzun*†, Alexandr A. Lomov*, Pavel I. Vanag*, Sergey I. Balandin‡, and Jukka Honkola§
*Department of Computer Science
Petrozavodsk State University – PetrSU
Petrozavodsk, Russia
†Helsinki Institute for Information Technology – HIIT
Aalto University
Helsinki, Finland
‡FRUCT Oy
Helsinki, Finland
§Innorange Oy
Helsinki, Finland
Email: {dkorzun, lomov, vanag}@cs.karelia.ru, sergey.balandin@fruct.org, jukka@innorange.fi

***Abstract*—Web Ontology Language (OWL) allows structuring smart space content in high-level terms of classes, relations between them, and their properties. Smart-M3 is an open-source platform that provides a multi-agent distributed application with a shared view of dynamic knowledge and services in ubiquitous computing environments. A Smart-M3 Semantic Information Broker (SIB) maintains its smart space in low-level terms of triples, based on Resource Description Framework (RDF). This paper describes SmartSlog, a software development tool for programming Smart-M3 agents (Knowledge Processors, KPs) that consume/produce smart space content according with its high-level ontological representation. SmartSlog applies the code generation approach. Given an OWL ontology description, SmartSlog produces the ontology library. The latter provides 1) API to access the smart space via its SIB and 2) data structures and functions to represent and maintain locally in KP code all ontology classes, relations, properties, and individuals. The developer easier constructs the KP code, thinking in high-level ontology terms instead of low-level RDF triples. SmartSlog supports generation of multilingual ontology libraries (ANSI C and C# in the current implementation). Such libraries are modest to the device capacity, portable and suitable even for small devices. The SmartSlog ontology library generation scheme, architecture, design solutions, and directions for use are the main output of this paper.**

***Keywords*-Smart spaces; Smart-M3; OWL/RDF ontology; code generator; knowledge processor; low-performance devices**

I. Introduction

Smart-M3 is an open-source platform for information sharing [1]–[3]. It provides applications with a smart space infrastructure to use a shared view of dynamic knowledge and services in ubiquitous computing environments [4]. Applications are implemented as distributed agents (knowledge processors, KPs) running on the various computers, including mobile and embedded devices. Shared knowledge is represented using Resource Description Framework (RDF) and kept in RDF triple-stores, each is accessible via a Semantic Information Broker (SIB). The RDF representation allows semantic reasoning; simple methods are available on the SIB side, more complex ones are implemented in dedicated KPs.

A Smart-M3 application consists of several KPs that share the smart space using the space-based [5] and pub/sub [6] communication models. The KPs produce (insert, update, remove) or consume (query, subscribe/unsubscribe) information. The Smart Space Access Protocol (SSAP) implements the SIB $\leftrightarrow$ KP communication, using operations with RDF content as parameters. Each KP understands its subset of information, usually defined by the KP ontology.

Real-life scenarios often involve a lot of information, which leads both to largish ontologies and possibly complex instances that the KPs need to handle. Thus, programming KPs on the level of SSAP operations and RDF triples bring unnecessary complexity for the developers, who have to divert effort for managing triples instead of concentrating on the application logic. The OWL representation of knowledge as classes, relations between classes, and properties maps quite well to object-oriented paradigm in practice (but not so well in theory). Therefore, it is feasible to map OWL classes into object-oriented classes and instances of OWL classes into objects in programming languages. (These objects only have attributes, but no methods and thus no behavior.) This approach effectively binds the RDF subgraph describing an instance of an OWL class (individual) to an object in a programming language.

SmartSlog is a Smart Space ontology library generator [1], [7] for Smart-M3. It maps an OWL ontology description to code (ontology library), abstracting the ontology and smart space access in KP application logic. As a result, SmartSlog simplifies constructing KP code compared with the low-level

RDF-based KP development. The code manipulates with ontology classes, relations, and individuals using predefined data structures and library Application Programming Interface (API). The number of domain elements in KP code is reduced. The API is generic, hence does not depend on concrete ontology; all ontology entities appear as arguments in API functions. Search requests to SIB are written compactly by defining only what you know about the object to find (even if the object has many other properties).

The vision of ubiquitous involves a lot of small devices to participate in surrounding computing environments. SmartSlog targets low-performance devices by producing ontology libraries in pure ANSI C with minimal dependencies to system libraries, the property is essential in many embedded systems [8]. SmartSlog takes into account the limited resources available on small computers such as mobile and embedded devices. For example, the KP code does not need to maintain the whole ontology as unused entities can be removed. Also, RDF triples are not kept indefinitely, and the local memory is freed immediately after the use. Even if a high-level ontology entity consists of many triples, its synchronization with SIB transfers only a selected subset, saving on communication.

ANSI C programming is too low-level for some classes of devices. For example, although writing KP in ANSI C for the Blue&Me platform (Windows mobile for Automotive) is possible, it is complicated, and some developers prefer the .NET/C# language for this case. SmartSlog allows multilingual ontology library generation. The current SmartSlog implementation supports ontology libraries in ANSI C and C#, validating our multilingual approach.

The rest of the paper is organized as follows. Section II provides an introduction to the Smart-M3 platform. Section III overviews Smart-M3 KP development tools with focus on SmartSlog; we describe the ontology library approach for KP development. In Section IV, we introduce the ontology library generation scheme designed for SmartSlog. Example application construction with SmartSlog is shown in Section VI. Then, Section VII analyzes the problems that are common for ontology library generators independently on target programming languages. It includes the issues of ontology manipulations and code optimization on the KP side. Section VIII summarizes the paper.

II. Smart-M3 Platform and its Notion of Application

Smart-M3 is an open-source interoperability platform for information sharing [2], [3], [9]. "M3" stands for Multidevice, Multidomain, and Multivendor. It has been developed by a consortium of companies and within research projects: EU Artemis funded Sofia project (Smart Objects for Intelligent Applications) [10] and Finnish nationally funded program DIEM (Device Interoperability Ecosystem) [11]. Smart-M3 implements smart space infrastructures for multi-agent distributed applications following the smart space concept [12]–[14]; the latter is becoming popular in semantic computing. In this section we provide an overview of Smart-M3 platform and its core concepts.

A. Space-based computing

Space-based (or tuplespace) computing has its roots in parallel and distributed programming. Gelernter [15] defined the generative communication model where common information is shared in a tuplespace; parallel processes of a distributed application cooperate by publishing/retrieving tuples into/from the space. A tuple is an ordered list of typed fields. Data tuples contain static data. Process tuples represent processes under execution. This asynchronous (publish-based) inter-process communication model allows building programs by gluing together active pieces [5].

Aiming at automated processing in such a giant distributed system as the World Wide Web, Berners-Lee [16] introduced the Semantic Web. Its content is described in a structured manner, where ontologies become the basic building block. Fensel [17] brought the idea of triple space computing as communication and coordination paradigm based on the convergence of space-based computing and the Semantic Web. Triple space computing inherits the publication-based communication model from the tuplespace communication model and extends it with semantics: tuples are RDF triples (subject, predicate, object). They in turn are composed to RDF graphs with subjects and objects as nodes and predicates as edges. Hence, semantic-aware queries to the space are possible, utilizing matching algorithms [18] and semantic query languages like SPARQL [19].

In fact, the triple space computing paradigm states a scalable semantic infrastructure for web applications; it enables integration, communication, and coordination of many autonomous, distributed, and heterogeneous web service providers and consumers. Information stored in the same space can be further processed, providing deduced knowledge that otherwise cannot be available from a single source [5]. Semantic web spaces [20] apply this possibility for a new coordination model: a participant can infer new facts as a reaction to knowledge that has been published by others. Semantic web spaces extends tuplespaces: tuples are RDF triples and matching uses RDF Schema reasoning.

Conceptual Spaces (CSpaces) [21] extends triple space computing to be applicable in different scenarios apart from web services. An important set of scenarios is due to the ubiquitous computing vision, i.e. when computers seamlessly integrate into human lives and applications provide right services anywhere and anytime [22]. One of the key features of CSpaces is a composition of the tuplespace publish-based model with the publish/subscribe model from the pub/sub communication paradigm (e.g., see [6]). Transaction support is included to guarantee the successful execution of a group of operations. This advanced coordination model provides flow decoupling from the client side [6], in addition to time and space decoupling already available in the tuplespace coordination model.

B. Smart spaces and Smart-M3 infrastructure

Smart spaces constitute a smart environment, which is “able to acquire and apply knowledge about its environment and to adapt to its inhabitants in order to improve their experience in that environment” [12]. In accordance with the ubiquitous computing vision, smart spaces encompass the following information spaces: (i) physical spaces with sensing devices such as homes or cars, (ii) service spaces with information retrieval and processing such as Internet services or surrounding services in tourist place, and (iii) user spaces with personal information such as user profiles or address books. The information is dynamically shared by multiple heterogeneous participants (humans and machines), allowing each user to interact continuously with the surrounding environment, and the services continuously adapt to the current needs of the user [14].

Smart spaces require a software infrastructure that turns the constituting spaces into programmable distributed entities. Smart-M3 provides such an infrastructure to use a shared view of dynamic knowledge and services within a distributed application. Although several studies have showed the convenience of the space-based approach for ubiquitous and pervasive computing environments and even for Internet of Things [23]–[25], to the best of our knowledge the Smart-M3 platform is the only general-purpose open-source platform available recently.

In addition to the normal range of personal computers and embedded devices, mobile devices with various means of connectivity become the primary gateway to the service space and the major storage point in the user space [13], [14]. Smart-M3 follows the space-agent approach. Each device, service, or storage point is programmable as an agent. In this ***multi***device system, agents place, share, and manipulate with local and global information using their own locally agreed semantics [13].

Information sharing in Smart-M3 is based on the space-based models using the same mechanisms as in the Semantic Web, thus allowing ***multi***domain applications, where the RDF representation allows easy exchange and linking of data between different ontologies, making cross-domain interoperability straightforward [26]. Smart-M3 currently supports only limited reasoning, e.g., queries with subclass relations; see [27] for more details and possible extensions. The security issues of information sharing in Smart-M3 can be found in [28]–[30].

The basic architecture of Smart-M3 space infrastructure is illustrated in Figure 1. Its core component is semantic information broker (SIB)—an access point to the smart space. Each SIB maintains a part of information represented

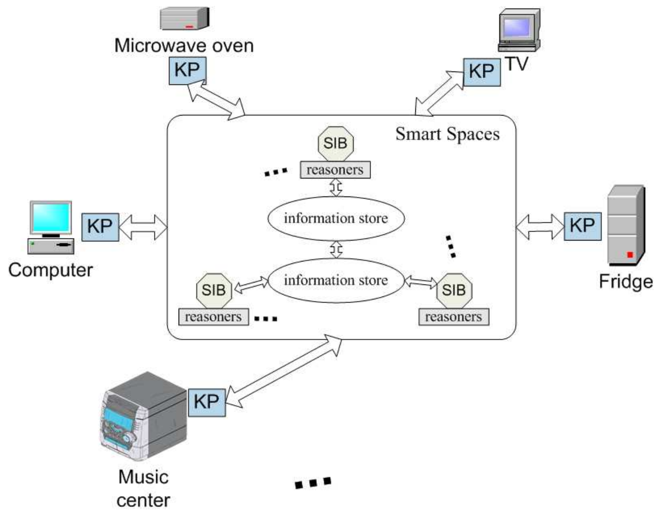

Figure 1. Smart spaces form a publish/subscribe system in a ubiquitous environment: KPs run on various types of computers and devices, the distributed knowledge store supports reasoning over cross-domain information

as an RDF triplestore. It provides simple reasoning, e.g., understanding the owl:sameAs concept. The current Smart-M3 implementation supports WilburQL as a basic query language; migration to SPARQL is in progress. Note that WilburQL was originally conceived as Nokia Research Center’s toolkit (Helsinki, Finland) for applications that use RDF, written in Common Lisp; the new Python-based toolkit (Piglet) is partially open-sourced as a part of Smart-M3.

A device participates in the space using a software agent—knowledge processor (KP). A KP connects a SIB over some network and can modify and query the information by insert, remove, update, query, and (un)subscribe operations using the smart space access protocol (SSAP). Each SIB provides many network connectivity mechanisms (e.g., HTTP, plain TCP/IP, NoTA, Bluetooth), yielding ***multi***vendor device interoperability. Accessing the space is session-based with join and leave operations, thus providing the base for mechanisms of access control and secure information sharing.

From the KP point of view the information in the space constitutes an RDF graph, usually according to some OWL ontology. The use of any specific ontology is not mandated, and a group of KPs can locally agreed which ontology to use for interpreting a certain part of the space. The consistency of stored information is not guaranteed. KPs are free to interpret the information in whatever way they want.

When several SIBs make up a smart space the SIB network follows a protocol with distributed deductive closure [31]. Hence any KP sees the same information content regardless the SIB it connects to. The current implementation supports the simplest case with one SIB only.

C. Smart-M3 applications

A Smart-M3 application can be considered a composition of possible scenarios enabled by a certain group of KPs. Execution of the composition targets the current needs of the user. For instance, an email application consists of the following scenarios [26]: sending, receiving, composing, and reading email. Each scenario can be implemented by a dedicated KP. The same KP can be used in different applications. For instance, a KP for composing email can be a part of application for browsing social networks.

From this point of view, the basic principle is that the user has a collection of KPs. They are capable to execute certain scenarios. If the given collection does not support a needed scenario, additional KPs should be found. Each KP should understand its own, non-exclusive set of information. The set is typically described with the ontology of the KP, at least implicitly. Overlap of the sets of different KPs is needed for interoperability; the KPs can see each other actions.

An application is constructed as ad-hoc assembly of KPs. Each scenario emerges from the observable actions taken by KPs based on smart space content or from the use of available services. Some scenarios can be transient: the execution is changed as the participating KPs join and leave the smart space as well as some services become available or unavailable.

The aim of this Smart-M3 approach is at the ease of combining multiple scenarios into various applications. The key point is the loose coupling between the participating KPs. They use the space-based and pub/sub communication models modifying and querying the information in the common smart space. Thus, the effect of any KP participation to others is limited by to the information the KP provides into the space. Note that Smart-M3 does not prevent direct contacts between KPs, thus some actions can be activated based on traditional inter-process communication models. Adding elements of this traditional approach, however, impedes the easy use of affected KPs in other applications, reducing the benefit from Smart-M3.

Concrete examples of Smart-M3 applications include context gathering in meetings [32], organization of conferences and meetings [33], [34], smart home [35], gaming, wellness and music mashup [26], social networks [36], and semantic multi-blogging [37].

III. Smart-M3 Ontology Libraries and Related Work

Existing Smart-M3 KP development tools are language-specific and platform-dependent. Many of them are oriented to the RDF representation of information, thus complicating the KP code compared with the OWL representation. In this section we overview available tools and motivate the ontology library approach for Smart-M3 applications. The approach is implemented in SmartSlog with possibility to write KPs in different languages and for different platforms.

The developers of KP application logic use a KP Interface (KPI) to access information in the 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 with triples as 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 providing the developer a programming language view to the concepts of the given ontology. The number of 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 2 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 for data exchange with SIB. In particular, the current SmartSlog implementation uses KPI_Low, both for ANSI C and C# ontology libraries. If the low-level KPI is in a different language then a kind of wrappers can be used for corresponding calls. For instance, SmartSlog utilizes wrappers to implement a C# ontology library since it uses KPI_Low written in C.

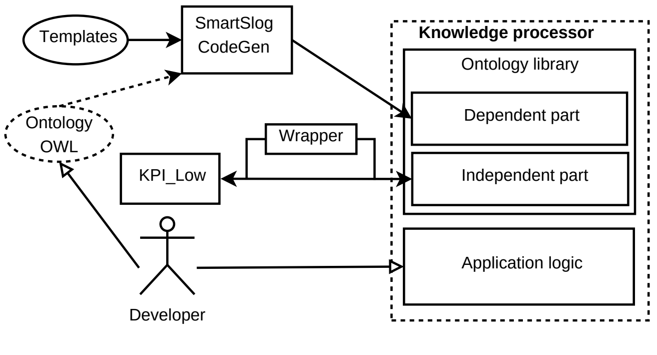

Figure 2. The SmartSlog ontology library architecture: ontology-dependent and ontology-independent parts

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/
Smart-M3 PHP KPI library	Language: PHP. Network: TCP/IP. University of Bologna (Italy), http://sourceforge.net/projects/sm3-php-kpi-lib/
C# KPI library	Language: C#. Network: TCP/IP. University of Bologna (Italy), http://sourceforge.net/projects/m3-csharp-kpi/
	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/

This library division into two parts improves development of Smart-M3 applications. If the ontology changes the ontology-independent part does not require recompiling; it is shared by several KPs although they use different ontologies. Ontology-dependent part can be shared by KPs with the same ontology. These cases are typical since multiple smart space applications with different ontologies can run on the same device as well as multiple KPs form one smart space application with a common ontology.

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. 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 is 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).

Ontology based code generation facilities are also provided as part of the Sofia ADK [38] for Java-based KPs. The Sofia ADK is an Eclipse-based toolset for creating smart space applications. The view towards software developer is very similar to the SmartSlog, namely providing programming language view to the concepts defined in an ontology.

Similar ideas also exist in the semantic web world, with projects aiming to provide object-RDF mapping libraries (in the spirit of object-relational mapping). These libraries are typically not tied to any ontology and implemented in interpreted languages, such as RDFAlchemy [39] in Python or Spira [40] in Ruby. Obviously the approach is very difficult both to implement and to use in statically typed compiled languages such as C, while very convenient in dynamically typed, interpreted languages.

IV. ONTOLOGY LIBRARY GENERATION SCHEME

In this section, we describe the multilingual ontology library generation scheme used in SmartSlog. Figure 3 shows the scope. We practically approved this scheme implementing the support for generating libraries in ANSI C and C# programming languages.

The scheme defines two basic 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 the ontology-independent part forming the ontology library for the target language. Second, the developer applies the library in the KP code by using given data structures and calling API functions. As a result, the KP logic is implemented in high-level terms of the specified ontology.

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 $\langle \text{name} \rangle$ 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 [41] where templates define an ontological model for the generation process and handlers implement template processors. The transformation follows the concept of automatic programming [42]. High-level objects (tags) are transformed to low-level elements (names in source code) by a set of logical applicability conditions (handlers). The scheme applies the horizontal transformation since only names of data structures and arguments in methods are affected.

SmartSlog CodeGen utilizes Jena toolkit [43] to construct

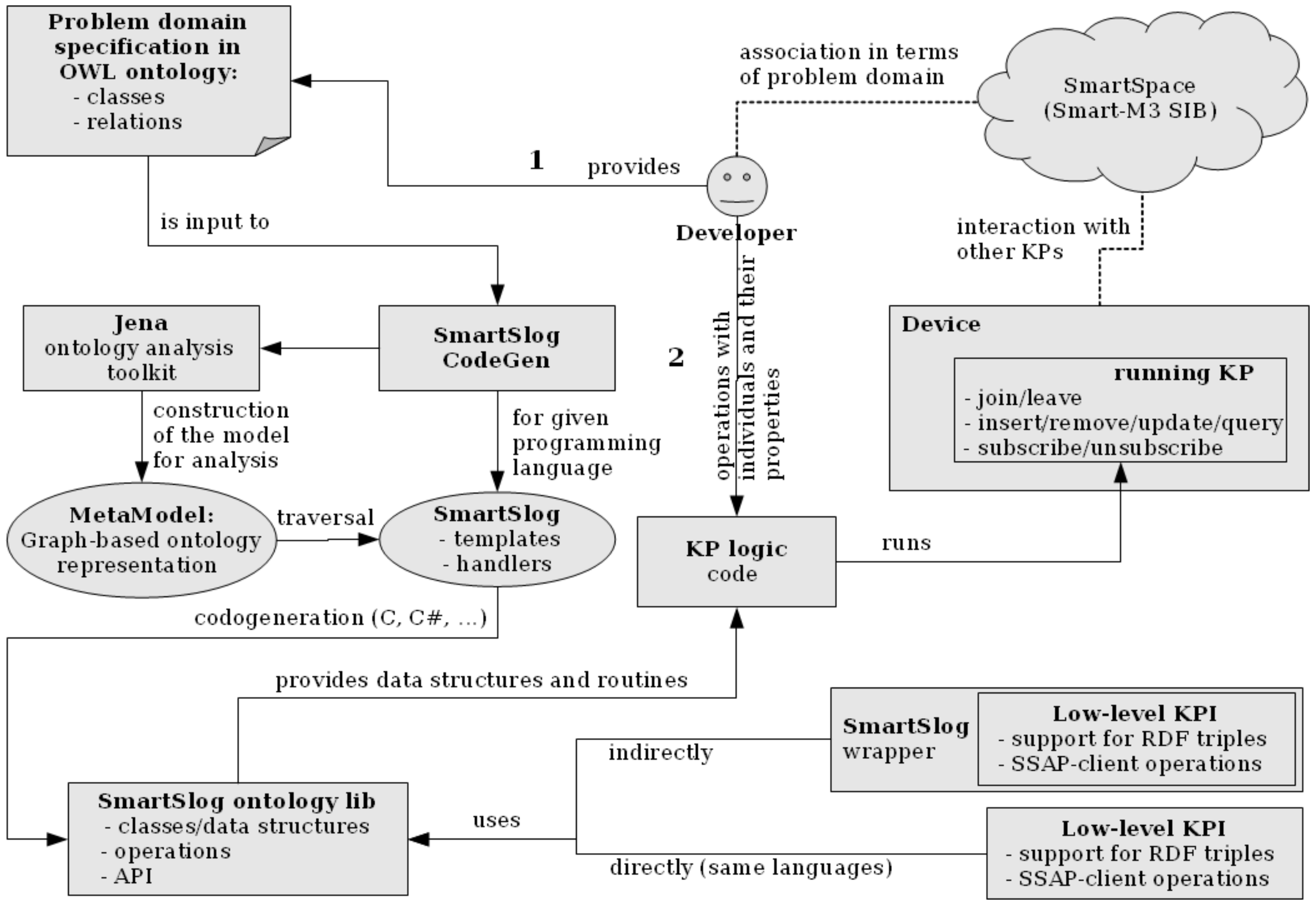
Problem domain specification in OWL ontology:
- classes
- relations
association in terms of problem domain
SmartSpace (Smart-M3 SIB)
1
provides
Developer
is input to
interaction with other KPs
operations with individuals and their properties
Jena ontology analysis toolkit
SmartSlog CodeGen
2
Device
running KP
- join/leave
- insert/remove/update/query
- subscribe/unsubscribe
construction of the model for analysis
for given programming language
MetaModel: Graph-based ontology representation
traversal
SmartSlog
- templates
- handlers
KP logic code
runs
codogeneration (C, C#, ...)
provides data structures and routines
SmartSlog wrapper
Low-level KPI
- support for RDF triples
- SSAP-client operations
indirectly
SmartSlog ontology lib
- classes/data structures
- operations
- API
uses
directly (same languages)
Low-level KPI
- support for RDF triples
- SSAP-client operations

Figure 3. Smart-M3 ontology library generation scheme.

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.

V. Library API

SmartSlog API provides generic API, both for ANSI C and C# variants of ontology library. Consequently, the SmartSlog API model covers two important classes of programming languages: procedural and object-oriented. In this section we focus on the API model of the ANSI C version.

The characteristic property of generic API is that names are independent on a concrete ontology. Classes, properties, and individuals appear as arguments in API functions. Datatype and object properties are treated similarly. One of the main retribution of this generic approach is the performance; run-time checking must be done for arguments.

In the ANSI C version, each ontology class, property, and individual is implemented as a C structure (types `property_t`, `class_t`, and `individual_t`). The API has generic functions that handle such data objects regardless of their real ontology content. Currently supported OWL constraints are class, datatypeproperty, objectproperty, domain, range, and cardinality. For example, a class knows all its superclasses, OWL one of classes, properties, and instances (individuals); the implementation is as follows.

```
typedef struct class_s {
  int rtti;          /* run-time type information */
  char *classtype;      /* type of class, name */
  list_t *superclasses; /* all superclasses */
  list_t *oneof;        /* class oneof value */
  list_t *properties;   /* all properties*/
  list_t *instances;    /* all individuals */
} class_t;
```

API functions are divided into two groups: for manipulating with local objects and for communicating with SIB. The first group (local) includes functions for

- Classes and individuals: creating data structures and manipulating with them locally.
- Properties: operations set/get, update, etc. in local store (also run-time checks for correctness, e.g., cardinality and property values).

For example, creating individual and setting its properties:

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

In this example, the definitions of `CLASS_WOMAN` and `PROPERTY_LNAME` are in the library ontology-dependent part for the ontology shown in Figure 4. (We used GrOwl tool [44]: classes are in blue rectangles, datatype properties are in brown ovals, object properties are in blue ovals.)

The second group (to/from smart space) has prefix "`ss_`" in function names and allows accessing smart space for

- Individuals: insertion, removal, and update.
- Properties: similarly to the local functions but the data are to/from smart space (it requires transformation to/from triples and calling the mediator library).
- Querying for individuals in smart space (existence, yes/no answer).
- Populating individuals from smart space by query or by subscription.

For example, inserting an individual and then updating some of its properties:

```
ss_insert_individual(aino);
     . . .
ss_update_property(aino,
    PROPERTY_LNAME, "Ericsson");
```

Subscription needs more discussion. In advance, a subscription container is created to add those individuals which to subscribe for. Optionally, the container contains the properties whose values are interested only. Then KP explicitly subscribes for selected properties of selected individuals.

Subscription is synchronous or asynchronous. The former case is simplest; KP is blocked waiting for updates. Even devices without thread support allow synchronous subscription. The latter case is implemented with a thread that controls updates from smart space and assigns them to the containers. KP is not blocked, and updates come in parallel.

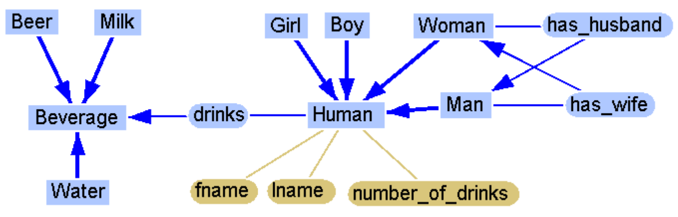

Figure 4. Ontology for humans and their drinks

Internally, communication with SIB leads to the composition/decomposition of high-level ontology entities from/to RDF triples and calling the low-level KPI for triple-based data exchange. To the best of our knowledge, SmartSlog is the only ontology library generator that uses KPI_Low as the low-level mediator KPI (see Table I). Since KPI_Low is oriented to low-performance devices, this design selection strengthens SmartSlog applicability in application development for this class of devices.

Compared with the Smart-M3 ontology to C-API generator, which provides similar communication primitives, SmartSlog has the following advantages. The Smart-M3 ontology to C-API generator depends on glib library, e.g., using list data structures. Low-performance devices do not support glib. In contrast, SmartSlog has no such requirements for underlying libraries. The Smart-M3 ontology to C-API generator does not allow asynchronous subscription important for smart space applications.

SmartSlog generic API is extended with 'knowledge patterns' for ontology-based filtering and search. A general model of a knowledge pattern will be considered later in Section VII-B; here we illustrate its representation for ANSI C. Each knowledge pattern is an `individual_t` structure and can be thought as an abstract individual where only a subset of properties is set. A knowledge pattern is either pattern-mask or pattern-request.

A pattern-mask is for selecting properties of a given a class or individual. It needs when a subset of properties is used, and the pattern includes only those properties. Then this pattern is applied to the given class or individual, e.g. for modest updating the properties. For example, let us update only the last name of "Aino" (see the ontology in Figure 4).

```
pattern_t *aino_p = new_pattern(CLASS_WOMAN, NULL);
add_check_property_pattern(aino_p, PROPERTY_LNAME,
        NULL, PATTERN_COND_NO);
ss_update_by_pattern(aino, aino_p);
```

As a result, only the last name value is transferred to smart space. Compared with `ss_update_property()` the benefit becomes obvious when KP needs to update several properties at once or it can form the property subset only in run-time. The same scheme works for population to transfer data modestly from smart space.

A pattern-request is for compact definition of search queries to smart space. A pattern is filled with those properties and values that characterize the individual to find. For example, let us find all men whose first name is "Timo" and wife's first name is "Aino".

```
pattern_t *timo_p = new_pattern(CLASS_MAN, NULL);
pattern_t *aino_p = new_pattern(CLASS_WOMAN, NULL);

add_check_property_pattern(timo_p, PROPERTY_FNAME,
        "Timo",  PATTERN_COND_NO);
add_check_property_pattern(aino_p, PROPERTY_FNAME,
        "Aino",  PATTERN_COND_NO);
add_check_property_pattern(timo_p, PROPERTY_HAS_WIFE,
        aino_p,  PATTERN_COND_NO);

timo_list = ss_get_individuals_by_pattern(timo_p);
```

In this example, two patterns ("Timo" and "Aino") and two properties (datatype "fname" and object "has_wife") form a subgraph. The SmartSlog library matches the subgraph to the smart space content. As a result, a list of available individuals is returned. Currently, searching leads to iterative triple exchange and matching on the local side. In future, it can be implemented on the top of SPARQL on the SIB side.

VI. Use Case Example

In this section, we show how SmartSlog can be used for constructing a simple Smart-M3 application in C. In spite of the simplicity, the example illustrates such SmartSlog features as knowledge patterns and subscriptions (synchronous and asynchronous). Both datatype and object properties are used.

Let Ericsson's family consist of Timo (husband) and Aino (wife). Timo likes drinking beer outside home. Aino has to control Timo's drinking via monitoring the amount of beer he has drunk already. If the amount is exceeding a certain bound (e.g., `MAX_LITRES_VALUE=3`) she notifies Timo by SMS that it's good time to come back to home.

The ontology for such personal human data was shown in Figure 4 above. When Timo starts drinking he associates his object property "drinks" with class "Beer". Then Timo keeps his drink counter "number_of_drinks" in smart space and regularly updates it. Aino can subscribe to this counter.

For messaging, the family uses the ontology shown in Figure 5. Aino sends SMS to notify Timo via smart space. Timo subscribes for SMS and checks each SMS he received for who sent it (by phone number). Hence Timo recognizes a notification SMS from his wife.

Given these two ontology files, SmartSlog generator produces files `drinkers.{c, h}`. Since the ontology includes more details than needed for this application, excessive classes and properties can be disabled in the final code by compiler preprocessor directives.

The KP code for Timo can be constructed with SmartSlog using the following scheme.

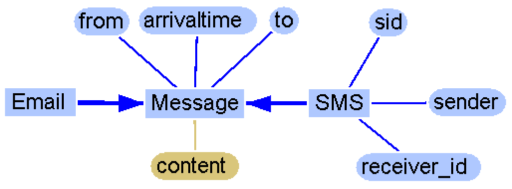

Figure 5. Ontology for messaging

1. Create Timo, set his properties, and insert the individual to the smart space.

```
individual_t *timo = new_individual(CLASS_MAN);
set_property(timo,PROPERTY_FNAME, "Timo");
          . . .
ss_insert_individual(timo);
```

2. Timo keeps his counter in the smart space.

```
individual_t *beer = new_individual(CLASS_BEER);
ss_set_property(timo, PROPERTY_DRINKS, beer);
```

3. Timo subscribes to SMS from Aino: creating an individual for SMS and filling the subscribe container. Then asynchronous (parameter "true") subscription starts.

```
individual_t *sms = new_individual(CLASS_SMS);
add_data_to_list(subscribed_prop_list,
     PROPERTY_FROM);
add_data_to_list(subscribed_prop_list,
     PROPERTY_TO);

subscription_container_t *container=
     new_subscription_container();
add_individual_to_subscribe(container,
     sms, subscribed_prop_list);

ss_subscribe_container(container, true);
```

4. Timo drinks, updates the counter, and checks SMS.

```
while(sms_notify(sms)) {
    amount += drink(timo);
    ss_update_property(timo,
         PROPERTY_NUMBER_OF_DRINKS, amount);
}
```

Similarly, the KP code for Aino is constructed as follows.

1. Aino searches Timo in the smart space by pattern.

```
individual_t *wife = new_individual(CLASS_WOMAN);
set_property(wife, PROPERTY_LNAME, "Ericsson");
set_property(wife, PROPERTY_FNAME, "Aino");

pattern_t *timo_p = new_pattern(CLASS_MAN, NULL);
add_check_property_pattern(timo_p, PROPERTY_FNAME,
        "Timo", PATTERN_COND_NO);
add_check_property_pattern(timo_p, PROPERTY_HAS_WIFE,
        aino_p, PATTERN_COND_NO);
     . . .

list = ss_get_individuals_by_pattern(timo_p);
individual_t *timo = ...;
```

2. Synchronous (parameter "false") subscription waits for Timo is starting to drink.

```
subscription_container_t *container=
     new_subscription_container();
add_individual_to_subscribe(container, timo,
                       properties);
ss_subscribe_container(container, false)

property_t *drinks = get_property(timo,
                       PROPERTY_DRINKS);
if (drinks==NULL) wait_subscribe(container);
```

3. Monitoring Timo's counter and checking the limit. Synchronous subscription is similar to the above.

```
/* Subscribing for Timo's counter */
```

```
    . . .

while(1) {
    amount = get_property(timo,
                 PROPERTY_NUMBER_OF_DRINKS);
    if (amount >= MAX_LITRES_VALUE) {
        /* Send SMS to Timo */
        break;
    }
    wait_subscribe(container_counter);
}
```

4. Create an individual for SMS and insert it to the smart space. Properties "to" and "from" are required.

```
individual_t *sms=new_individual(CLASS_SMS);
set_property(sms, PROPERTY_TO,
                  TIMO_PHONE_NUMBER);
set_property(sms, PROPERTY_FROM,
                  WIFE_PHONE_NUMBER);
ss_insert_individual(sms);
```

VII. Design Features

The same ontological and optimization methods, which improves the KP development and code efficiency, can be applied in the multilingual case. In this section, we discuss currently implemented features as well as recent design solutions.

A. Ontology composition

Smart space content can be structured with a set of different ontologies instead of a single big ontology. Figure 6 shows that in this case the generator produces a common library for several ontologies.

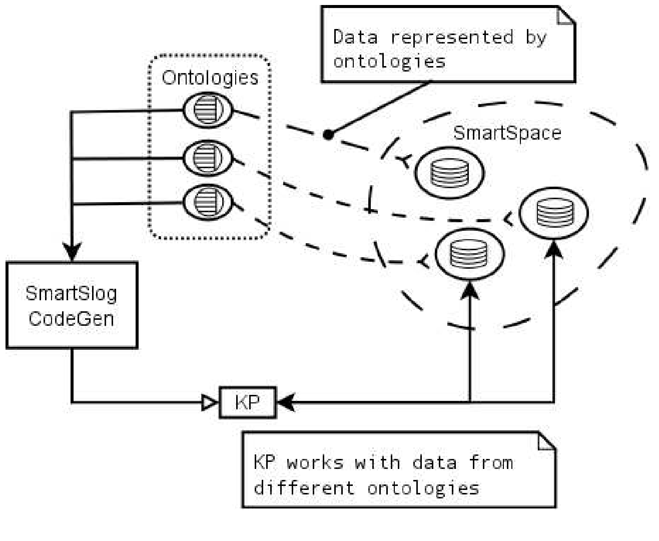
Data represented by ontologies
Ontologies
SmartSpace
SmartSlog CodeGen
KP
KP works with data from different ontologies

Figure 6. Ontology composition: a common ontology library.

1) Ontology integration: Integration is either complete or partial [45]. Complete integration means that the multiple ontologies are treated as all 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 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).

2) The same property in different ontologies: Figure 7 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 uses additional properties—bridge properties [46]. Values of such a multi-ontology property are stored in all corresponding parts of the smart space.

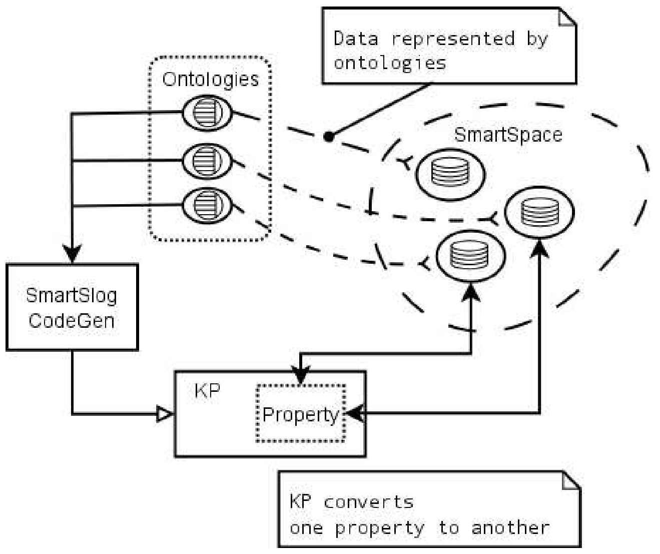
Data represented by ontologies
Ontologies
SmartSpace
SmartSlog CodeGen
KP
Property
KP converts one property to another

Figure 7. Ontology composition: different ontologies refer to the same property

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 property refers to a date then the bridge property converts the value to the format the KP requires.

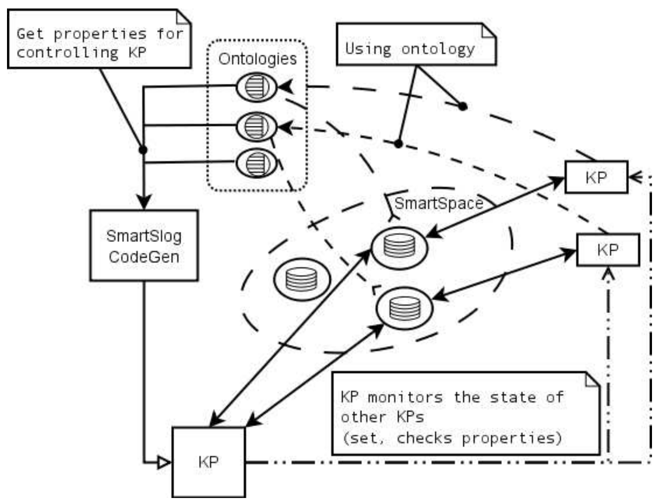
Get properties for controlling KP
Ontologies
Using ontology
SmartSlog CodeGen
SmartSpace
KP
KP
KP monitors the state of other KPs (set, checks properties)
KP

Figure 8. KP controller.

3) KP controller: There are smart applications where access to the smart space is controlled by a dedicated KP. Smirnov *et al.* [33] suggested a KP for resolving the problem of simultaneous access to the smart space content. Luukkala and Honkola [27] introduced the same idea for coordinated access to devices. Korzun *et al.* [47] employed a KP mediator for sharing the knowledge between two smart spaces: smart conference and blogosphere.

KP controller has to know several ontologies, see Figure 8. It controls ontology entities that are shared by 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 can support on–off states. Such a state is described differently in different ontologies. If the application needs to turn off several devices, this function can be implemented using a dedicated KP controller.

B. KP code optimization

SmartSlog cannot optimize its low-level mediator KPI, since the latter is an external library. Instead, SmartSlog optimizes local data structures, the (de)composition (to)from triples, and the way how the low-level mediator KPI is used. Clearly, these optimizations are also valid for computers with no hard performance restrictions.

1) Local data structures for OWL ontology entities: Each ontology entity is implemented as a C structure or a C# class of constant size. Consequently, for an ontology with N entities the SmartSlog ontology-dependent part of ontology library is of size $O(N)$.

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

Furthermore, if an object in the smart space has many properties, the KP can keep locally only a part of them. For example, in Figure 9, the object D is represented locally only with 3 datatype properties, regardless that D has also an object property in the smart space.

Note that when KP modifies an object locally the KP is responsible for timely updates. That is, in Figure 9, the object B has locally an extra object property compared with the primary instance of B in the smart space.

2) Local RDF triple repository: The Smart-M3 ontology C-to-API generator follows the straightforward and expensive strategy: its ontology library requires KP to maintain locally a cache of the whole smart space content. In contrast, SmartSlog does not intend to store any RDF triple for long time. OWL ontology entities are stored in own structures. When a triple is needed it is created locally or retrieved from the smart space. Then the local memory is freed immediately after the use of the triple.

3) Knowledge patterns: They provide a mechanism for searching and filtering the content: selecting those individuals that are of the current interest. To define which individuals the KP logic needs to process the developer constructs knowledge patterns. Then they are applied in filtering locally available objects or in searching and retrieving appropriate objects from the smart space.

A knowledge pattern can be thought as a graph of abstract ontology objects. Its nodes are objects augmented with datatype properties. Nodes are linked by object properties. It is similar to OWL ontology instance graph, but objects are abstract; they do not represent actual individuals. The developer specifies only a part of properties available for such objects in the ontology. For filtering these properties in the pattern are compared with properties of locally stored individuals. For searching these properties are used to find and retrieve individuals from the smart space. This way reduces the amount of data to keep, process, and transfer, even if concrete individuals have many properties.

Figure 9 shows an example. Applying the pattern for filtering with the abstract object A results in the individual A having been stored locally. Applying the same pattern for searching with the abstract object D results in the real object D having been shared in the smart space. In both cases, an application solves the matching problem for a subgraph (pattern with abstract objects) to a ontology instance graph (real objects in the local KP storage or the global smart space). Note that result can consists of several individuals that satisfy the pattern.

In fact, a pattern represents a semantic query. Currently Smart-M3 does not support SPARQL, which can be used for efficient implementation of the knowledge pattern mechanism. SmartSlog implements own algorithms that run on the KP side. In searching, it leads to transferring a lot of triples form the smart space with their subsequent iterative

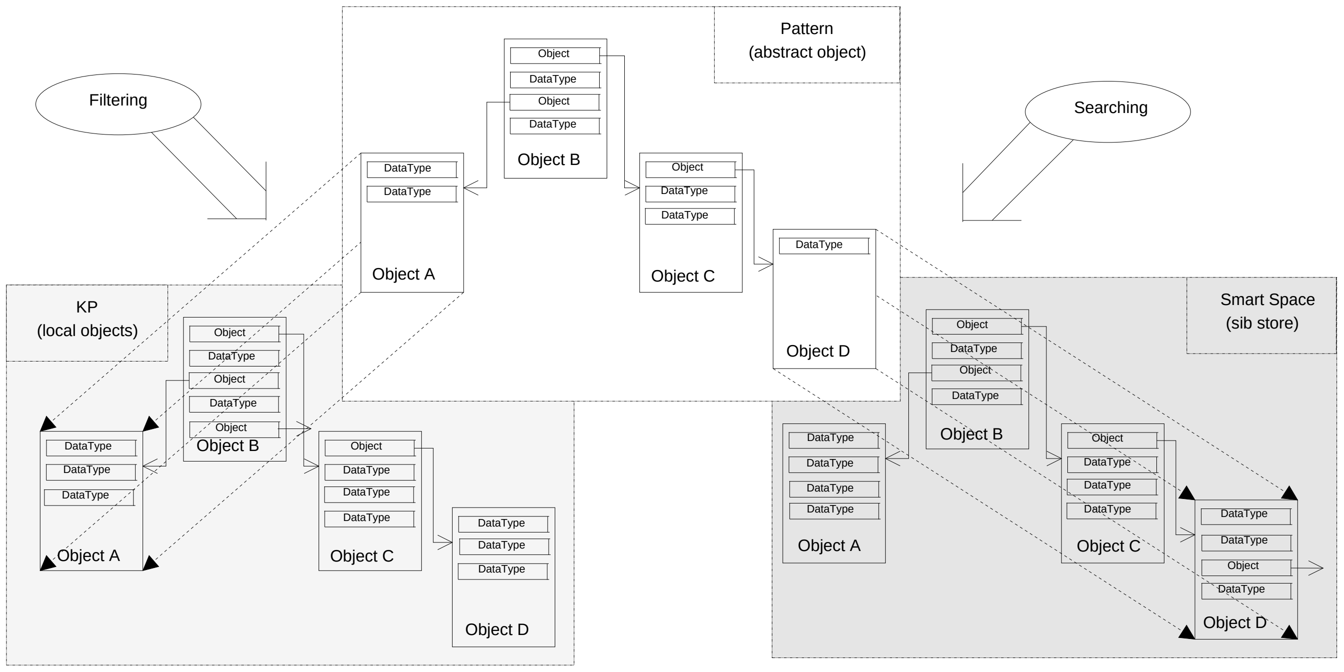

Figure 9. SmartSlog content representation and knowledge patterns for filtering and searching. Objects are stored globally in the smart space (SIB); KP caches them partially; knowledge patters allow efficient manipulations with both object stores.

processing.

Knowledge patterns allow defining an object by ontological class, UUID, and checked properties (properties that object should have). For more intelligent characterization, patterns can be extended to support unchecked properties (properties that object should not have) and conditional properties (with relations like $\leq, \geq, \leq, \geq$).

The possible points of further optimization are the following. Optimization of patterns as semantic queries since the performance of matching algorithms depends on the query representation [18]. For filtering, the access to properties can be optimized using hash tables.

4) Synchronization: SmartSlog supports both types of subscriptions: synchronous and asynchronous. The latter case requires threading. SmartSlog uses POSIX threads, available on many embedded systems [8]. Nevertheless, SmartSlog allows switching the asynchronous subscription off if the target device has no thread support.

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

Consider the example in Figure 10. 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 the synchronization both sides keep the same A'. Then A' is transformed to A'' on the SIB side (by some other participants) 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. The synchronization problem is to minimize these periods.

SmartSlog supports blocking and non-blocking synchronization (synchronous and asynchronous subscription). Both require setting explicitly the objects to synchronize. In some cases it can be difficult from the point of view of a KP programmer. 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 the smart space then the KP synchronizes the object in the non-blocking mode. As a result, the developer uses local objects assuming that they are always up to date. Since frequent synchronization leads to the high resource consumption (network throughput, SIB processing time) there should be options to control synchronization. For instance, the developer sets the data importance for better tradeoffs.

Finally, there should be a mechanism for determination of synchronization time moments. The following parameters affects this mechanism.

- Memory use: marking changed objects uses additional memory, synchronize when the memory threshold has been reached.
- Latest synchronization: synchronize when a time threshold has been reached after the latest synchronization. For instance, if a data item is changed rarely it is synchronized immediately after its change.
- Network load: if the network is overloaded then the

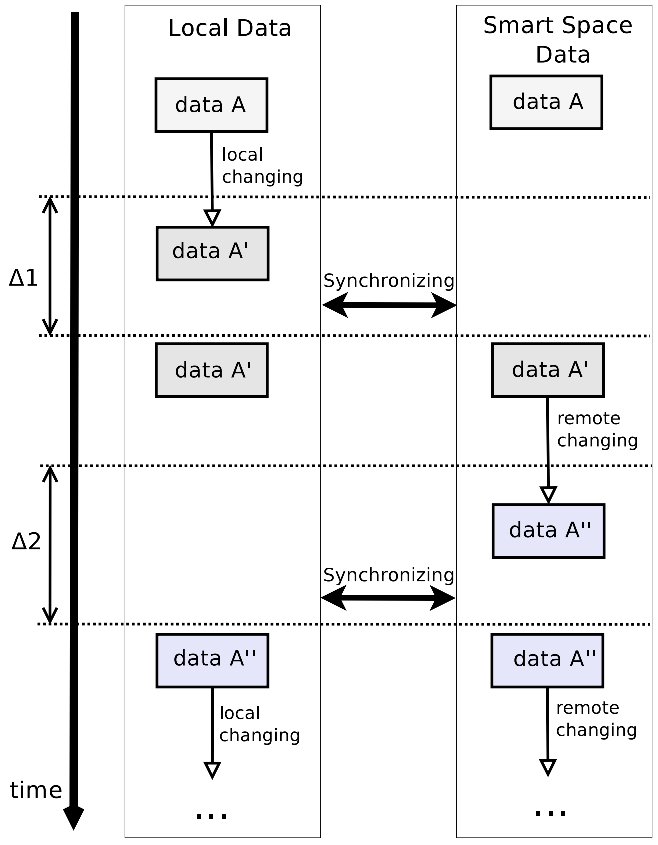

Figure 10. Synchronization problem. δs are periods of desynchronization.

synchronization rate is reduced;

- Device load: if the device is overloaded then the synchronization rate is reduced.

VIII. Conclusion

The addressed area of ontology library generation for multitude of devices is very important. The realization of the ubiquitous computing vision will by definition include a lot of heterogeneous and transiently available devices around us. Allowing these devices to easily share information with other devices and architectures, large or small, will be very important. SmartSlog is a tool that supports easy programming of such devices for participating them in Smart-M3 applications.

This paper contributed the design of SmartSlog with the advanced scheme of ontology library generation. A KP developer can choose among several programming languages for the ontology library. The current implementation supports ANSI C and C#. The operability was tested on Linux- and Windows- based platforms, including console (ANSI C), Qt (C/C++), and .NET (C#) environments.

The KP code is compact due to high-level ontology style. SmartSlog ontology libraries are portable due to the reduction of system dependencies. For low-performance devices the ontology library code follows ANSI C and POSIX standards. There are mechanisms for making ontology library code modest and optimizable to device capacity.

The SmartSlog design allows adopting advanced ontological and optimization methods. We showed that the ontology library 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 are topics of our ongoing research.

Acknowledgment

Authors would like to thank Open Innovations Association FRUCT for the provided support and R&D infrastructure. We would also like to thank Iurii Bogoiavlenskii, Vesa Luukkala, and Ronald Brown for providing feedback and guidance during the construction of the SmartSlog tool.

References

[1] 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.

[2] 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.

[3] "Smart-M3: Free development software downloads at SourceForge.net," Release 0.9.5beta, Dec. 2011. [Online]. Available: http://sourceforge.net/projects/smart-m3/

[4] 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.

[5] L. J. B. Nixon, E. Simperl, R. Krummenacher, and F. Martin-recuerda, "Tuplespace-based computing for the semantic web: A survey of the state-of-the-art," *Knowl. Eng. Rev.*, vol. 23, pp. 181–212, Jun. 2008.

[6] 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.

[7] "SmartSlog: free development software downloads at SourceForge.net," Dec. 2011. [Online]. Available: http://sourceforge.net/projects/smartslog/

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

[9] P. Liuha, A. Lappeteläinen, and J.-P. Soininen, "Smart objects for intelligent applications - first results made open," *ARTEMIS Magazine*, no. 5, pp. 27–29, Oct. 2009.

[10] "SOFIA project – smart objects for intelligent applications," Dec. 2011. [Online]. Available: http://www.sofia-project.eu/

[11] "Devices and interoperability ecosystem," Dec. 2011. [Online]. Available: http://www.diem.fi/

[12] 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.

[13] I. Oliver and S. Boldyrev, "Operations on spaces of information," in *Proc. IEEE Int'l Conf. Semantic Computing (ICSC '09)*. IEEE Computer Society, Sep. 2009, pp. 267–274.

[14] 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.

[15] D. Gelernter, "Generative communication in linda," *ACM Trans. Program. Lang. Syst.*, vol. 7, pp. 80–112, Jan. 1985.

[16] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, pp. 34–43, May 2001.

[17] D. Fensel, "Triple-space computing: Semantic web services based on persistent publication of information," in *Proc. IFIP Int'l Conf. Intelligence in Communication Systems (INTELLCOMM 2004)*, ser. LNCS 3283. Springer, Nov. 2004, pp. 43–53.

[18] J. Euzenat and P. Shvaiko, *Ontology matching*. Heidelberg (DE): Springer-Verlag, 2007.

[19] E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF," W3C Recommendation, Jan. 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

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

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

[22] M. Weiser, "The computer for the twenty-first century," *Scientific American*, vol. 265, no. 3, pp. 94–104, 1991.

[23] D. Khushraj, O. Lassila, and T. W. Finin, "sTuples: Semantic tuple spaces," in *Proc. 1st Annual Int'l Conf. Mobile and Ubiquitous Systems (MobiQuitous 2004)*. IEEE Computer Society, 2004, pp. 268–277.

[24] R. Krummenacher, J. Kopecký, and T. Strang, "Sharing context information in semantic spaces," in *Proc. OTM 2005 Workshops on the Move to Meaningful Internet Systems 2005*, ser. LNCS 3762. Springer, 2005, pp. 229–232.

[25] A. Gómez-Goiri and D. López-De-Ipiña, "A triple space-based semantic distributed middleware for internet of things," in *Proc. 10th Int'l Conf. Current trends in web engineering (ICWE'10)*. Springer-Verlag, 2010, pp. 447–458.

[26] J. Honkola, H. Laine, R. Brown, and I. Oliver, "Cross-domain interoperability: A case study," in *Proc. 9th Int'l Conf. Next Generation Wired/Wireless Networking (NEW2AN'09) and 2nd Conf. Smart Spaces (ruSMART'09)*, ser. LNCS 5764. Springer-Verlag, 2009, pp. 22–31.

[27] V. Luukkala and J. Honkola, "Integration of an answer set engine to smart-m3," in *Proc. 3rd Conf. Smart Spaces (ruSMART'10) and 10th Int'l Conf. Next Generation Wired/Wireless Networking (NEW2AN'10)*. Springer-Verlag, 2010, pp. 92–101.

[28] J. Suomalainen, P. Hyttinen, and P. Tarvainen, "Secure information sharing between heterogeneous embedded devices," in *Proc. 4th European Conf. Software Architecture (ECSA '10): Companion Volume*. ACM, 2010, pp. 205–212.

[29] A. Koren and A. Buntakov, "Access control in personal localized semantic information spaces," in *Proc. 3rd 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. 84–91.

[30] A. D'Elia, D. Manzaroli, J. Honkola, and T. S. Cinotti, "Access control at triple level: Specification and enforcement of a simple RDF model to support concurrent applications in smart environments," in *Proc. 11th Int'l Conf. Next Generation Wired/Wireless Networking (NEW2AN'11) and 4th Conf. Smart Spaces (ruSMART'11)*. Springer-Verlag, 2011.

[31] S. Boldyrev, I. Oliver, and J. Honkola, "A mechanism for managing and distributing information and queries in a smart space environment," *UBICC Journal*, Jul 2009.

[32] I. Oliver, E. Nuutila, and S. Törmä, "Context gathering in meetings: Business processes meet the agents and the semantic web," in *The 4th Int'l Workshop on Technologies for Context-Aware Business Process Management (TCoB 2009) within Proc. Joint Workshop on Advanced Technologies and Techniques for Enterprise Information Systems*. INSTICC Press, May 2009.

[33] 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. Next Generation Wired/Wireless Networking (NEW2AN'09) and 2nd Conf. Smart Spaces (ruSMART'09)*, ser. LNCS 5764. Springer-Verlag, 2009, pp. 42–51.

[34] D. Korzun, I. Galov, A. Kashevnik, K. Krinkin, and Y. Korolev, "Integration of Smart-M3 applications: Blogging in smart conference," in *Proc. 4th Conf. Smart Spaces (ruSMART'11) and 11th Int'l Conf. Next Generation Wired/Wireless Networking (NEW2AN'11)*.

[35] K. Främling, A. Kaustell, I. Oliver, J. Honkola, and J. Nyman, "Sharing building information with smart-m3," *International Journal on Advances in Intelligent Systems*, vol. 3, no. 3&4, pp. 347–357, 2010.

[36] S. Balandin, I. Oliver, and S. Boldyrev, "Distributed architecture of a professional social network on top of M3 smart space solution made in PCs and mobile devices friendly manner," in *Proc. 3rd Int'l Conf. Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2009)*. IEEE Computer Society, 2009, pp. 318–323.

[37] D. Zaiceva, I. Galov, and D. Korzun, "A blogging application for smart spaces," in *Proc. 9th Conf. of Open Innovations Framework Program FRUCT and 1st Regional MeeGo Summit Russia–Finland*, Apr. 2011, pp. 154–163.

[38] J. F. Gómez-Pimpollo and R. Otaolea, "Smart objects for intelligent applications – ADK," in *Proc. 2010 IEEE Symp. Visual Languages and Human-Centric Computing (VL/HCC)*, Sep 2010, pp. 267–268.

[39] "RDFAlchemy: an ORM (Object RDF Mapper) for semantic web users," Dec. 2011. [Online]. Available: http://www.openvest.com/trac/wiki/RDFAlchemy

[40] B. Lavender, "Spira: A linked data ORM for Ruby," Dec. 2011. [Online]. Available: http://blog.datagraph.org/2010/05/spira

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

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

[43] "Jena: Java toolkit for developing semantic web applications based on W3C recommendations for RDF and OWL," Dec. 2011. [Online]. Available: http://jena.sourceforge.net/,http://incubator.apache.org/jena/

[44] S. Krivov, R. Williams, and F. Villa, "GrOWL: A tool for visualization and editing of OWL ontologies," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 54–57, 2007.

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

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

[47] 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 and 1st Regional MeeGo Summit Russia–Finland*, Apr. 2011, pp. 63–73.