Smart-M3 Information Sharing Platform

Jukka Honkola, Hannu Laine, Ronald Brown, Olli Tyrkkö
Nokia Research Center
Helsinki, Finland
{jukka.honkola, hannu.e.laine, ronald.brown, olli.tyrkko}@nokia.com

***Abstract*—We describe the concept, architecture and key design decisions of Smart-M3 interoperability platform. The platform is based on the ideas of space-based information sharing and semantic web ideas about information representation and ontologies. The interoperability platform has been used as the basis for multiple case studies.**

Keywords: semantic interoperability, semantic web, smart spaces, RDF, ontologies

I. Introduction

The ongoing digital convergence has resulted in multitude of devices that have considerable computing power and are capable of communicating with the external world in various ways, often wirelessly. Furthermore, the information contained in these devices may be interesting also to other devices. Also, the number of nomadic devices such as smart phones has increased dramatically in recent years.

The potential of these devices has also greatly increased the interest in getting these devices to interoperate. Currently, the standards for interoperability have been mostly created for single domains, such as UPnP for home entertainment, or are controlled by a single company, such as the Apple ecosystem.

The domain specific interoperability standards pose considerable challenges for nomadic devices which ideally should be able to interoperate with whatever devices are in the locality at any given time. A nomadic device will currently have to implement several different standards to be able to participate in the different domains. Furthermore, the existing standards often target specific use cases rather than attempt to specify a general framework for interoperability.

One attempt to define and implement a generic interoperability framework is the semantic web [7]. The semantic web aims to provide machine understandable semantic information about the World Wide Web in order to allow automating many tasks that the web is currently used for manually. Eventually this would result in one "giant global graph" describing the resources of the web in RDF [17] according to Ontologies defined in OWL [18].

However, the web is not a good mechanism to share the rapidly changing, dynamic local information about the immediate environment of a device. Thus, we propose Smart-M3 as an information interoperability approach for devices to easily share and access local semantic information, while also allowing access to the locally relevant parts of the "giant global graph" to be available. The information is represented by using same mechanisms as in semantic web, thus allowing easy exchange of global and local information. The RDF representation also allows extremely easy linking of data also between different ontologies, thus making cross-domain interoperability straightforward.

Furthermore, as interoperability is based on the exchange of information expressed according to some ontology, we aim to lift interoperability standardization from use case standardization to ontology standardization. Standardizing[1] an ontology allows an indefinite set of use cases to be implemented, keeping the door open for future innovation without the need to commit to lengthy and uncertain standardization process.

Smart-M3 is publicly available under BSD open source license [8] and thus suitable both for research purposes and industrial use.

II. Overview

The Smart-M3 interoperability platform is based on a blackboard architectural model and the ideas of space-based computing. It consists of two main components: semantic information broker (SIB) and knowledge processor (KP). A smart space is defined as a named search extent of information, where the information is stored in one or more SIBs. A domain model describing the central concepts of a smart space is presented in Figure 1: Smart-M3 domain model. In the simplest case, one SIB will store all information in a smart space, but there is a possibility of connecting multiple SIBs to make up a smart space. The SIBs making up a smart space will be connected with a protocol that provides distributed deductive closure [3]. Thus, any KP sees the same information content in the smart space regardless of the SIB it connects to[2].

The information in the smart space is stored as a RDF graph, usually according to some defined ontology. The use of any specific ontology is not mandated, however. The KPs may modify and query the information using the insert, remove, update, query and subscribe operations provided by SIB.

[1] The standard could be also a de-facto standard not coming from an official standardization body.

[2] The distributed deductive closure protocol is not implemented in the current Smart-M3 implementation.

The communication between KPs and SIB may be implemented by using any SOA style service network such as the Network on Terminal architecture [16], or by using a suitable transport protocol such as XMPP or TCP/IP. This approach allows using already deployed communication mechanisms and protects previous investments. Furthermore, it also means that KPs can connect to smart spaces by using the mechanism most suited to them.

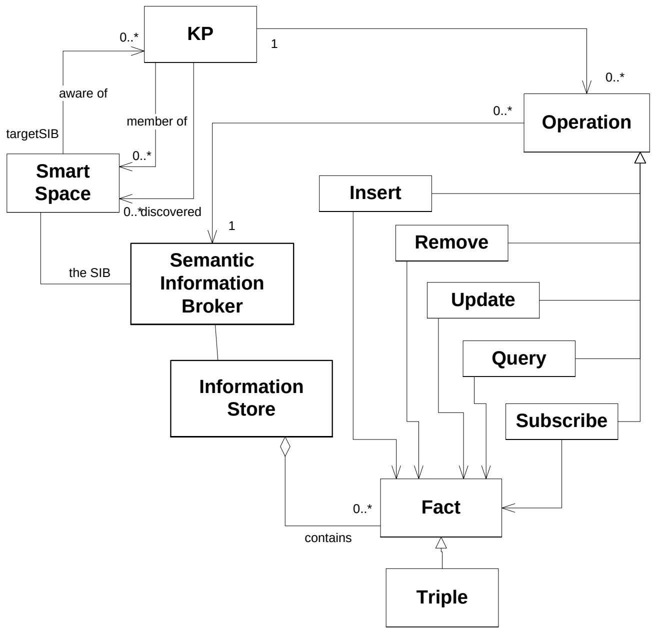

Figure 1: Smart-M3 domain model

Figure 2 shows a logical architecture with a SIB providing the SIB service on three different communication mechanisms, and two KP implementations, one with the SIB service accessible with communication mechanisms X and Y, and one with communication mechanism Z. The SIB service should implement all the SSAP operations described in Section IV. The developers of the application logic of KPs use a knowledge processor interface (KPI) to access information in smart space. The KPI may provide means to call the basic operations of insert, remove, etc, or may provide an ontology specific API to allow developers to program the application logic using the concepts of the chosen ontology. In addition, there may be ontology libraries providing the ontology concepts for the developers. These ontology libraries may be either manually or automatically generated.

Figure 3 shows the overview of the current Smart-M3 implementation architecture. The SIB daemon handles the information access and storage. Two communication mechanisms, TCP/IP and NoTA are handled by separate processes connected to SIB daemon over DBus. There are three KPIs implemented. The C/GLib and C++/Qt share common infrastructure whose purpose is to allow on-the-fly insertion and removal of communication modules. The KP Daemon process hides the different communication mechanisms, allowing the KPIs to handle only the SSAP communication with the SIB, without having to know the underlying communication mechanism. The Python KPI is a standalone pure-python library. There are also design time ontology library generators for generating ontology APIs for C/GLib, Python [9] and ANSI C [11] KPIs.

The principles guiding the design of Smart-M3 are simplicity, extensibility and being agnostic to the used communication mechanisms. The simplicity ensures scalability for small devices, and also for large number of users, while the extensibility makes it possible to tailor the implementation easily to uses where the standard functionality is not sufficient. Furthermore, by not dictating a specific communication mechanism the Smart-M3 should be easy to deploy on top of any existing infrastructure.

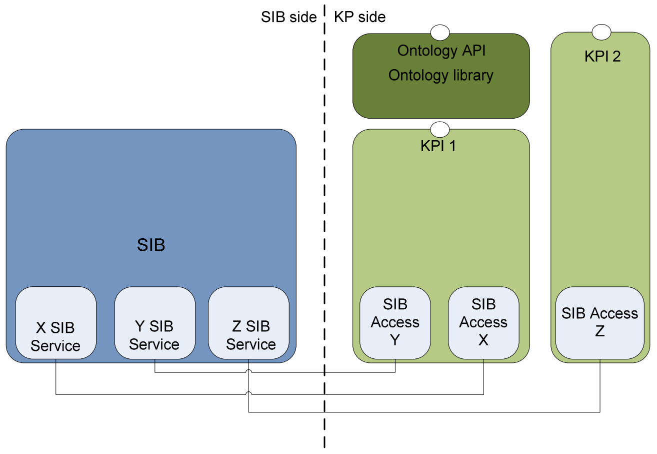

Figure 2: Smart-M3 logical architecture

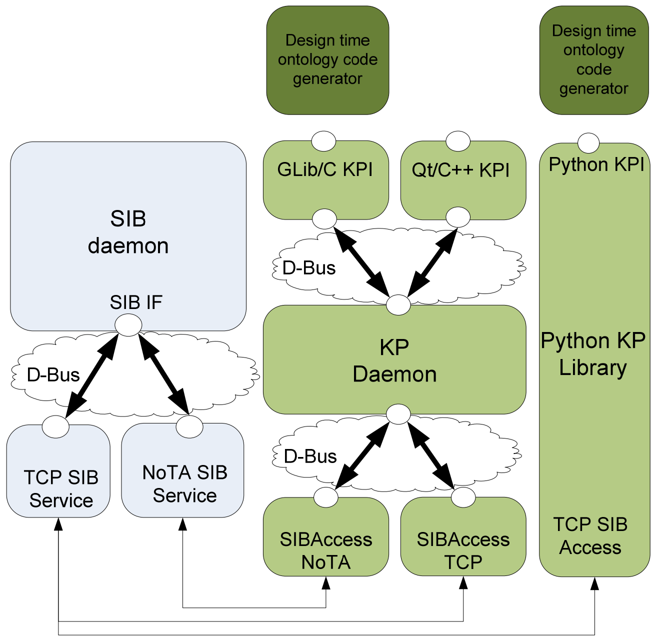

Figure 3: Smart-M3 current implementation architecture

III. Notion of Application

The notion of application in a smart space differs radically from the concept of traditional application. Instead of a monolithic application running on a single screen, the smart space applications are better seen as scenarios that can be executed to meet the goals of the user. The scenario emerges from the observable actions taken by knowledge processors based on the information in the smart space, and also from the use of available services. The scenarios may also be transient: the scenario will change as the participating KPs join and leave the smart space, and also as services become available or unavailable.

One of the targets of Smart-M3 has been to make combining the scenarios easy. This is achieved by the loose coupling between the KPs, as they only communicate by modifying and querying the information in the smart space[3]. Thus, any effect that appearance or disappearance of KPs may have on the rest of the environment is limited to the information seen in the smart space.

A KP understands its own, non-exclusive set of information[4] as illustrated in Figure 4. Overlap of the sets of information understood by KPs is an essential precondition for achieving interoperability. Otherwise the KPs will never see each others actions.

A. *Case Studies using Smart-M3*

Smart-M3 has been used as a basis for several smart space application case studies.

A case study of cross domain interoperability is described in [1]. The scenario involved a wellness domain represented by SportsTracker application, home entertainment domain represented by music streaming using a UPnP service, gaming domain represented by SuperTux game, and telecom domain represented by a phone call observer application. All domains share information using a M3 smart space, resulting in improved user experience compared to the case where the components operate independently. For example, when a call is received, the game and music player see the information that a call is ongoing in the smart space and can take appropriate action, in this case pause the music and game. When the call ends, music continues automatically. Furthermore, the played music changes according to the state of the game, for example, when the player loses lives, the music changes appropriately. Finally, the game may award extra lives if the player has been exercising lately.

Other case studies include a building automation case [4], a smart meeting room case [5], a home sensor network case [12], and a health care case developed by University of Bologna.

[3] KPs may communicate over other shared communication mechanisms. However, this is outside the scope of Smart-M3.

[4] This set of information may be thought as the ontology of the KP. It is only rarely defined formally, and is usually an implicit but at times useful concept.

In the building automation case, information from the installed sensors and systems is shared for others by using Smart-M3, allowing also devices that do not implement the oBIX or other building automation protocols to access the information.

In the smart meeting room case, Smart-M3 is used to coordinate access to the resources of a meeting room, such as a projector for presentations.

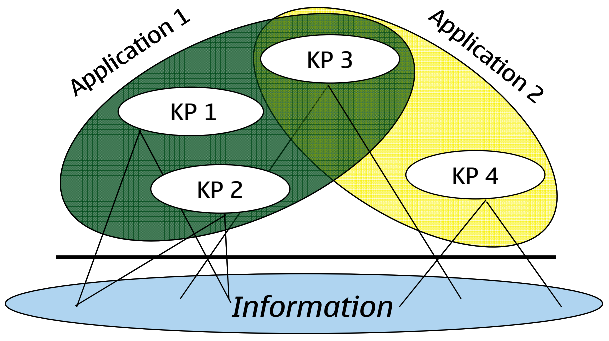

Figure 4: The relation of information, KPs and smart space application

The home sensor network case is remarkable in that all the hardware used is commercially available. The demo shows input from several sensors on multiple devices, such as Apple IPhone, Nokia N810 and laptop.

The health care case uses information from temperature and humidity sensor and patient's heart rate monitor to determine if the external conditions may be hazardous for the patient.

Furthermore, a demonstration showing a combination of the home sensor network and healthcare demos where the information from the sensors used in home sensor network demo was used in the healthcare system, illustrating the ease of creating mash-up applications, has been created.

IV. The Smart Space Access Protocol

The Smart Space Access Protocol (SSAP) is the protocol that the KPs use to access a SIB. It has seven operations as listed in Table 1: SSAP operations. The operations are abstract, that is, they are defined in terms of their parameters and the actions that SIB and KP should take. The operations may be encoded in different ways, for example in XML or JSON.

The protocol is session-based, assuming that the KP wanting to join the smart space will first have to join the smart space with the *join* operation. The KP will provide its credentials in the *join* message, and the SIB receiving the message will examine the credentials and decide whether the KP can join. After joining, the KP can perform the other operations.

The SSAP is the main integration point of the Smart-M3 architecture. All implementations of SIBs and KPs should support all SSAP operations. This will guarantee interoperability across different implementations.

In addition to the currently specified and implemented operations, there is work ongoing to add further operations. In particular, a conditional update operation is seen as a useful addition, as it would provide the means to implement ontological concurrency control mechanisms for information in M3 spaces.

Table 1: SSAP operations

Operation name	Description of operation
Join	Joins a KP to a named M3 space if the credentials match what is required.
Leave	Leaves a named M3 space. No more operations may be performed after a leave until a join.
Insert	Atomically inserts a graph into SIB.
Remove	Atomically removes a graph from SIB.
Update	Atomically updates a graph in SIB. This is an atomic combination of Delete and Insert operations, where Delete is performed first.
Query	Queries for information in the SIB using one of the supported query mechanisms.
Subscribe	Sets up a subscription (persistent query) in SIB. The KP is notified when the subscription results change.
Unsubscribe	Cancel a subscription.

V. The Semantic Information Broker

The Semantic Information Broker (SIB) is the component where the semantic information provided by KPs is stored. Ideally, any SIB should be accessible from many SOA systems, enabling a truly cross-domain information exchange and the resulting possibilities for interoperability.

The SIB consists of five layers as shown in Figure 5: SIB internal architecture.:

1. Transport layer
2. Operation handling layer
3. Graph operations layer
4. Triple operations layer
5. Persistent storage layer

The transport layer consists of one or more transport processes providing the SIB service to different service architectures and networks. The transport processes are connected to the operation handling layer by DBus, making it possible to add or remove transports at run-time.

The request handling layer handles the SSAP operations, with each operation running autonomously in its own thread. While the heavy use of threads causes some overhead, the resulting clarity of the program code was deemed more important.

The graph operations layer handles the insertion, removal and querying of graphs from the RDF store as requested by the request handling layer. The layer runs in a single thread which schedules and executes the requests from the threads handling the SSAP operations. The communication between the SSAP operations threads is handled by using asynchronous queues.

The triple operations layer is currently implemented by using Piglet RDF store. Piglet contains query facilities for SPARQL select queries, Wilbur query language (WQL) [8] queries and triple pattern queries, as well as operations for inserting and removing triples from the RDF store. The persistent store used by Piglet is SQLite.

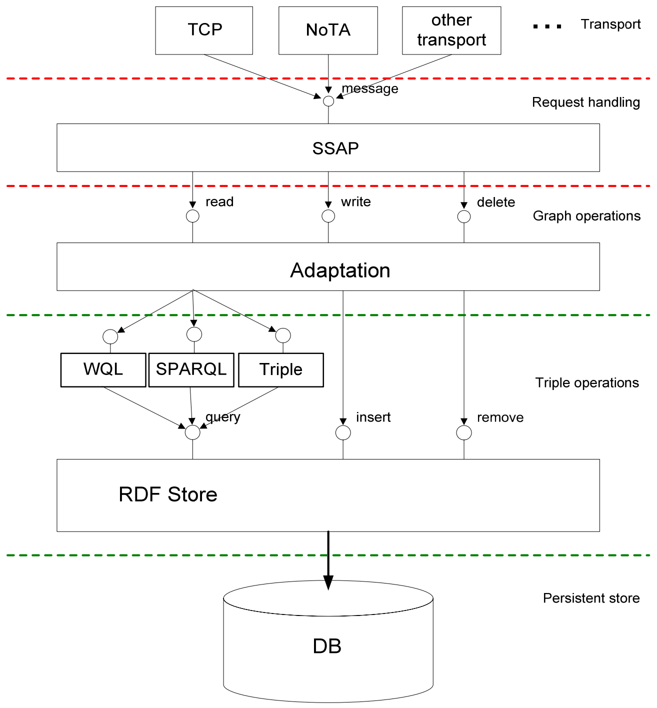

Figure 5: SIB internal architecture.

The triple operations layer is not tied to any specific RDF store, and any RDF store supporting the basic operations of read, write and delete may be substituted in the place of Piglet. However, changing the RDF store will require changing the code in the graph operations layer to adapt to the concrete interface provided by the new RDF store.

The SIB in the basic configuration provides reasoning capability for certain rdfs and owl properties, such as rdf:type, rdfs:subClassOf and owl:sameAs, in the WQL query engine. For example, when querying about the type of an instance, you get as a response the immediate type and all supertypes.

However, we plan to include a plugin interface for custom reasoners that can be attached to the SIB in order to perform reasoning according to domain-specific rules. The interface offered to these reasoners will resemble the SSAP interface offered to KPs, with possibly some extensions such as the possibility to lock the RDF store for short durations.

The reasoners may be written in any suitable programming language, or they may use a rules engine, such as the smodels tool [6]. For example, the ssls tool [10] allows the user to use rules written for smodels to modify the content in the smart space[5].

[5] The ssls is a KP providing a command-line interface to the information in the smart space, and thus is outside the SIB. However, a similar setup using the plugin interface to the SIB is easily conceivable.

VI. Knowledge Processors

Knowledge processors are the active parts of a system using Smart-M3. They provide the information, may modify and query it and also take externally observable actions based on the information that they see in smart space. Figure 6 presents an overview of a generic KP architecture.

A KP is typically created by using one of the available KPIs or ontology APIs suitable for the platform that it is going to run. The KPI that different implementations provide for the developer can be at any abstraction level, though in practice the functions or method calls available in the KPI will closely resemble the SSAP operations. However, it should be noted that SSAP and KPI are different concepts and the SSAP is the integration point that different implementations are required to adhere to.

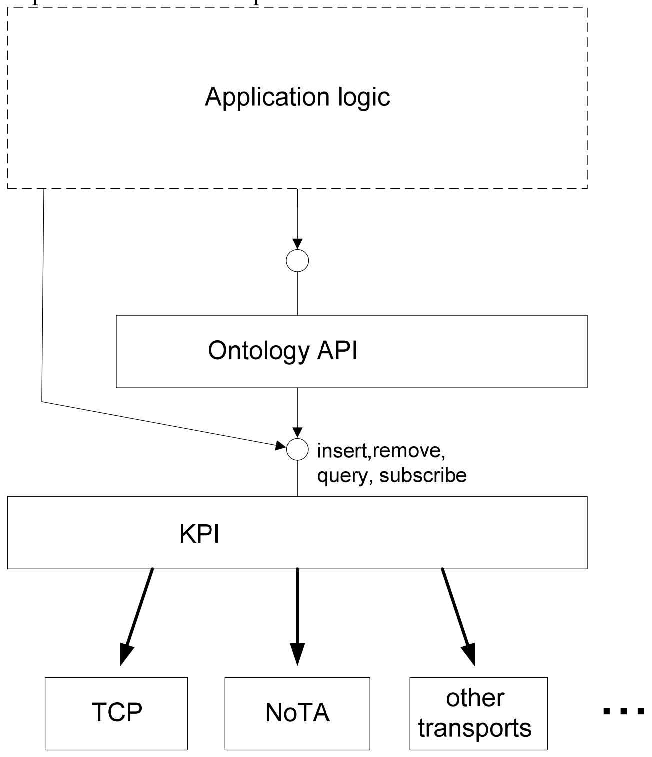

Figure 6: Overall KP architecture

A. *Knowledge Processor Interface Implementations*

The Smart-M3 release contains implementations of KPIs for GLib/C, Qt/C++ and Python languages. In addition to the three KPIs published in Smart-M3 release, there are several other KPIs: an ANSI C KPI targeted towards resource constrained embedded devices, C# based KPI for .NET environment, and Java KPI.

The GLib/C and Qt/C++ KPIs are integrated to GLib and Qt main loops, respectively. This integration allows the creation of GUI applications that support smart spaces.

The python KPI is a standalone, pure python library. Python provides an environment for rapid development, and as such allows also KPs to be created rapidly. However, python is also a mature programming environment and as such its use is not restricted to prototype programs. Furthermore, python interpreters are available on a variety of platforms, and thus allow easy porting of Python KPs.

B. *Ontology API*

Ontology APIs allow a developer to program using ontology level concepts instead of working with SSAP and RDF. Thus, the existence of ontology APIs is critical in order to attract developers to create smart space applications.

Ontology APIs may be created manually, especially for relatively static and small ontologies. However, for ontologies that change or are complex, automatic generation is a more suitable solution.

Ontologies described using owl may be mapped to object-oriented (OO) class structures, even though the concept of type in description logic and class in OO world are not the same. An ontology API generator will thus first traverse the ontology description, possibly inferring some additional properties, and then create a class model of the ontology. From this class model, code can then be generated to map the concepts defined in the ontology to programming language operations which will then synchronize the information with a smart space.

Currently, there is a generic ontology API backend available, and code generators targeting GLib/C, Python [9] and ANSI C KPIs [11] are available.

VII. Conclusions

The Smart-M3 is an interoperability platform operating on principles of space-based information exchange. We have defined an architecture consisting of knowledge processors and a semantic information broker. The semantic information broker stores and makes available information inserted to it by knowledge processors. The communication mechanism between knowledge processors and semantic information broker is called the smart space access protocol, or SSAP. The SSAP is the main integration point in the architecture, and any knowledge processor or semantic information processor implementing it can participate in the system.

The knowledge processors co-operating in different scenarios are extremely loosely coupled in the Smart-M3 world, though they may have dependencies outside the Smart-M3 platform. Thus, the appearance or disappearance of knowledge processors will have little effect on other knowledge processors.

By defining the concepts of domains in domain-specific ontologies and standardizing these, we allow an indefinite set of interoperability use cases to be implemented. This removes the need for standardizing separate use cases and thus lowers the barrier of entry and allows also smaller companies to innovate. Naturally, as the ontologies are unlikely to remain static, this also calls for suitable processes for governing ontologies.

The Smart-M3 platform implementation [9] is available in Sourceforge and it is licensed under BSD license, making the Smart-M3 easy to take into use also in projects where the participants do not want to share their code with others.

The Smart-M3 is currently being used as the baseline for Sofia interoperability platform in Sofia/Artemis project [14] funded by European Commission. It is also being used for similar purpose in a DIEM project [15] funded by Finnish national authorities. In addition to these two publicly funded research projects, Smart-M3 is being used as a platform for smart application research in FRUCT [13] collaboration framework.

ACKNOWLEDGMENT

This work was supported by SOFIA (Smart Objects For Intelligent Applications) project and funded by Tekes (the Finnish Funding Agency for Technology and Innovation) and the European Commission, and DIEM (Devices and Interoperability Ecosystem) funded by Tekes.

REFERENCES

[1] J. Honkola, H. Laine, R. Brown and I. Oliver, “Cross-domain interoperability: a case study,” in proceedings of Smart Spaces and Next Generation Wired/Wireless Networking, LNCS, Volume 5764/2009, pp. 22-28.

[2] I. Oliver and J. Honkola, “Personal semantic web through a space based computing environment,” in second IEEE International Conference on Semantic Computing, August 4-7, 2008, Santa Clara, CA, USA. IEEE, 2008.

[3] S. Boldyrev, I. Oliver and J. Honkola, ”A mechanism for managing and distributing information and queries in a smart space environment,” in MDMD2009, May 6-7, 2009, Milan, Italy.

[4] K. Främling, I. Oliver, J. Honkola, and J.Nyman, “Smart spaces for ubiquitously smart buildings,” in UBICOMM 2009, October 16-19, Sliema, Malta

[5] A. Smirnov, A. Kashevnik, N. Shilov, I. Oliver, S. Balandin, and S. Boldyrev, “Anonymous Agent Coordination in Smart Spaces: State-of-the-Art,” in proceedings of Smart Spaces and Next Generation Wired/Wireless Networking, LNCS, Volume 5764/2009, pp. 42-51.

[6] P. Simons, “Extending and implementing the stable model semantics,” Doctoral dissertation, Research report 58, Helsinki University of Technology, Helsinki, Finland, April 2000

[7] T. Berners-Lee, J. Hendler and O. Lassila, “The semantic web,” Scientific American, May 2001

[8] O. Lassila, “Programming Semantic Web Applications: A Synthesis of Knowledge Representation and Semi-Structured Data,” Doctoral dissertation, ISBN 978-951-22-8985-1, Helsinki, Finland, October 2007

[9] Smart-M3 release. http://sourceforge.net/projects/smart-m3/. Referenced March 4th 2010.

[10] Ssls release. http://sourceforge.net/projects/ssls/. Referenced March 4th 2010.

[11] SmartSlog release. http://sourceforge.net/projects/smartslog/. Referenced March 4th 2010.

[12] OpenM3 demonstration release. http://sourceforge.net/projects/smart-m3/files/OpenM3.tar.gz/download. Referenced March 4th 2010.

[13] Finnish-Russian University Cooperation in Telecommunications, http://www.fruct.org/. Referenced in 4th March 2010.

[14] Sofia/Artemis project. http://www.sofia-project.eu. Referenced March 4th 2010.

[15] Device and interoperability ecosystem. http://www.tivit.fi/en/device. Referenced March 4th 2010.

[16] Network on Terminal Architecture (NoTA), http://www.notaworld.org. Referenced 4th March 2010.

[17] Resource Description Framework. http://www.w3.org/RDF/. Referenced March 4th 2010.

[18] OWL web ontology language. http://www.w3.org/standards/techs/owl. Referenced March 4th 2010.