Received June 12, 2014, accepted June 27, 2014, date of publication August 18, 2014, date of current version August 27, 2014.

Digital Object Identifier 10.1109/ACCESS.2014.2347992

Semantic Interoperability Architecture for Pervasive Computing and Internet of Things

JUSSI KILJANDER[1], ALFREDO D'ELIA[2], FRANCESCO MORANDI[2], PASI HYTTINEN[3], JANNE TAKALO-MATTILA[1], ARTO YLISAUKKO-OJA[1], JUHA-PEKKA SOININEN[1], (Member, IEEE), AND TULLIO SALMON CINOTTI[2]

[1]VTT Technical Research Centre of Finland, Oulu FI-90570, Finland
[2]Advanced Research Center on Electronic Systems, University of Bologna, Bologna IT-40125, Italy
[3]VTT Technical Research Centre of Finland, Kuopio FI-70211, Finland

Corresponding author: J. Kiljander (jussi.kiljander@vtt.fi)

This work was supported in part by SOFIA and IoE ARTEMIS Projects, in part by IMPRESS/EU Project, in part by the Finnish Funding Agency for Technology and Innovation, in part by the VTT Technical Research Centre of Finland, Oulu, Finland, in part by the Italian Ministry of Education and Research, and in part by the University of Bologna, Bologna, Finland.

ABSTRACT Pervasive computing and Internet of Things (IoTs) paradigms have created a huge potential for new business. To fully realize this potential, there is a need for a common way to abstract the heterogeneity of devices so that their functionality can be represented as a virtual computing platform. To this end, we present novel semantic level interoperability architecture for pervasive computing and IoTs. There are two main principles in the proposed architecture. First, information and capabilities of devices are represented with semantic web knowledge representation technologies and interaction with devices and the physical world is achieved by accessing and modifying their virtual representations. Second, global IoT is divided into numerous local smart spaces managed by a semantic information broker (SIB) that provides a means to monitor and update the virtual representation of the physical world. An integral part of the architecture is a resolution infrastructure that provides a means to resolve the network address of a SIB either using a physical object identifier as a pointer to information or by searching SIBs matching a specification represented with SPARQL. We present several reference implementations and applications that we have developed to evaluate the architecture in practice. The evaluation also includes performance studies that, together with the applications, demonstrate the suitability of the architecture to real-life IoT scenarios. In addition, to validate that the proposed architecture conforms to the common IoT-A architecture reference model (ARM), we map the central components of the architecture to the IoT-ARM.

INDEX TERMS Internet of Things, pervasive computing, RDF, semantic interoperability, system architecture, SPARQL.

I. INTRODUCTION

The term IoT was coined by Kevin Ashton in 1999 when he envisioned new ideas for *radio-frequency identification* (RFID) in a presentation he gave at Procter & Gamble [1]. A central idea in Ashton's IoT vision was to enable devices to publish information about physical world objects into the Web. Object identification, especially RFID, played a central role in the vision. Today the IoT vision is more ambitious and aims at enabling smart and context-aware applications for people everywhere in the world. In a way the current IoT vision can be also seen as a global scale extension of ubiquitous and pervasive computing paradigms [2], [3] focusing on enabling seamless device interaction to assist people in everyday life.

We believe that in order to realize pervasive computing and IoT visions there is a need for common approaches to enable high level interoperability[1] between heterogeneous IoT devices. This would make the functionality and information provided by IoT devices easily accessible for 3rd party application developers and would therefore open a completely new business sector in the same way as smart phones opened the market for mobile apps.

The ability of two systems to interoperate can be presented using different kinds of layered models. For example, Tolk presents a model consisting of six levels: *no connection*,

[1]Here the term interoperability refers to the ability of two systems to communicate and share services with each other.

technical, *syntactical*, *semantic*, *pragmatic/dynamic* and *conceptual* [4]. In [5] similar model is presented by Pantsar-Syväniemi *et al.* consisting of following levels: *connection*, *communication*, *semantic*, *dynamic*, *behavioral,* and *conceptual*. Lappeteläinen *et al.*, on the other hand, describe a more simplified model for interoperability with three levels: *device* level, *service* level and *information* level [6]. In this paper we divide the interoperability challenge into two levels: *connectivity* and *semantic*. In order to achieve device interoperability in pervasive computing and IoT the interoperability challenges in these levels needs to be solved in general way.

The *connectivity* level interoperability covers basically the traditional Open System Interconnection (OSI) model layers from the physical to the transport layer. When devices are interoperable at the *connectivity* level they are able to transmit data with each other. The devices are not, however, able to understand the meaning of data.

The *semantic* level interoperability covers the technologies needed for enabling the meaning of information to be shared by communicating parties. The lack of explicit semantics has not been a problem in traditional communication systems such as phones and computers in general, because the *semantic* level interoperability has been solved by human users who communicate with each other by using devices. In pervasive computing and IoT, devices also become users and they therefore need to communicate directly with each other and interpret the meaning of information in run-time. Early machine to machine (M2M) communication systems (*e.g.* industrial automation systems) were typically closed systems built for a particular purpose and did not thus require common *semantic* level interoperability solutions. The exact functionality of such systems and interactions between different components were fixed and tested in design-time. Because of this the systems were predictable and robust. Disadvantages are also obvious, namely difficulty in adding new features and devices into the system. Consequently, this kind of hard-wiring is not feasible for IoT or pervasive computing systems that require open and extensible approach for interoperability.

In order to provide interoperability and achieve wide acceptance, it is nowadays typical to utilize existing and widely used technologies and protocols such as the Devices Profile for Web Services (DPWS) [7], Universal Plug and Play (UPnP) [8], OSGi [9], Constrained Application Protocol (CoAP) [10] and MQ Telemetry Transport (MQTT) [11] when developing M2M communication systems. While the aforementioned technologies provide basic interoperability mechanisms within their domain, they do not enable interoperability at the *semantic* level (rather they can be classified as *syntactical* and *service* level interoperability solutions using the interoperability levels proposed in the literature). What is further needed are mechanisms for explicating and representing semantics of information in a reusable and machine-interpretable format.

Fortunately, the semantic level interoperability has been identified as a main goal in another Future Internet paradigm called the Semantic Web [12]. Although the Semantic Web knowledge representations technologies such as Resource Description Framework (RDF) [13], RDF Schema (RDFS) [14], and Web Ontology Language (OWL) [15] have been originally designed for representing Web resources, the same technologies can be utilized also in other domains. In fact, the structured and flexible linked data model provided by the RDF is well-suited for representing knowledge about the physical world. Consequently, semantic technologies have been used in various IoT related projects such as Networked Embedded System Middleware for Heterogeneous Physical Devices (HYDRA)[2], SENSEI (Real World Dimension of the Network of the Future)[3], Open Source Solution for the Internet of Things into the Cloud (OpenIoT)[4], and Internet Connected Objects for Reconfigurable Ecosystems (iCore)[5], just to name a few.

In this paper we present novel semantic level interoperability architecture for pervasive computing and IoT systems. A core idea in the architecture is to enable all kind of devices, even low capacity sensors and actuators, to interact with each other only by sharing semantic information via common knowledge brokers. The architecture builds upon semantic information sharing solution called M3 (or Smart-M3) [16] which we have been developing in several projects. The physical object identification and resolution in the proposed architecture is based on the *Ubiquitous ID* (uID) architecture (from uID Center, Japan) [17] which provides globally unique identifiers, called *ucodes*, for physical objects and methods to find information about objects from all over the world.

The architecture design has been an iterative process in which we have used feedback and experience obtained from numerous applications and reference implementations that we have developed during the last five years. Main differences in the proposed architecture and existing semantic technology based IoT solutions can be summarized as follows:

1) In the proposed architecture all interaction between devices is based on semantic information sharing with SPARQL 1.1 [18], [19]. This means that even data provided by resource restricted sensors in high frequency is published and accessed in semantic format. To make this possible we have developed both a protocol [20] that provides SPARQL 1.1 like functionality in a compact binary format and a processing engine that enables efficient processing of SPARQL subscriptions.
2) In contrast to the Giant Global Graph enforced by Linked Data [21] community we propose that the virtual representation of the physical world is divided into numerous local graphs that represent the dynamically changing context for a particular pervasive computing environment in a given point of time. To enable devices to discover these local graphs from anywhere in the

[2]http://projecthydra.org/
[3]http://www.sensei-project.eu/
[4]http://openiot.eu/
[5]http://www.iot-icore.eu/

world the architecture proposes a global graph that represents necessary information about all the local graphs in the world.

The rest of the paper is structured as follows. In the section II we present existing semantic technology enhanced pervasive computing and IoT interoperability approaches and highlight the main novelties in the proposed architecture by comparing it to these solutions. Section III gives an overview to the architecture. In section IV we describe the interaction model within a single smart space. Section V presents how the architecture is extended from local pervasive computing environment to the world wide IoT. In the section VI, we present the evaluation for the architecture. The evaluation starts by mapping the proposed architecture to the IoT-A Architecture Reference Model (ARM) [22] to show that the architecture conforms to the common IoT architecture reference model. Then we present reference implementations of central architecture components and proof-of-concept implementations to various application domains. Performance of the reference implementation is also evaluated in a typical IoT scenario. Finally, the section VII presents conclusions and ideas for future work.

II. RELATED WORK

The possibility for semantic level machine to machine interoperability has inspired various research projects to exploit Semantic Web technologies in development of pervasive computing and Internet of Things systems. Many of these projects use semantic technologies to improve traditional service oriented architectures. Some noteworthy examples of this type of solutions include HYDRA middleware (currently known as LinkSmart) [23], Task Computing Environment (TCE) [24], COCOA [25], Ambient Intelligence for the Networked Home Environment (Amigo) middleware [26], and Semantic Middleware for IoT [27]. In HYDRA middleware semantic technology enhanced Model Driven Architecture is combined with service oriented architecture to provide methods for developing pervasive computing applications on top of heterogeneous devices and sensor networks. In TCE the Semantic Web technologies are used to enhance SOAP based Web Services to enable service discovery and mash-up on behalf of the user. The COCOA continues with the idea introduced by TCE and provides OWL-S [28] based middleware for service discovery and composition. The COCOA approach was further developed in the Amigo project. This work led to Amigo-S semantic service description language which can be used to enhance various SOA based services supported by the Amigo middleware. In Semantic Middleware for IoT the idea is to transform existing syntactical level interoperability solutions such as Bluetooth and UPnP into Semantic Web services that can be more easily discovered and combined according to user requirements.

In addition to enhancing SOA based services the semantic web technologies can be used in many other ways to improve the functionality of a pervasive computing middleware. One example of this is the breadboard architecture [29] which focuses especially on the integration of sensors and perceptual components manufactured by different technology providers. The perceptual components access sensor data via the NIST Smart Flow middleware [30] and extract context cues from the raw sensor data. In addition to sensors and perceptual components central entities in the approach are agents that track higher level contextual situations and provide service logic for the pervasive applications. Semantic technologies are used for persistent storage of the data created by sensors, perceptual components and agents.

In contrast to the aforementioned approaches where semantic technologies are used either for representing metadata about services or as a persistent storage for context information, the basis of the whole interaction model in our architecture lies on semantic information sharing. In practise, this means that all the agents interact with each other only by performing queries, updates, and subscriptions represented in SPARQL 1.1 syntax (*i.e.* there is no need for traditional services or proprietary data formats in our architecture model). Because of this our approach can benefit from semantic level interoperability in all communication and is thus, in a way, closer to the original vision of Semantic Web.

There exists also pervasive computing and IoT solutions that use semantic technologies in more similar way to our architecture than the aforementioned approaches. For example, the Context Broker Architecture for Pervasive Computing (CoBrA) [31], Semantic Space [32], and SPITFIRE [33] are examples of solutions that utilize semantic technologies for sharing information about real world things.

The CoBrA is broker-centric agent architecture that aims at providing centralized view of context for pervasive computing applications. Context is represented according to CoBrA ontology and a centralized context broker creates and shares the context for all parties in the pervasive computing environment. The context broker consists of four functional components: Jena 2 based knowledge base, rule-based context-reasoning engine, context-acquisition module and privacy-management module. The agent communication in CoBrA is based on Foundation of Intelligent Physical Agents (FIPA) [34] standards.

Semantic Space is a semantic technology based pervasive computing infrastructure that focuses on explicit representation, flexible reasoning, and expressive querying in pervasive computing environments. Similarly to CoBrA a core part of the Semantic Space approach is context ontology, namely *upper-level context ontology* (ULCO), which defines the basic concepts common for pervasive computing environments. The context information is collected, stored, and processed by a context infrastructure which consists of context wrappers, context aggregator, context reasoner and knowledge base with RDQL query engine. The context wrappers are implemented as UPnP services and context aggregator as UPnP control point.

Our architecture differs from CoBrA and Semantic Space architectures in three main ways. The first notable difference is that in CoBrA and Semantic Space there is high number of primary actors because of clear distinction between agents which utilize context information and entities (*e.g.* sensors and cameras) that provide information to the RDF database. In CoBrA the context broker actively acquires context information from various sources via the context acquisition module. In Semantic Space the architecture contains context wrappers for weather, devices, location, environment and activity, for example. In our architecture, on the other hand, all information is produced and consumed by agents. This is made possible by compact semantic information sharing protocol that we have developed to enable utilization of semantic technologies in resource restricted devices and networks. The second difference is that the CoBrA and Semantic Space do not describe how the link between physical object and their virtual representation is created. In our architecture the physical objects are identified with 128-bit *ucodes* (typically stored into a tag) and the same *ucode* is used as an identifier for the object in the RDF graph. We first introduced this idea in the paper [35]. The third difference is that the CoBrA and Semantic Space are targeted for local pervasive computing environments and are consequently centralized. We, on the other hand, extend the approach to IoT and propose a highly distributed architecture where information is divided into numerous information brokers that are differentiated, for example, by geographical location, owner, and information they contain.

In contrast to the CoBrA and Semantic Space that use semantic technologies to provide context-aware services for local pervasive computing environments, the SPITFIRE targets to the IoT domain and provides service infrastructure for Semantic Web of Things. In particular, the SPITFIRE aims at making it as easy as possible for application developers to exploit Internet-connected sensors. The SPITFIRE architecture is based on linked open data cloud and RESTful CoAP/HTTP sensors operating on top of 6LowPAN network. The basic idea is to fetch raw data from sensors via RESTful interface, transform the data into semantic format, and finally link the data to the linked open data cloud. The linking is enabled by a specific ontology vocabulary proposed by the SPITFIRE.

The main contributions in SPITFIRE over preceding semantic sensor web approaches are twofold. First, SPITFIRE proposes semi-automatic creation of semantic sensor descriptions based on hypothesis that sensors producing similar data can be presented using similar representations. In our architecture we have partly avoided this problem by making it possible to utilise semantic technologies in resource restricted sensors and actuators (and other type of devices). This way the resource restricted devices can directly publish information in semantic format and there is no need to create descriptions manually or semi-automatically. However, in situations where non-semantic devices need to be integrated the semi-automatic creation of device descriptions proposed by SPITFIRE would be very useful and we plan to investigate the possibility utilise them in the future. Second main contribution of SPITFIRE is prediction model based approach for avoiding sensor state updates at high frequency into the linked data could. In our approach this problem has been tackled by dividing the world wide linked data cloud into plethora of local pervasive computing environments. This way the sensor updates (and other information sharing) can be localised and parallelized into local area instead of the whole world. Additionally, by optimizing the SPARQL query engine for subscriptions we have made it possible to subscribe directly to sensor state updates using SPARQL 1.1 query syntax. Other notable differences between SPITFIRE and the architecture proposed in this paper are following: 1) SPITFIRE seems to mainly focus on sensors (and objects they monitor) whereas we provide also means to interact with actuators (or any other type of device) with semantic technologies, 2) the current reference implementation of SPITFIRE is centralized[6] whereas we propose a distributed architecture, 3) similarly to the previous approaches the SPITFIRE does not describe how the link between a physical object and their virtual counterpart is created.

We are not the only ones exploiting unique object codes as links to data in semantic technology enhanced IoT systems however. For example, in [36] Ziegler *et al.* present an approach for using RFID as universal entry point to Linked Data clouds. Instead of using tag technology independent identifiers such as *ucode* the approach utilizes unique identifiers stored into every RFID tag. The obvious benefit with this approach is that the RFID tags do not have to be programmed by developers. Another notable difference between the approaches is the way object codes are represented in RDF graphs. In the approach proposed by Ziegler *et al.* the RFID identifier is linked to the URI of the virtual representation via a property called *rfid*. We, on the other hand, use directly the *ucode* in URI format as a virtual identifier for the physical object.

When compared to the approach proposed by Ziegler *et al.* main advantages of our approach are twofold. First, our approach allows any kind of tag technologies to be used (*e.g.* RFID, BLE, NFC, QR code). The second benefit of using *ucodes* instead of normal RFIDs comes from the *Ubiquitous ID* (uID) architecture which provides methods for locating information related to tagged objects. The resolution functionality in the uID architecture is provided by a distributed three level server hierarchy, called the *ucode Resolution Server*. The highest level in *ucode Resolution Server* hierarchy is called a *Root server*. It manages the whole 128 bit *ucode* space. Top Level Domain (TLD) servers are located below the Root server and manage a 108 bit code space. The Second Level Domain (SLD) servers form the third level below TLD servers. Their address space ranges from 16 bits to 96 bits.

III. ARCHITECTURE OVERVIEW

There are two fundamental characteristics in the proposed semantic level interoperability architecture. First, to make

[6]The paper mentions that it is possible to implement the central RDF triplestore as a distributed cloud service however.

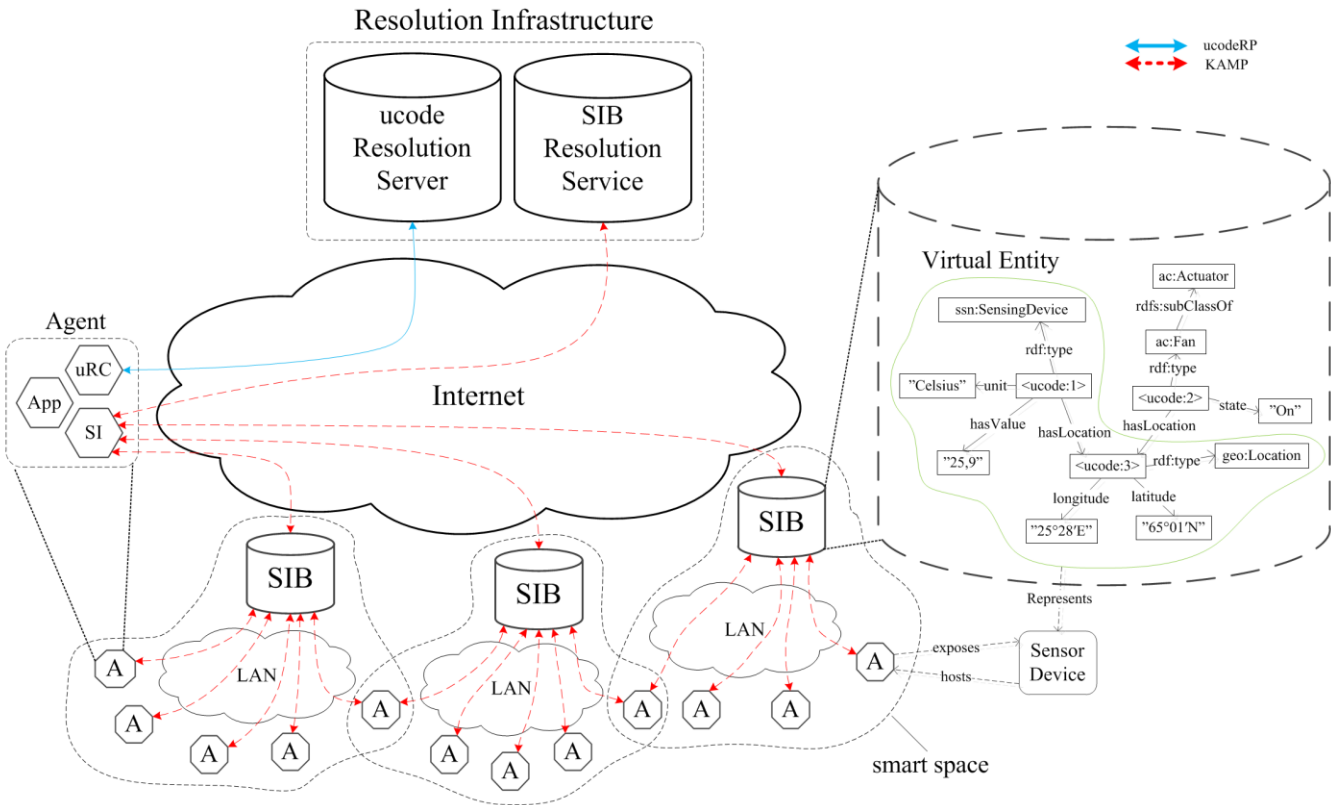

FIGURE 1. Overview of the semantic level interoperability architecture for IoT.

the interaction between devices as flexible as possible we rely on information centric view in the interaction. This means that instead of traditional end-to-end communication between domain specific services and clients the interaction model is based on autonomous agents which interact by sharing semantic information (*i.e.* information represented with RDF and OWL) with each other. The main advantage of using RDF and OWL is that they provide a common way to describe information in generic machine-interpretable form and thus both provide means for semantic level interoperability and support information reusability. Second, to make the interaction in IoT more scalable we propose a distributed architecture model consisting of plethora of local ubiquitous computing environments, called smart spaces. The main advantage of this kind of approach is that it makes privacy management easier and improves performance since all local information does not need to be globally accessible in a centralized database.

There are four main entities in the system model illustrated in the Fig. 1: *Sematic Information Broker* (SIB), *Agent*, *Virtual Entity* (VE), and *Resolution Infrastructure* (RI). The SIB is a semantic information sharing service providing semantic level communication channel for *Agents* to interact with each other. In this paper the protocol defining the rules and syntax for *Agent*-SIB communication is referred to as Knowledge Access and Management Protocol (KAMP). We propose KAMP where the operations to access and manipulate the knowledge are based on SPARQL 1.1 Query and Update languages. The possible realizations of the KAMP protocol are discussed in more detail in the section IV.

Information available in a SIB represents the context for a single smart space. Here the term smart space refers to certain physical space where one or more *Agents* monitor the environment and share their information and capabilities via a SIB. Typically, the smart space relates to a static physical location such as room, but it is also possible to have mobile smart spaces such as vehicles and people. Several smart spaces can also share the same physical location. For instance, there can be one SIB assigned for an apartment building as a whole, for each apartment inside the apartment building, for each room inside an apartment, and for each person (*i.e.* mobile smart space) inside a room. Although all these SIBs exist in the same area they still represent logically different smart spaces from the architecture point of view.

Main difference between the SIB and a normal SPARQL endpoint is that whereas SPARQL endpoints are designed for performing efficient queries on relatively slowly changing data, the SIB is targeted for enabling the *Agents* to share knowledge about the physical world in real time.[7] To this end,

[7]By real time we mean that the SIB is typically able to notify subscribers without perceivable delay so that, for example, controlling lights is possible in responsive manner. Fixed deadlines required in traditional real-time systems cannot be guaranteed however.

the SIB provides means both for subscribing to events occurring in the smart space (expressed as persistent SPARQL query) and for creating persistent SPARQL updates that modify the context of the environment when certain events occur. The operations provided by the SIB are defined in more detail in the section IV.

The virtual representation of the smart space is created by *Agents* which realize applications for end-users by interacting each other via the SIB. *Agent* can be divided into three parts: *Semantic Interface* (SI), *ucode Resolution Client* (uRC), and *application logic*. The *Semantic Interface* provides means both for transforming information from local data formats to semantic format and for interacting with the *local SIBs* and the *SIB Resolution Service*. The role of the *uRC* is to interact with the *ucode Resolution Server* to locate a SIB service hosting information about a particular physical world object. In addition to the aforementioned components an *Agent* can contain means for both collecting information about and interacting with the physical world (*i.e.* sensor and actuator agents). This functionality is not necessarily required however (*i.e.* the *Agent* can also be just an aggregator that modifies the information provided by other *Agents*).

When *Agent* publishes information about a real word object into a SIB the collection of RDF triples representing the object is called VE. The VE can represent any kind of physical world entities such as sensors, actuators, food products, buildings and people, to name a few. The VE both provides information about and allows interaction with the corresponding physical world entity. Our architecture is agnostic to the ontology used to represent VEs and we do not enforce any particular ontology. A key idea in proposed architecture is that VEs are identified with *ucodes* and when feasible the same *ucode* is stored into a tag attached to the corresponding physical object. This way we can easily link the Virtual Entity to its physical counterpart.

The *Resolution Infrastructure* enables *Agents* to locate Semantic Information Brokers of interest. It provides resolution both based on identifiers and specifications. The former is called *lookup* and the latter *discovery*. In practise, the logical *Resolution Infrastructure* consists of two components: *ucode Resolution Server* and *SIB Resolution Service*. The *SIB Resolution Service* is a SIB (with known Internet address) that stores semantic descriptions of all other SIBs and provides discovery functionality for the *Agents*. The *ucode Resolution Server* is responsible for providing lookup functionality by mapping the physical object identifiers (*i.e. ucodes*) directly to SIB address where information about the particular VE can be found.

IV. AGENT INTERACTION WITHIN LOCAL SMART SPACE

A key idea in the proposed information centric interaction model is that the interaction between *Agents* is based solely on the information they share via a common knowledge broker. This means that sensor measurements can be read and actuators controlled by simply querying and modifying their *Virtual Entities* available in the SIB. This kind of blackboard architectural pattern based interaction model is flexible and decouples *Agents* efficiently so that the *Agents* do not need to know anything about each other.

TABLE 1. Data manipulation and access operations.

Operation	*Description*
Insert	Inserts RDF triples into the SIB. The triples can be either concrete or constructed in WHERE pattern matching.
Delete	Deletes RDF triples from the SIB. The triples can be either concrete or constructed in WHERE pattern matching.
Update	First deletes triples from then inserts triples into the SIB. The triples can be either concrete or constructed in WHERE pattern matching.
Query	Request information from the SIB. Returned results depend on the query format (SELECT, ASK, or CONSTRUCT).
Subscription	Registers a persistent query into the SIB. The SIB sends a notification (new and obsolete results) every time the results of the query change.
Persistent insert	Registers persistent insert operation into the SIB. New triples are constructed (and inserted) every time the WHERE pattern matching provides a solution.
Persistent delete	Registers persistent delete operation into the SIB. New triples are constructed (and deleted) every time the WHERE pattern matching provides a solution.
Persistent update	Registers persistent update operation into the SIB. New triples are constructed (and inserted and deleted) every time the WHERE pattern matching provides a solution.
Terminate	Terminates a persistent operation.

The operations proposed in the architecture to manipulate and access semantic information in the smart space are presented in the Table 1. All the operations (except *terminate*) are based on SPARQL 1.1 Query and Update Languages. In addition to these operations a SIB can support various graph level management operations defined in SPARQL 1.1 Update Language.

The presentation format for the operations (*i.e.* encoding and serialization) depends on the used KAMP. There are currently two realizations for the KAMP available: *Smart Space Access Protocol* (SSAP) and *Knowledge Sharing Protocol* (KSP).

The SSAP is the name for the M3 communication protocol supported by the first SIB reference implementations. The original SSAP utilized non-standard data manipulation and query operations. We have modified the SSAP so that it now uses SPARQL 1.1 as a payload and provides *Agents* with the operations defined in the Table 1. There are two serialization formats for SSAP available: XML and World Aligned XML (WAX). SSAP/XML is the original serialization format. The SSAP/WAX provides a more compact serialization format enabling it to be more easily exploited in resource restricted devices [37].

The KSP is designed especially for low capacity devices and resource restricted networks [20]. It provides SPARQL 1.1 like mechanism for querying, updating, and subscribing to RDF data in compact binary format. The persistent format for update operations was first introduced in KSP. KSP also enables *Agents* to define maximum size for SIB responses which makes it more suitable for resource restricted devices. In a way the KSP is the same for SSAP that the CoAP is for HTTP.

One of the main principles in the proposed architecture is that we do not enforce any particular transport, network or radio protocol to be used by *Agents* within a single smart space. The only requirement is that the local SIB needs to support all the connectivity level technologies used by the *Agents*. Because the *Agents* only interact with each other via the SIB this makes it possible for devices using heterogeneous *connectivity* solution to share information and interoperate with each other. If an *Agent* needs to interact with a remote smart space via the Internet it needs, of course, use IP or 6LowPan network protocols.

In order to illustrate the information centric communication model in practise we will next demonstrate it in a simple scenario consisting of three types of *Agents*: *Sensor Agent*, *Fan Agent*, and *Control Agent*. In this scenario we have a house with many rooms. Each room is equipped with a temperature sensor and a fan actuator exposed via the *Sensor Agent* and the *Fan Agent*, respectively. A single *Control Agent* is responsible for controlling the fans in the rooms (*i.e.* turning the fans on in a room when the temperature rises above a certain limit[8] defined separately for each room).

All the *Agents* need to first discover the SIB responsible for the local smart space (section V describes the discovery process in more detail). Then each *Sensor* and *Fan Agent* need to virtualize the corresponding physical entities by publishing their VE descriptions into a SIB responsible for the smart space of the house. Following INSERT DATA operations demonstrate how this can be done for a single temperature sensor and fan actuator.

```
# Inserts example temperature sensor
INSERT DATA
{
  ucd:3690 a ex:TemperatureSensor ;
        ex:value 22.9 ;
        ex:unitType "C" ;
        ex:location ucd:7891 .
  ucd:7891 a ex:Room .
}
```

[8]This limit could be, for example, calculated from the temperature preference values of the persons present in the given room.

```
# Inserts example fan actuator
INSERT DATA
{
  ucd:3695 a ex:FanActuator ;
        ex:state 0 ;
        ex:location ucd:7890 .
  ucd:7891 a ex:Room .
}
```

In these operations it is assumed 1) that the *ucode*[9] (and URI) of the room is obtained by reading a tag attached to the room when the sensor and the actuator are deployed to the room and 2) that the ucode of the sensor and actuator is known by the developer of the given *Agent*. Note that for the sake of clarity prefix definitions are omitted from all the examples presented in this paper.

In addition to publishing the VE descriptions, each *Fan Agent* needs to subscribe to the VE attribute specifying the fan state to be aware of modifications made by other agents and each *Sensor Agent* needs to update the temperature value whenever the temperature changes in the room. Examples of these operations can are presented below.

```
# Subscription to the fan state
SELECT ?state
WHERE
{
  ucd:3695 ex:state ?state .
}
# Temperature value update .
DELETE { ucd:3690 ex:value ?oldvalue }
INSERT { ucd:3690 ex:value 27.0 }
WHERE { ucd:3690 ex:value ?oldvalue }
```

The application logic in this scenario is realized by the *Control Agent* which subscribes to the fans that need to be turned on. This subscription can be performed as follows.

```
SELECT ?fan
WHERE
{
  ?sensor a ex:TemperatureSensor ;
            ex:value ?temperature ;
            ex:location ?room .
  ?room a ex:Room ;
        ex:maxTemperature ?maxTemp ;
  FILTER (?temperature > ?maxTemp) .
  ?fan a ex:Fan ;
       ex:location ?room .
}
```

The first result set of the subscription contains all the fans that need to be turned on. After this the *Control Agent* will receive notifications of fans that need to be turned on (new results) and fans that need to be turned off (obsolete results). The fans can be turned on and off with following simple DELETE and INSERT DATA operations.

```
# Turns the fan ucd:3695 on.
DELETE DATA { ucd:3695 ex:state 0 }
INSERT DATA { ucd:3695 ex:state 1 }
```

[9]For clarity sake the ucodes are presented in a simple format.

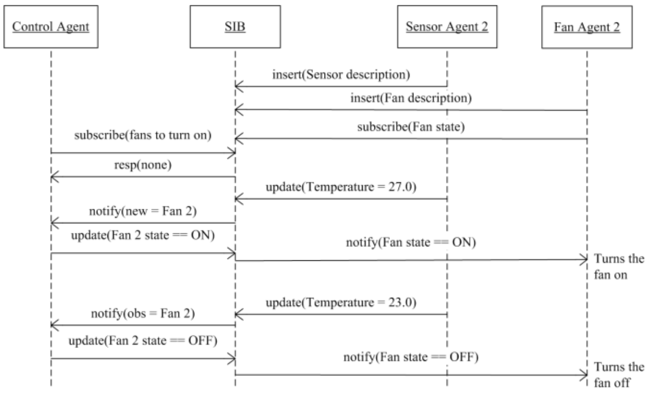

FIGURE 2. Example of agent interaction within a smart space (control agent implemented with SPARQL subscription and update).

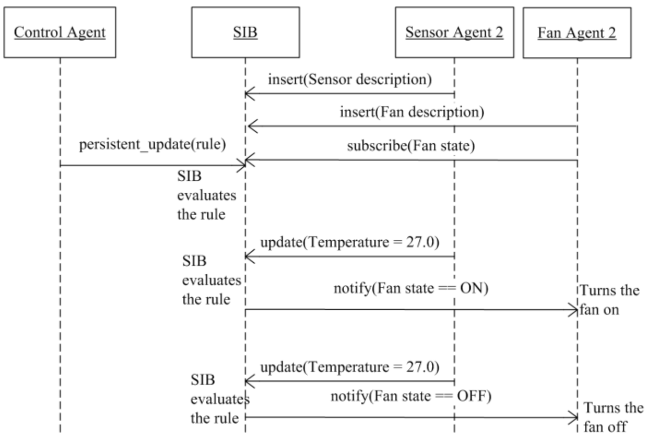

FIGURE 3. Example of agent interaction within a smart space (control agent implemented with a single persistent SPARQL update).

```
# Turns the fan ucd:3695 off.
DELETE DATA{ ucd:3695 ex:state 1 }
INSERT DATA{ ucd:3695 ex:state 0 }
```

Alternatively, the *Control Agent* could also create a persistent SPARQL update rule which modifies the state of the fans when necessary. This generic persistent update can be presented as follows.

```
DELETE { ?fan ex:state ?current }
INSERT { ?fan ex:state ?new }
WHERE
{
  ?sensor a ex:TemperatureSensor ;
           ex:value ?temp ;
           ex:location ?room .
  ?room a ex:Room ;
        ex:maxTemperature ?maxTemp .
  ?fan a ex:Fan ;
       ex:location ?room ;
       ex:state ?current .
  BIND (IF(?temp > ?maxTemp, 1,0) AS ?new)
  FILTER (?new != ?current)
}
```

In this case the whole *Control Agent* is implemented with this single rule and no other interaction by the *Control Agent* is needed. Fig. 2 and 3 illustrate the message exchange between the *Agents* and the SIB in the example scenarios.

V. FROM LOCAL SMART SPACES TO INTERNET OF THINGS

When the idea of semantic interoperability is extended to global IoT domain there is a need 1) for globally unique methods to identify physical objects and 2) for ways to resolve the address of the SIB that provides information about or ways to interact with the physical object. In the sections A–C we describe how this is achieved in the proposed architecture.

A. VIRTUAL ENTITY IDENTIFICATION AND LLOOKUP

We use *ucode* based Uniform Resource Name (URN) [38] as the identifier (*i.e.* URI) of the virtual representation instead of HTTP URI [39] recommended for Linked Data. There are two reasons for this. First reason is that the HTTP URI is meant for identifying Web resources whereas *ucode* is designed to identify physical world objects. This might seem to be more a philosophical issue than an actual problem, but this can also have impact in practise. This is because the HTTP URI points always to a specific network location and that location is presented in the identifier itself (*i.e.* the domain name). This causes problems in practise if the author owning the domain name does not want to maintain the database presenting the physical object anymore or if the domain name is lost for some reason. This would mean that a new URI for the physical object would need to be assigned and this new URI would need to be updated to every description related to that VE and to the tag attached to the physical object the VE represents. Additionally, many physical objects are mobile which would cause problems with the architecture proposed in the paper if the HTTP URIs are used as physical object identifiers. This is because each smart space represents VEs in its geographical location and the HTTP URIs point always to a fixed service and it would not be thus possible to update the VE address (*i.e.* URI) when the VE moves from one smart space to another. The second reason to use *ucode* as identifier for physical objects instead of HTTP URI is that the *ucode* has a fixed length (typically much shorter than verbose HTTP URIs) and it is therefore more predictable in term of memory and faster to process. This makes it more feasible for resource constrained devices and networks that are typical in IoT.

When compared to other physical object identification methods the main advantage of using *ucodes* comes from the *uID Resolution Architecture* which provides methods for 1) creating new identifiers for objects and for 2) resolving the network address of the SIB containing information about the physical object. This type of resolution is referred as *lookup* in the architecture proposed in this paper. The SIB associated to a given *ucode* is called *primary SIB*. In addition to the *primary SIB*, it is possible that information about given VE exists in other SIBs. This distribution of VE information into multiple SIB services is discussed in more detail in the section B.

When *Agent* needs to *lookup* an address of a SIB containing information about a VE identified by *ucode* it simply sends a *resolve* request with the *ucode* as a parameter to the *ucode Resolution Server.* The *ucode Resolution Server* responds

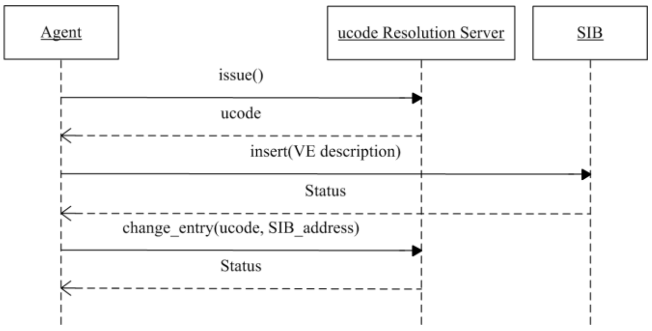

FIGURE 4. Virtual Entity publishing process.

with the SIB address represented with a non-standard *sib* URI scheme defining the hostname and port for the SIB (e.g. sib:example.com:10010).

In order to make the above-described *lookup* possible the *Agents* publishing representations of the physical world entities are required to update the SIB address containing the representation of the physical object to the *ucode Resolution Server*. The process related to creating and publishing a VE (illustrated in the fig. 4) works as follows. When a new VE is created an *Agent* requests a unique *ucode* for the physical object (and the corresponding VE) from the *ucode Resolution Server* with the *issue* request. If possible this ucode is stored to a tag and the tag is attached to the physical object. Then the *Agent* publishes the VE description into a local SIB. Finally the *Agent* updates the SIB address for the given ucode by sending a *change_entry* request to the *ucode Resolution Server*.

B. VIRTUAL ENTITY DISTRIBUTION

There are many situations where information about a single VE can be distributed into multiple SIB services. This is very typical in the case of mobile objects such as cars, people, and mobile phones, but can also happen with stationary objects. For instance, a device manufacturer could store basic information about a product (*e.g.* washing machine) in their official semantic database (*i.e.* the *primary SIB*) which is linked to the *ucode* of the product unit in the *ucode Resolution Server*. When user purchases the washing machine and deploys it in her/his personal smart space the washing machine will start publishing semantic information to the local SIB. This information cannot be found by resolving the *ucode* attached to the washing machine however. To make information distributed into various SIBs discoverable, links to these other SIBs need to be added to the *primary SIB*. In practise this could happen so that when a customer registers the product her/his *Personal Agent* will add a link to the home SIB into the *primary SIB* (*i.e.* the manufacturer's SIB in this case). This allows the manufacturer to keep track of their devices and act immediately in the case of device malfunction.

To make the other SIB services associated to certain VE attributes discoverable there is a need for a common ontology which defines these associations in a machine interpretable format. To this end, we have defined two properties, namely *as:hasAttribute* and *as:associatedTo,* which can be used to represent VE attributes that are associated to other SIBs instead of the *primary SIB*. The *as:hasAttribute* property is used to define the property associated to another SIB. The *as:associatedTo* property links the property to a specific SIB instance. Following RDF triples illustrate parts of an example VE description (stored to a *primary SIB*) that has two attributes associated to another local SIB:

```
ex:Device a ssn:SensingDevice ;
     ex:madeIn dpedia-owl:Taiwan ;
     as:hasAttribute ex:value ;
     as:hasAttribute ex:location .

ex:value as:associatedTo < sib:vtt.fi:10011> .
ex:location as:associatedTo < sib:vtt.fi:10011> .
```

C. SEMANTIC INFORMATION BROKER DISCOVERY

In addition to the *lookup* operation described in the section V-A, our architecture provides methods for resolving SIB address based on SIB service specification. This operation is called *discovery*.

The information model defining the type of information that can exist in SIB service description and specification is represented in the SIB service profile (SSP) ontology. A central concept in the SSP ontology is the *ssp:SIB* class representing all SIB services. The *ssp:serviceArea* property ties a SIB instance to a physical location in which the SIB is active. The range for the *ssp:serviceArea* property is defined as both the *geo:SpatialThing* class from the W3C Basic Geo[10] vocabulary and the *geo:Feature* class from the GeoSPARQL ontology [40]. This is because although the GeoSPARQL provides far more advanced methods for spatial queries than the very simple W3C Basic Geo it is not yet widely supported in SPARQL endpoints. In practise, this means that the SIB service area can be presented either as a simple point using W3C Basic Geo or GeoSPARQL ontologies, or as a more complex geographical area are using GeoSPARQL ontology.

The *ssp:owner* property can be used to specify the owner of the SIB service. The range for the ssp:*owner* property is the *foaf:Agent* class from FOAF ontology [41] and it includes thus both organizations and individual people. In addition to the properties and classes used the present the owner and the geographical area of the SIB, the SSP ontology contains properties, namely *ssp:containsClass* and *ssp:containsProperty* for representing the domain specific RDFS/OWL classes and properties that are used to represent VEs inside a SIB. These properties make it possible to model the content of each smart space in terms of classes and properties and therefore enable an *Agent* to discover SIBs that contain information modelled according to a certain domain specific ontology (or more precisely information modelled using the classes and properties). This information is useful when there are many SIBs in the same geographical area but the SIBs contain information represented with different classes and properties.

[10]http://www.w3.org/2003/01/geo/

To enable global SIB discovery there is a need for entity which stores the RDF graph representing all the SIBs in the world. This is the role of the *SIB Resolution Service*. Although the *SIB Resolution Service* is logically a single entity the global SIB service graph can be also stored into distributed cloud service. The distribution of the *SIB Resolution Service* is out of the scope of the paper however. Following listing represents a simple example for the content of the *SIB Resolution Service* consisting of two SIB descriptions:

```
<sib:example.com:10010> a ssp:SIB ;
     ssp:owner <http://www.vtt.fi/> ;
     ssp:containsClass ex:SensorDevice ;
     ssp:containsProperty: ex:hasValue ;
     ssp:serviceArea _:area1 .
_:area1 a geo:SpatialThing ;
     geo:lat 65.056697 ;
     geo:long 25.45819 .
<sib:example.com:10011> a ssp:SIB ;
     ssp:owner <mailto:john.doe@vtt.fi> ;
     ssp:containsClass ex:Actuator ,
                  ex:Sensor ;
     ssp:serviceArea _:area2 .
_:area2 a geo:SpatialThing ;
     geo:lat 65.056695 ;
     geo:long 25.45818 .
```

An *Agent* that wants to discover a SIB that provides means to interact with an actuator represented with *ex:Actuator* class within the premises of VTT Oulu could perform a following SPARQL 1.1 query to the *SIB Resolution Service*:

```
SELECT ?sib
WHERE {
  ?sib ssp:containsClass ex:Actuator
         ssp:serviceArea ?area .
  ?area geo:long ?long;
          geo:lat ?lat .
  FILTER (xsd:float(?long) > 25.45580 &&
            xsd:float(?long) < 25.45600 &&
            xsd:float(?lat) > 65.05680 &&
            xsd:float(?lat) < 65.05700)
  ?sib a ssp:SIB }
```

The SIB descriptions are published and updated by a special purpose *Agent,* called the *SIB Advertiser Agent,* assigned for every SIB. When a new SIB (and *SIB Advertiser Agent*) is launched, the *SIB Advertiser Agent* first publishes the SIB service description into the *SIB Resolution Service*. Then it subscribes both to all RDFS/OWL classes that have instances and all RDF properties that are used in triples. This way it is able to keep track of the type of VEs published into the SIB at any given time. When the SIB notifies that a new class or property has been inserted or that an instance of a class or property has been removed, the *SIB Advertiser Agent* updates the SIB service description into the *SIB Resolution Service*. In addition to updating the SIB description when the content of the SIB change, the *SIB Advertiser Agent* is also responsible for updating the SIB service description whenever any other SIB description parameter changes (*e.g*. service area in the case of a mobile SIB). Fig. 8 presents a sequence chart illustrating the SIB service description management process.

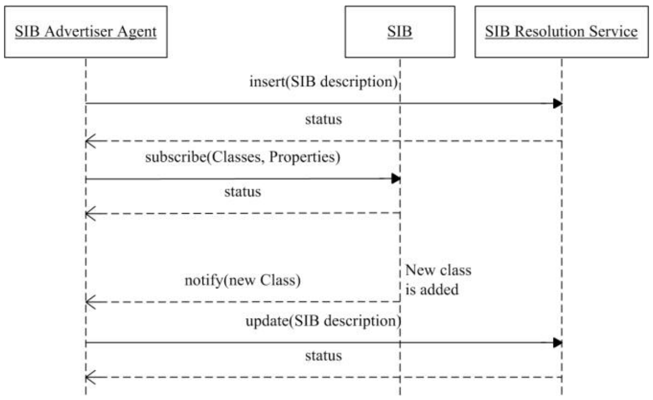

FIGURE 5. SIB service description management process.

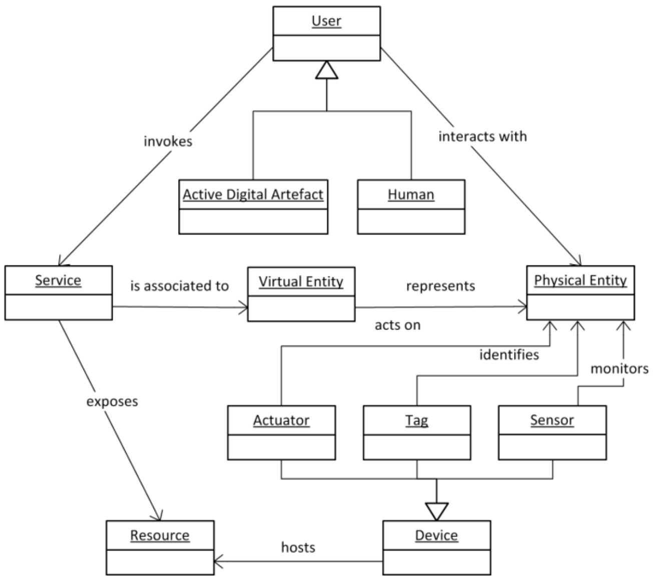

FIGURE 6. IoT-A Domain Model.

VI. EVALUATION

A. MAPPING TO THE IoT-A ARCHITECTURAL REFERENCE MODEL

To our knowledge, the only formal reference model for IoT architectures is the IoT-A ARM developed in the Internet of Things - Architecture project. The IoT-A ARM has four main purposes. First, it serves as a cognitive aid by providing terminology for IoT concepts. Second, the reference model provides a common grounding for specific IoT architectures. Third, the IoT-ARM can be exploited in architecture generation process. Fourth, by mapping existing IoT architectures to the IoT-A ARM it is possible to better compare these architectures and find similarities between them. In order to both evaluate that the proposed architecture aligns with the IoT-A ARM and to make the architecture more easily understandable for people who are familiar with the IoT-A ARM we will next make a brief comparison between them.

The IoT-A ARM consists of several models and views. In this mapping we will mainly focus to the Domain Model (DM) (depicted in the Fig. 6) which is a central model in the ARM and defines the main IoT concepts and their relationships. There are six core entities in the Domain Model: *Physical Entity (PE)*, *User, Virtual Entity (VE)*, *Resource, Service*, and *Device*.

A *Physical Entity* is an identifiable object of the physical world that is relevant for an IoT application. They can be living organisms such as humans and animals or inanimate objects such as cars and buildings, for example. The *Users* are *Humans* or *Active Digital Artifacts* (*e.g*. software agents) that interact with a *Physical Entity* to achieve a certain goal. The *Virtual Entity* is a digital representation of a *Physical Entity*. A *VE* description consists of a unique identifier (VE-ID) and a set of attributes. Ideally, the *VE* attributes provide a synchronized representation of *PE* properties. This means that the attributes not only provide real-time information about the *PE*, but may also allow interacting with it. For example, by modifying the state attribute of a light switch *VE* it is possible to turn the lights on and off in the real world.

The *VEs* are created by computing platforms that have methods to monitor and interact with the *PE*. This kind of computing platform is typically divided into software and hardware components. In IoT-A terms the software component is called *Resource* and the hardware component is called *Device*. The IoT-ARM further classifies devices into three groups: *Sensors*, *Actuators*, and *Tags*. To interact with the *Resource* (and *Device*) there is a need for standard interface that defines the rules and syntax for interaction. In IoT-A ARM this interface is called *Service*. The relation between *Service* and *VE* is called *Association*. The *Association* models which *VE* attributes can be read and modified via the specific *Service*. *Service* can be associated to any number of *Virtual Entities*.

It is practical to start the mapping between the architecture proposed in this paper and the IoT-A ARM from the VE concept because it is very similar in both models. In both IoT-ARM and the architecture proposed in this paper, the VE attributes both provide information about and allow interaction with physical world entities. The main difference is that the IoT-A ARM does not dictate how the VE should be represented or identified in practise (*i.e*. binary, XML, RDF, e*tc*.). In the proposed architecture VEs are represented with Semantic Web knowledge representation technologies and identified with *ucodes*.

The main difference between the IoT-A ARM and the architecture proposed in this paper is that the IoT-A ARM is service oriented whereas we propose information centric interaction model. To concretize, in IoT-A ARM the interaction between users and physical world is achieved via domain specific services that provide specific operations to read or modify certain VE attributes. In contrast the interaction with physical world in our architecture is achieved by modifying and accessing RDF graph(s) representing the current state of the physical world. However, if we think the SIB as a general purpose *Service* (from IoT-A ARM) which is associated with all the VEs (and their attributes) published into the SIB, the fundamental difference (*i.e*. service vs. information centric) between the architecture models becomes more a philosophical point of view than an actual difference.

There is no direct one-to-one representative for an *Agent* in the IoT-A ARM. The *Agent* can be seen as *Active Digital Artefact* because it invokes services provided by the SIB service. Additionally, since some *Agents* (*e.g*. sensor and actuator *Agents*) are responsible for exposing and interacting with the physical word entities, the *Agent* can be also modelled as a *Resource* hosted on a *Device*.

The *Resolution Infrastructure* does not have a representative in the IoT-A DM. This is because the *Resolution Infrastructure* is a functional component enabling the lookup and discovery of SIB services and VEs, and the DM is focused on presenting the main IoT concept - not functional components enabling the discovery of them. However, like mentioned earlier there are also other models in the IoT-A ARM and the Resolution Infrastructure maps directly to the Service Level Resolution component described in the IoT-A ARM Functional Model.

B. REFERENCE IMPLEMENTATIONS

The architecture presented in this paper is an outcome of an iterative process where different ideas and technologies have been developed and evaluated through several applications and reference implementations. In this section we will briefly describe these reference implementations and most notable applications.

1) SEMANTIC INFORMATION BROKERS

The first official SIB reference implementation is the Smart-M3 [42] based on the Piglet [43] RDFS++ database. It provides support for the original SSAP primitives and utilises non-standard XML-based encoding for RDF triples, called RDF-M3. For query and subscription it supports non-standard RDF query language and Wilbur Query language [44]. With the emergence of SPARQL (especially SPARQL 1.1 Query and Update Languages) the limitations in these query, subscribe and update operations became evident. Additionally, the performance and scalability of the Smart-M3 was not adequate for pervasive computing applications where it is required to react to changes in near real-time. To address these challenges, we have developed various SIB reference implementations including: RDF Information Base Solution (RIBS), OSGi-SIB and Red-SIB.

The RIBS is a secure and fast SIB implementation targeted for resource limited devices [45]. It is implemented with ANSI C programming language and the system dependencies are limited to Berkeley Software Distribution (BSD) style socket library and *select()* function. This makes RIBS easily portable to different computing platforms. RIBS manages memory resident bit-cube triple store in which RDF triple subjects, predicates and objects span the 3D vector space. The main advantage of the bit-cube triple store is that after URIs and RDF Literals are converted into bit-cube coordinates the access to the store becomes a simple random access lookup operation with constant latency. This is achieved with the side-effect of cubical memory consumption. Communication security and access control in RIBS are based on standards and their open source implementations. The selection of standard to be used during a session is negotiated between *Agents* and the RIBS at runtime. This makes it possible for *Agents* to use different security mechanism depending on their need. At information level RIBS uses a fine grained approach where access rights can be specified separately for each triple.

The OSGi-SIB is based on OSGi technology and takes inspiration from the principles presented by Manzaroli *et al.* in [46]. The main advantages of OSGi are modularity, expressivity and portability that simplify the deployment and maintenance of OSGi based systems. The OSGi-SIB uses Jena framework[11] with Pellet [47] to manage and reason over RDF graphs. Consequently, it supports SROIQ [48] based description logics (DL) and provides means for DL based backward chaining. Smart space system administrators can exploit these features to define what kind of reasoning their SIB executes. In addition to the DL based reasoning, the OSGi-SIB provides support for persistent SPARQL update rules. Because persistent SPARQL update operations can be expressed with KAMP it enables *Agents* to specify their own domain specific reasoning rules for a smart space with SPARQL 1.1.

The Red-SIB has been developed from the Smart-M3 reference implementation by substituting the Piglet with Redland RDF database and by improving performance and stability of the SIB [49]. The Red-SIB is available as open-source[12] and is consequently the most used SIB reference implementation. We have obtained decisive improvements to performance by optimizing the subscription processing in three ways. First, we use selective subscription processing that makes it possible to avoid unnecessary access to the RDF store when the inserted or removed triples cannot alter the SPARQL subscription results. This optimisation is used also in the OSGi-SIB. Second, the subscriptions processing is implemented in independent threads containing separate context triple stores for each subscription. This optimisation minimises the impact to the main triple store and allows parallel processing of subscriptions. This is especially beneficial in multi-core computing platforms that can assign separate cores for each subscription handler. Third, we have designed a SPARQL subscription algorithm that does not compute the whole result set but focuses only how the results have changed from last notification.

2) AGENTS AND APPLICATIONS

In this section we present seven smart space systems that we have developed following principles of the architecture presented in this paper. These systems include: Open-M3, Smart Greenhouse, Smart Building Maintenance, Smart Meeting, Smart Health Monitor, Home Garden and Smart Lighting.

[11]https://jena.apache.org/

[12]http://sourceforge.net/projects/smart-m3/

The Open-M3 was one of the first applications where we utilized semantic technologies to achieve interoperability in a pervasive computing environment [50]. It is a very simple system that contains two types of *Agents* deployed on various COTS devices; sensors that publish measurement information and viewers which display the information for the user. The interoperability between the sensor and viewer *Agents* is achieved by agreeing on common sensor ontology developed for the application.

The sensor ontology and sensor *Agents* developed for the Open-M3 are also used in the Smart Greenhouse [51]. When compared to the Open-M3, the Smart Greenhouse demonstrates a more complex pervasive computing system which has been developed incrementally by adding new *Agents* for specific purposes. The first version of the Smart Greenhouse consisted of *Actuator*, *Sensor* and *Gardener Interface Agents*. It enabled the gardener to view sensor measurements and control physical actuators manually via his personal device. In the second generation, the demonstration was expanded by introducing *Autocontrol Agent* which controls actuators on behalf of the gardener. The third generation of the Smart Greenhouse expanded the previous versions by making it possible to interact with the physical objects by touching. This interaction is based on *ucodes* stored into RFID tags.

The Smart Building Maintenance pilot [52], [53] deals with many topics relevant to the maintenance of large office buildings. The maintenance processes are managed and monitored by *Agents* with minimal human effort. In particular, the pilot focuses on detection and correction of faults occurring in office buildings. All the main actors in the scenario are taken in consideration from maintenance companies to building managers and maintenance operators. The developed *Agents* detect faults automatically and classify and associate them to correction procedures. For the maintenance operator the *Agents* provide seamless interfaces that enable the maintenance personnel to monitor faults, accept or refuse proposed correction procedures, and follow the accepted correction processes step by step. The workers in the area affected by a fault are promptly notified by *Agents* and provided with means to follow the status of fault correction. This way the waiting time of workers can be minimized. Additionally, procedures strictly connected to business and quality of service (*e.g.* the intervention requests are sent only to personnel skilled for that particular kind of problem) are solved by using semantic connections between maintenance personnel and different types of faults that can occur in the building.

The Smart Meeting application [54] demonstrates how a local meeting application is built using the interaction model proposed in this paper. In Smart Meeting users can share their contact information and files with each other via their personal *Agents* implemented to various mobile phones. The file sharing methods are not predefined but instead the *Agents* can advertise their files and then negotiate about the file transfers methods via the SIB. The semantic level interoperability is achieved by agreeing on a common ontology which describes concept such as meetings, participants, files, requests, and transfer protocols.

We have also applied the architectural principles in healthcare domain and developed a Smart Health Monitor system [55] that merges environmental and biomedical data to provide services for monitoring the wellbeing and health of people. The idea behind the Smart Health Monitor system is that health services based on telemonitoring can be improved by relating biomedical parameters of interest to their context and combining them with environmental data. To this end, we have implemented *Agents* for collecting environmental and physiological data such as room temperature, humidity, heart rate, skin temperature, respiration rate and activity index. Additionally, the system includes *Agents* that track the location of patients, refine data into more descriptive format (*e.g.* Thom Index), and provide means to monitor the wellbeing of patients in real time.

The Home Garden application continues the gardening theme in the applications. The system consists of two *Agents*: *Active Tag* and *Home Garden Agent*. The *Active Tag* was our first attempt to implement an *Agent* into extremely resource restricted computing platform [37]. The idea is that the user can equip her/his potted plants with *Active Tags* that blink a LED if the soil is too dry for a given plant and the user is present in the room. To do this the *Active Tag* utilizes information about its location, plant preferences and user presence published into the SIB by a *Home Garden Agent*. By using the WAX encoded SSAP format we managed to implement the *Active Tag* to a redwire LLC *Econotag* board (32-bit ARM7 platform) with total code size of 39.7 kB and RAM consumption of 25.3 kB. We also estimated an average current consumption of 241 μA at 60 s wake-up and communication interval which means an approximate battery duration of 1.3 years with two 1.5V, 2700 mAh alkaline batteries.

A more recent example of *Agents* in resource restricted computing platforms is presented in the Smart Lighting system. This demonstration consists of two low capacity *Agents*: battery operated motion sensor and light actuator. The motion detector is a simple *Agent* which detects motion with passive infrared sensor (PIR) and publishes the motion information into the SIB. A *Light Agent* is subscribed to the motion events and controls the lights based on this information. Additionally, the *Light Agent* enables other *Agents* to override this automatic control by modifying its representation inside the SIB. This way the system can be extended if the lights need to be used by other applications (*e.g.* by alarm system) in the future. Both *Agents* use KSP messages over Bluetooth Low Energy (BLE) radio in communication with the SIB. While the *Home Garden Active Tags* were implemented on 32-bit ARM7 platform, the *Light* and *Motion Detector Agents* are implemented on a lower capacity, 8-bit 8051 microcontroller with 128 kB of flash and 8 kB of RAM memory. For the motion sensor *Agent*, the flash memory consumption was 122.9 kB and RAM memory consumption was 6.3 kB. This means 11% and 3% increase in flash and RAM memories respectively when compared to the implementation without KSP protocol. The Bluetooth Low Energy stack consumed major part of the memories. A 1000 mAh capacity battery would last for 1.5 years in the motion detector when motion is sensed once per minute, or more than 10 years at 30 s interval if the sensor consumption was omitted (the PIR type motion sensor consumes 100 μA of current continuously).

C. PERFORMANCE EVALUATION

1) AGENT INTERACTION WITHIN SINGLE SMART SPACE

It is impossible to make a general performance evaluation for the *Agent* interaction. This is because the performance of the operations depends heavily on many factors (*e.g.* the complexity of the SPARQL WHERE patterns; the amount, the type and the organization of context information in the SIB; the computing platform the SIB is deployed on; *etc.*). However, to demonstrate that it is feasible to use semantic technologies to directly interact with sensors and actuators (as proposed in the architecture) we measured the performance in a typical scenario (the one presented in the section IV).

Round-trip latency is used as a measure for the performance and it is calculated from 100 iterations for each operation. In this evaluation we focused to the performance of the SIB (*i.e.* we did not want to measure the latency caused by the different connectivity technologies) and measured the latency taken to perform the *updates* and *subscriptions* used in the example scenario. For *update* the latency is defined as the time interval between the SIB receives an *update request* and the update operations has been performed. For *subscription* the latency is defined as the time interval between the SIB receives an *update* that triggers a *subscription* and the SIB is ready to send the notification to the *Agent*.

As a SIB we used the RedSIB reference implementation deployed on 64-bit Ubuntu 12.04 LTS virtual private server (VPS) with one processor unit and 4GB RAM reserved for each VPS. Dell Precision T5600 Server with two Inter Xeon E5-2665 processors (eight core, 2.4 GHz) and 64 GB RAM is used as a host computer with VirtualBox 4.3.6 as the virtualizing software.

To evaluate the scalability we measured the latencies with different amount of data in the SIB. Because latency of subscription to the fans to turn on depends on the amount of rooms the *Control Agent* needs to monitor this latency was also measured with different amount of rooms. The RDF triples presenting the temperature sensor, fan actuator, and the room (with temperature preference) were created as presented in the example (see section IV). All the other triples have been created randomly (*i.e.* it is assumed that they are produced by *Agents* not part of this application). Table II presents and Fig. 7 illustrates the latencies for operations with different amount of RDF triples in the SIB.

The SIB scales well respect to the amount of RDF triples as can be seen from the latencies that do not increase when the

TABLE 2. Latency (Ms) for agent interaction operations.

RDF Triples	update temperature	update fan state	subscribe fan state	subscribe fans (1 room)	subscribe fans (10 rooms)	subscribe fans (100 rooms)
1000	1,89	1,04	0,53	1,30	2,34	39,71
10000	1,79	1,05	0,53	1,36	2,31	40,30
100000	1,91	0,99	0,52	1,30	2,26	46,22
1000000	1,88	1,02	0,53	1,26	2,29	46,33

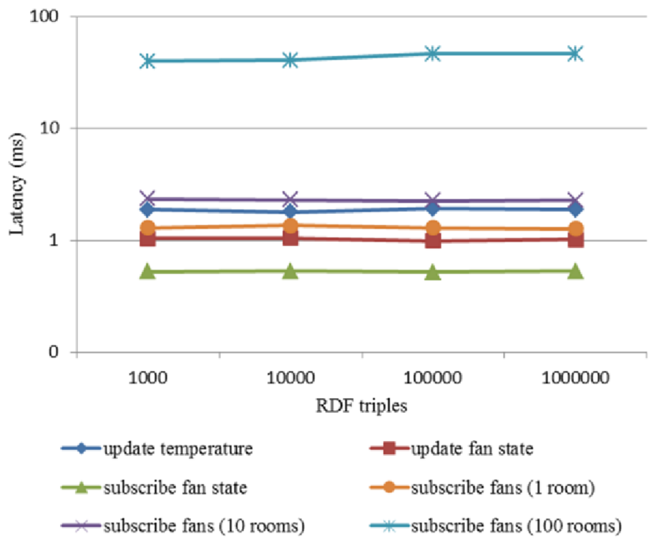

FIGURE 7. Latency for Agent interaction operations respect to the number of RDF triples stored into the SIB.

amount of triples is increased. For subscribe fans operations it can be seen that the latency increases significantly when the amount of rooms is increased. This is because the SPARQL subscription performed by the *Control Agent* needs to process more data as the variables in the SELECT query pattern receive more bindings when the amount of rooms (and sensors) is increased. However, even with 100 rooms the latency is still reasonable as the *Agent* will be notified on average 43 ms after the status has changed. For other operations the latencies are between 0,5 ms and 2,4 ms which is fast enough for typical IoT systems.

2) SEMANTIC INFORMATION BROKER LOOKUP AND DISCOVERY

To evaluate the performance and scalability of the *Resolution Infrastructure*, we measured the average latencies for SIB lookup and discovery operations with five different SIB service description counts: 1, 10, 100, 1000, and 10.000 services. We used similar SIB description to the one presented in the section V with the exception that instead of one class and property each SIB contained 10 different classes and properties.

The latency of the *discovery* operation is highly affected by the SIB service specification (*i.e.* the SPARQL query). To evaluate this effect we measured the latency for two different SIB specifications. In the first specification (used in *discovery 1*) only the service area and owner of the SIB are used and the SPARQL query is formulated as follows:

```
SELECT ?sib
WHERE {
  ?sib ssp:owner <http://www.vtt.fi/> ;
         ssp:serviceArea ?area .
  ?area geo:long ?long ;
            geo:lat ?lat .
  FILTER (xsd:float(?long) > 25.45580 &&
                xsd:float(?long) < 25.45600 &&
                xsd:float(?lat) > 65.05680 &&
                xsd:float(?lat) < 65.05700)
  ?sib a ssp:SIB
}
```

In the second SIB specification (used in *discovery 2*) information about the classes of interest is added. A SPARQL query that searches SIBs containing instances of both the example sensor and actuator classes can be presented as follows:

```
SELECT ?sib
WHERE {
  ?sib ssp:owner < http://www.vtt.fi/> ;
        ssp:containsClass ex:Sensor ,
                    ex:Actuator ;
        ssp:serviceArea ?area .
  ?area geo:long ?long ;
          geo:lat ?lat .
  FILTER (xsd:float(?long) > 25.45580 &&
                xsd:float(?long) < 25.45600 &&
                xsd:float(?lat) > 65.05680 &&
                xsd:float(?lat) < 65.05700)
  ?sib a ssp:SIB
}
```

The latency was measured by an *Agent* implemented with Python and deployed on Ubuntu virtual machine in HP Elitebook 8460p laptop with Core i5-2520M processor and 2 GB of RAM. The *TLD* and *SLD ucode Resolution Servers* responsible for the *lookup* run in virtual private servers on top of VirtualBox with 2 GB of memory reserved for each server. As a host computer we used Dell PowerEdge 2900 server with two Intel Xeon dual-core processors and Ubuntu 11.10 as the operating system. The RedSIB reference implementation was used as the *SIB Resolution Service* and it was deployed on Dell OptiPlex 755 with Intel Core 2 Duo vPro processor, 2 GB of RAM, and Ubuntu 12.10 operating system.

Table III presents and Fig. 8 illustrates latencies for SIB service *discovery* and *lookup* operations. To evaluate the performance and the actual latency caused by the Resolution

TABLE 3. Latency for resolution operations (ms).

SIB count	Lookup	Discovery 1	Discovery 2
1	192,14	67,94	68,43
10	192,51	67,50	69,11
100	191,16	71,21	77,85
1000	190,48	95,90	143,96
10000	192,66	315,13	808,15

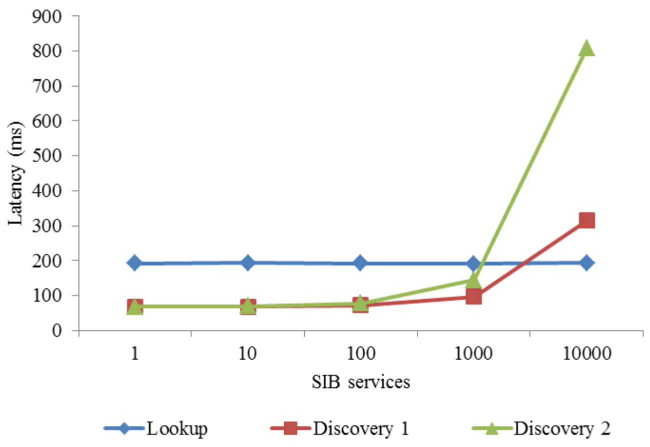

FIGURE 8. Latency for resolution operations with different amount of SIB service descriptions.

TABLE 4. Latency for resolution operations (ms).

Operation	Latency (ms)
Lookup	150,02
Discovery 1	65,11
Discovery 2	65,46

Infrastructure we measured also the latency caused by the network for each operation (depicted in Table 4).

The results show that the reference implementation scales quite well up to 1.000 SIB services. With 10.000 SIB service specifications there is a dramatic increase in latency of the *discovery* 2 operation. The latency of *discovery 1* increases also significantly, but it still scales better than the second operation where more complex SIB specification is used. This is because with simpler SIB specifications (*i.e*. simpler graph pattern) there are fewer bindings for the variables in the query pattern and the SPARQL query engine perform less computing.

In contrast to the *discovery* operation there is no notable increase in latency of the *lookup* even with 10.000 SIB service descriptions. The main reason for better scalability is that the *lookup* is quite simple process (compared to SPARQL query) where a *ucode* is mapped to a SIB address. Additionally, the *ucode Resolution Server* used for *lookup* is designed especially for scalability in mind.

Another important observation is that most of the latency for SIB resolution operations comes from the network (excluding *discovery* 1 and 2 with 10.000 SIB services). For instance, for SIB *discovery* with 100 SIB services the network latency is around 95% and 96% respectively. In SIB *lookup* the network latency part is not as high but still 78% of the whole latency. Since the actual processing executed by the *SIB Resolution Service* takes typically only around 3 ms it means that the it is able to process around 350 *discovery* operations per second (assuming 1.000 or fewer SIB services and similar SIB specifications to the test setup). In the *lookup* operation the *TLD server* part of the latency is around 41 ms which leads to 24 operations per second. However this does not take into account the fact that the *ucode Resolution Server* is distributed. In reality there can be up to 1.048.576 (*i.e*. 20 bits reserved for TLD server addresses) parallel *TLD servers* which could theoretically process even 25 million operations every second (assuming even distribution of *lookup* operations between the servers).

VII. CONCLUSION

In this paper novel semantic level interoperability architecture for pervasive computing and IoT was presented. The architecture design has been an iterative process in which we have followed the original Semantic Web vision and based the whole interaction model on semantic technologies. A key idea in the architecture is that the worldwide IoT is divided into smaller, more manageable smart spaces. The SIB is the central unit of a smart space and provides methods for agents to share semantic information with each other. The main difference with a SIB and normal SPARQL endpoint is that the SIB provides means both to monitor events in real time with SPARQL subscription and to update the context in the smart space with persistent SPARQL update rules. Agents can discover the SIB of interest by using either a simple lookup service that uses ucode attached to a physical object as a pointer to related information or a *SIB Resolution Service* with SIB specification as the discovery parameter.

To evaluate that the proposed architecture is in line with the IoT-A ARM we described how the main entities in the architecture map to the main ARM Domain Model entities. The architecture is also evaluated through reference implementations and numerous applications that we have developed according to the architecture principles. We also executed performance measurements for various resolution and agent interaction operations in an example IoT use case. The main observation from the performance measurements is that the *Agent* interaction operations scale very well and enable interaction with the physical world in real time fashion. For resolution operations most of the latency is caused by the network and the only scalability issues were measured for the *SIB Resolution Service* which had significant increase in the latency when SIB count was increased to 10.000 SIB services.

Common approach to enable semantic level interoperability by abstracting the heterogeneity of different devices, communication technologies and protocols is essential in order to enable 3rd party developers to create applications for future pervasive computing and IoT systems. The architecture described in the paper provides vital information and guidance for device manufactures and IoT system developers in this regard. The current reference implementations still need improvements before large scale IoT systems can be fully supported however. For instance, a more scalable implementation of the *SIB Resolution Service* is needed in the future. This could be implemented, for example, as a distributed hierarchical service where each *SIB Resolution Service* manages a certain geographical area in a similar way to the Domain Name System (DNS) architecture. Also approaches to manage resource access of applications with conflicting goals are needed to really enable 3rd party developers to create applications to our everyday living environments. In addition to the architecture and reference implementations there is also a need for tools that support development and deployment of devices and applications into the future IoT systems.

REFERENCES

[1] K. Ashton, "That 'Internet of Things' thing," *RFID J.*, Jul. 2009. [Online]. Available: http://www.rfidjournal.com/article/view/4986

[2] M. Weiser, "The computer for the 21st century," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, Jul. 1999.

[3] D. Saha and A. Mukherjee, "Pervasive computing: A paradigm for the 21st century," *Computer*, vol. 36, no. 3, pp. 25–31, Mar. 2003.

[4] A. Tolk, "Composable mission spaces and M&S repositories—Applicability of open standards," in *Proc. Simulat. Interoperability Workshop*, Washington, DC, USA, 2004, pp. 1–14.

[5] S. Pantsar-Syväniemi, A. Purhonen, E. Ovaska, J. Kuusijärvi, and A. Evesti, "Situation-based and self-adaptive applications for the smart environment," *J. Ambient Intell. Smart Environ.*, vol. 4, no. 6, pp. 491–516, 2012.

[6] A. Lappeteläinen, J.-M. Tuupola, A. Palin, and T. Eriksson, "Networked systems, services and information the ultimate digital convergence," in *Proc. 1st Int. NoTA Conf.*, Helsinki, Finland, 2008, pp. 1–7.

[7] D. Driscoll and A. Mensch. (Jul. 1, 2009). *Device Profile for Web Services Version 1.1*. [Online]. Available: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf

[8] Contributing Members of UPnP Forum. (Oct. 15, 2008). *UPnP Device Architecture 1.1. 136 p*. [Online]. Available: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

[9] (Oct. 2005). *OSGi Service Platform Release 4*. [Online]. Available: http://www.osgi.org/Release4/Download

[10] Z. Schelby, K. Hartke, and C. Bormann, (Aug. 28, 2013) "Constrained application protocol (CoAP)," CoRE Working Group Internet-Draft. [Online]. Available: http://datatracker.ietf.org/doc/draft-ietf-core-coap/

[11] IBM and Eurotech. *MQTT V3.1 Protocol Specification*. [Online]. Available: http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/MQTT_V3.1_Protocol_Specific.pdf, accessed Jun. 10, 2014.

[12] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Sci. Amer.*, May 2001 [Online]. Available: http://www.scientificamerican.com/article/the-semantic-web/

[13] R. Cyganiak, D. Wood, and M. Lanthaler, (Feb. 25, 2014) "RDF 1.1 concepts and abstract syntax," *W3C Recommendation*. [Online]. Available: http://www.w3.org/TR/rdf11-concepts/

[14] D. Brickley and R. V. Guha, (Feb. 25, 2014) "RDF schema 1.1," *W3C Recommendation*. [Online]. Available: http://www.w3.org/TR/rdf-schema/

[15] (Dec. 11, 2012) "OWL 2 web ontology language document overview," W3C OWL Working Group. [Online]. Available: http://www.w3.org/TR/owl2-overview/

[16] P. Liuha, J. Soininen, and R. Otaolea, "SOFIA: Opening embedded information for smart applications," in *Proc. Embedded World*, Nuremberg, Germany, 2010, pp. 1–8.

[17] N. Koshizuka and K. Sakamura, "Ubiquitous ID: Standards for ubiquitous computing and the Internet of Things," *IEEE Pervas. Comput.*, vol. 9, no. 4, pp. 98–101, Oct./Dec. 2010.

[18] S. Harris and A. Seaborne, Eds., (Mar. 21, 2013) "SPARQL 1.1 query language," *W3C Recommendation*. [Online]. Available: http://www.w3.org/TR/sparql11-query/

[19] P. Gearon, A. Passant, and A. Polleres, Eds., (Mar. 21, 2013) "SPARQL 1.1 update," *W3C Recommendation*. [Online]. Available: http://www.w3.org/TR/sparql11-update/

[20] J. Kiljander, F. Morandi, and J.-P. Soininen, "Knowledge sharing protocol for smart spaces," *Int. J. Adv. Comput. Sci. Appl.*, vol. 3, no. 9, pp. 100–110, 2012.

[21] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data—The story so far," *Int. J. Semantic Web Inf. Syst.*, vol. 5, no. 3, pp. 1–22, 2009.

[22] A. Bassi *et al.*, *Enabling Things to Talk: Designing IoT Solutions With the IoT Architectural Reference Model*. Heidelberg, Germany: Springer-Verlag, 2013.

[23] M. Eisenhauer, P. Rosengren, and P. Antolin, "A development platform for integrating wireless devices and sensors into ambient intelligence systems," in *Proc. SECON Workshops*, 2009, pp. 1–3.

[24] R. Masuoka, B. Parsia, and Y. Labrou, "Task computing—The Semantic Web meets pervasive computing," in *Proc. 2nd Int. Semantic Web Conf. (ISWC)*, vol. 2870. 2003, pp. 866–881.

[25] S. Ben Mokhtar, N. Georgantas, and V. Issarny, "COCOA: Conversation-based service composition in pervasive computing environments with QoS support," *J. Syst. Softw.*, vol. 80, no. 12, pp. 1941–1955, 2007.

[26] G. Thomson, S. Bianco, S. Ben Mokhtar, N. Georgantas, and V. Issarny, "Amigo aware services," in *Constructing Ambient Intelligence*, vol. 11. Berlin, Germany: Springer-Verlag, 2008, pp. 385–390.

[27] S. Zhexuan, A. A. Cárdenas, and R. Masuoka, "Semantic middleware for the Internet of Things," in *Proc. Internet Things (IOT)*, Nov./Dec. 2010, pp. 1–8.

[28] D. Martin *et al.*, (Nov. 22, 2004) "OWL-S: Semantic markup for web services," *W3C Member Submission*. [Online]. Available: http://www.w3.org/Submission/OWL-S/

[29] J. Soldatos, N. Dimakis, K. Stamatis, and L. Polymenakos, "A breadboard architecture for pervasive context-aware services in smart spaces: Middleware components and prototype applications," *Personal Ubiquitous Comput.*, vol. 11, no. 3, pp. 193–212, 2007.

[30] L. Rosenthall and V. Stanford, "NIST smart space: Pervasive computing initiative," in *Proc. IEEE 9th Int. Workshops Enabling Technol.: Infrastruct. Collaborat. Enterprises (WET ICE)*, Jun. 2000, pp. 6–11.

[31] H. Chen, T. Finin, A. Joshi, L. Kagal, F. Perich, and D. Chakraborty, "Intelligent agents meet the Semantic Web in smart spaces," *IEEE Internet Comput.*, vol. 8, no. 6, pp. 69–79, Nov./Dec. 2004.

[32] X. Wang, J. S. Dong, C. Y. Chin, S. R. Hettiarachchi, and D. Zhang, "Semantic space: An infrastructure for smart spaces," *IEEE Pervas. Comput.*, vol. 3, no. 3, pp. 32–39, Jul./Sep. 2004.

[33] D. Pfisterer *et al.*, "SPITFIRE: Toward a semantic web of things," *IEEE Commun. Mag.*, vol. 49, no. 11, pp. 40–48, Nov. 2011.

[34] P. Charlton, R. Cattoni, A. Potrich, and E. Mamdani, "Evaluating the FIPA standards and their role in achieving cooperation in multi-agent systems," in *Proc. 33rd Annu. Hawaii Int. Conf. Syst. Sci.*, vol. 2. 2000, p. 10.

[35] J. Takalo-Mattila, J. Kiljander, M. Eteläperä, and J.-P. Soininen, "Ubiquitous computing by utilizing semantic interoperability with item-level object identification," in *Mobile Networks and Management*, vol. 68. Berlin, Germany: Springer-Verlag, 2011, pp. 198–209.

[36] J. Ziegler, M. Graube, and L. Urbas, "RFID as universal entry point to linked data clouds," in *Proc. IEEE Int. Conf. RFID-Technol. Appl. (RFID-TA)*, Nov. 2012, pp. 281–286.

[37] A. Ylisaukko-Oja, P. Hyttinen, J. Kiljander, J.-P. Soininen, and E. Viljamaa, "Semantic interface for resource constrained wireless sensors," in *Proc. Int. Conf. Knowl. Eng. Ontology Develop. (KEOD)*, 2011, pp. 505–511.

[38] C. Ishikawa, (Apr. 2012) "A URN namespace for ucode," IETF Network Working Group. [Online]. Available: https://tools.ietf.org/html/rfc6588

[39] T. Berners-Lee, R. Fielding, and L. Masinter, (Jan. 2005) "Uniform resource identifier (URI): Generic syntax," IETF Network Working Group. [Online]. Available: http://www.ietf.org/rfc/rfc3986.txt

[40] R. Battle and D. Kolas, "Enabling the geospatial semantic web with parliament and GeoSPARQL," *Semantic Web*, vol. 3, no. 4, pp. 355–370, 2012.

[41] D. Brickley and L. Miller, (Jan. 14, 2014) "FOAF vocabulary specification 0.99," *Namespace Document*. [Online]. Available: http://xmlns.com/foaf/spec/

[42] J. Honkola, H. Laine, R. Brown, and O. Tyrkko, "Smart-M3 information sharing platform," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2010, pp. 1041–1046.
[43] O. Lassila, "Programming Semantic Web applications: A synthesis of knowledge representation and semi-structured data," Ph.D. dissertation, Dept. Comput. Sci. Eng., Helsinki Univ. Technology, Espoo, Finland, Oct. 2007.
[44] O. Lassila, "Generating rewrite rules by browsing RDF data," in *Proc. 2nd Int. Conf. Rules Rule Markup Lang. Semantic Web*, 2006, pp. 51–57.
[45] J. Suomalainen, P. Hyttinen, and P. Tarvainen, "Secure information sharing between heterogeneous embedded devices," in *Proc. 4th ECSA*, 2010, pp. 205–212.
[46] D. Manzaroli *et al.*, "Smart-M3 and OSGi: The interoperability platform," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2010, pp. 1053–1058.
[47] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semantics: Sci., Services and Agents World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.
[48] I. Horrocks, O. Kutz, and U. Sattler, "The even more irresistible SROIQ," in *Proc. 10th Int. Conf. Principles Knowl. Represent. Reason. (KR)*, 2006, pp. 57–67.
[49] F. Morandi, L. Roffia, A. D'Elia, F. Vergari, and T. S. Cinotti, "RedSib: A smart-M3 semantic information broker implementation," in *Proc. 12th Conf. FRUCT Assoc.*, 2012, pp. 86–98.
[50] M. Eteläperä *et al.*, "Open-M3: Smart space with COTS devices," in *Proc. 12th Int. Conf. Ubiquitous Comput. (UbiComp)*, 2010, pp. 363–364.
[51] J. Kiljander, M. Eteläperä, J. Takalo-Mattila, and J.-P. Soininen, "Opening information of low capacity embedded systems for smart spaces," in *Proc. 8th Workshop Intell. Solutions Embedded Syst. (WISES)*, 2010, pp. 23–28.
[52] A. D'Elia *et al.*, "Smart applications for the maintenance of large buildings: How to achieve ontology-based interoperability at the information level," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2010, pp. 1077–1082.
[53] S. Pantsar-Syvaniemi *et al.*, "Case study: Context-aware supervision of a smart maintenance process," in *Proc. IEEE/IPSJ 11th Int. Symp. Appl. Internet (SAINT)*, Jul. 2011, pp. 309–314.
[54] J. Kiljander, M. Eteläperä, J. Takalo-Mattila, J.-P. Soininen, and K. Keinänen, "Autonomous file sharing for smart environments," in *Proc. 1st Int. Conf. Pervasive Embedded Comput. Commun. Syst.*, 2011, pp. 191–196.
[55] F. Vergari *et al.*, "A smart space application to dynamically relate medical and environmental information," in *Proc. Design, Autom. Test Eur. Conf. Exhibition (DATE)*, 2010, pp. 1542–1547.

JUSSI KILJANDER is currently a Research Scientist with the VTT Technical Research Centre of Finland, Espoo, Finland. He received the M.Sc. (Tech.) degree from the University of Oulu, Oulu, Finland, in 2010. His current research and the Ph.D. studies are focused on ubiquitous computing, Internet of things, and device interoperability with semantic web technologies. He has authored over 10 scientific papers, and contributed to several research projects related to semantic interoperability and pervasive computing. Before his graduation studies, he was a Seasonal Trainee in Nokia, Finland.

ALFREDO D'ELIA was born in Catanzaro, Italy, in 1981. He received the master's (*summa cum laude*) degree from the University of Bologna, Bologna, Italy, in 2006, and the Ph.D. degree in information technology from the University of Bologna in 2012, where he was a Research Assistant. He worked in several European projects, such as SOFIA, CHIRON, Internet of Energy, and Arrowhead. He also collaborated with important industries, like Telecom Italia, Rome, Italy, and was an Intern in Nokia, Finland, for six months, where he was a co-inventor of a patent. He also teaches with the University of Bologna and Cesena as an Assistant Professor, where he is responsible for didactic modules. His main topics of interest in research are semantic web, interoperability, information representation techniques, software architecture, and system characterization and optimization.

FRANCESCO MORANDI was born in Lugo, Italy, in 1984. He received the degree in electronic engineering and the Specialist degree in telecommunication engineering from the University of Bologna, Bologna, Italy, in 2006 and 2009, respectively. In 2010, he was with Fondazione Ugo Bordoni, Bologna, as a Consultant for the digital television transition. Since 2011, he has been a Researcher with the Advanced Research Center on Electronic Systems, University of Bologna. His current research is focused on semantic technology for interoperable platforms applied on Internet of things.

PASI HYTTINEN is currently a Senior Scientist with the VTT Technical Research Centre of Finland (VTT), Espoo, Finland. His research topics are programming and management of complex computer systems, such as smart spaces, Internet of things, and clouds. During his nine years at VTT, he was involved in three EU research projects iCore (2012–2014), IoT-A (2012–2013), and SOFIA (2009–2011), and the Customer RF Front-End Processor Project (2006–2008). Before his research career at VTT, he had 10 years experience in industrial research and development. He was with Flextronix (2001–2005), Medikro (2000–2001), Honeywell (1995–2000), and the University of Oulu, Oulu, Finland (1993–1994), where he received the M.Sc. (Tech.) degree in 1994. Before his graduation studies, he was an Entrepreneur and a Freelance Programmer.

JANNE TAKALO-MATTILA is currently a researcher with the VTT Technical Research Centre of Finland (VTT), Espoo, Finland. He received the M.Sc. (Tech.) degree from the University of Oulu, Oulu, Finland, in 2009. His current research is focused on smart environments, networked embedded devices, and Internet of things. From 2008 to 2014 at VTT, he has been a technical expert and project manager in various projects, including the European Union research projects, joint research projects, and contract research projects.

ARTO YLISAUKKO-OJA received the M.Sc. degree in electrical engineering. He has been a Research Scientist with the VTT Technical Research Centre of Finland, Espoo, Finland, since 1999. His research topics include low-power electronics, wireless sensor networks, semantic sensor interfaces for resource constrained devices, near-field communications, and wireless charging.

JUHA-PEKKA SOININEN (M'99) is currently a Research Professor of Computing and Computer Architectures with the VTT Technical Research Centre of Finland (VTT), Espoo, Finland. He received the M.Sc., Lic.Tech., and D.Sc. (Tech.) degrees from the University of Oulu, Oulu, Finland, in 1987, 1997, and 2004, respectively. He has been a Research Scientist with VTT since 1988, Senior Research Scientist since 1996, and Research Professor since 2007. He was the leading expert in various large research projects with VTT, from 1993 to 2011, including contract research projects, joint research projects, and the European Union research projects. His current research deals with ubiquitous and distributed computing, system architectures, platform-based design methodologies, system architecture evaluation methods, and system-level design methods. He has been a reviewer of several international conferences, journals, and books. He has authored over 70 scientific publications.

TULLIO SALMON CINOTTI was born in Bologna, Italy, in 1950. He received the degree in electrical engineering from the University of Bologna, Bologna, in 1974. He is currently an Associate Professor with the School of Engineering and Architecture, University of Bologna, where he is also in charge of courses on computer architecture, logic design, and interoperability of embedded systems. He has a long-standing experience on joint research with primary Italian and international academic, research, and industrial institutions. He has co-authored papers written with researchers from Intel Labs, Hillsboro, OR, USA, Nokia Research Center, Sunnyvale, CA, USA, Siemens Corporate Technology, Princeton, NJ, USA, Telecom Italia Laboratory, Turin, Italy, VTT, the Polytechnic of Milano, Milan, Italy, and the University of Kent, Canterbury, U.K. His research interest is focused on embedded systems and semantics-based data distribution architectures for environment-driven multidomain ecosystems. He is the Coordinator of the University of Bologna participation to the European research initiatives in the areas of open cultural heritage, smart environments, and electric mobility.