RDF Recipes for Context-Aware Interoperability In Pervasive Systems

Anna M. Kosek*, Aly A. Syed* and Jon M. Kerridge†
*NXP Semiconductors, Corporate Innovation and Technology/Research
High Tech. Campus 32, 5656 AE Eindhoven, The Netherlands
{anna.kosek,aly.syed}@nxp.com
†Edinburgh Napier University, School of Computing
Merchiston Campus, Edinburgh EH10 5DT, United Kingdom
j.kerridge@napier.ac.uk

***Abstract*—Nowadays home automation systems integrate many devices from security system, heating, ventilation and air conditioning system, lighting system or audio-video systems. Every time a new device is installed, problems with connecting it to other devices and synchronization may appear. The technology trends are to build more powerful and functional new autonomous devices, rather than improve device synchronization, data and functional distribution and adaptation to changing environment. This paper highlights interoperability problems in pervasive computing and presents a solution for devices with limited processing capabilities by use of an ontology for knowledge formulation and semantic interoperability.**

***Index Terms*—ontology, semantic interoperability, knowledge representation.**

I. Introduction

The number of electronic devices surrounding us increases every day, many devices are already deployed in the environment and are hidden from peoples' sight. Ongoing device miniaturization makes it possible to manufacture smaller devices, therefore more of them can be embedded in one space. A network of many small devices requires a flexible approach to pervasive configuration and management.

A well known problem in pervasive environments is how to achieve interoperability between components in the system. In distributed systems, there are three classic interoperability levels: platform, programming language and service interoperability [1]. Service interoperability divides further in to signature, protocol and semantic levels [1]. The signature interoperability problems are associated with the syntax of the interface of the service and are successfully solved by popular language interface specifications like CORBA's IDL (Interface Definition Language) [2] or WSDL (Web Service Definition Language) [3]. Many network interoperability problems (at the protocol level of interoperability) can be addressed by Internet protocols, however these do not solve interoperability problems on a semantic level [4]. When considering many multi-vendor devices and components, solutions for semantic interoperability based only on standards do not scale [5]; flexible and updatable standards are needed.

A. Ontologies in Interoperability

An ontology is a representation of a set of concepts, functions and relations between concepts [6]. Concepts are classes of entities existing in a domain, functions and relations are respectively one-to-one and one-to-many properties existing in the environment. An ontology, together with syntax and semantics provides a language that is necessary for proper communication. Building an ontology is a difficult task and because domains and language always evolve it is impossible to state that the ontology describes all possible concepts and relations in a specific domain. Because an ontology can always evolve and adapt to an ever-changing domain, it can be a very flexible and up-to-date description of the domain. Therefore an ontology can play the role of a flexible standard and can be used to achieve semantic interoperability.

It has been shown that interoperability can be achieved by using an ontology in consumer electronics devices [7]. Another project, Sofia (Smart Objects for Interactive Applications) [8], is targeted to enable and maintain cross-industry interoperability for smart spaces using a core ontology based on Dolce [9] and domain ontologies. The approach presented in this paper is an extension of both [7] and Sofia concepts to create a pervasive environment with interoperability at the semantic level and re-usable knowledge representation based on an ontology. The specific environment that this project is considering is a network of very small, energy-frugal devices with limited processing capability. Services that are offered by these devices are simple, therefore it is possible to represent them using an uncomplicated knowledgebase drawn from an ontology.

B. Knowledge Representation and Ontology

In artificial intelligence (AI) domains a key to building powerful and large systems is capturing knowledge in a computer-readable form. Representing knowledge is a complex and time-consuming

task. Therefore knowledgebase construction remains one of the biggest costs when constructing an AI system [10]. Most complicated AI systems require building a knowledgebase from scratch. An ontology can offer a core structure for representing knowledge and enables reuse of knowledgebases when designing systems in similar domains. Neches [10] points out that a knowledgebase should be constructed from many levels of knowledge. These layers are increasingly specialized, therefore layers' division begins with very general concepts, based on the ontological model and finishes with very specialized and less reusable knowledge [10]. Therefore a branch of an ontology that is used for constructing knowledgebases should consist of general (core), domain-specific and device-specific concepts and relations. The knowledgebase in a device or component should consist only of necessary concepts, depending on a component's functions and domain as described in [7]. The knowledgebase can be extended if new functionalities or services appear. The most specialized part of knowledgebase are instances and data associated with a particular device or component.

C. Context-Aware Pervasive System

The solution for interoperability and knowledge representation in the pervasive system presented in this paper is based on building a knowledgebase based on an ontology. It is important to fix the ontology model, while the ontology content can be changed when new classes and relationships appear. The ontology can be modified by domain experts and can be kept in a repository accessible by both software and hardware manufacturers.

Because of the use of a common language described by an ontology, the interoperability is achieved on a semantic level [7]. Components from the system use concepts that are provided by the ontology, therefore understanding is achieved at a high level of abstraction. This way it is possible to build a context-aware system, that can represent and share understanding of context. Context can be associated with environment conditions, presence of particular devices or people, or periods and events associated with a calendar. Context-aware systems benefit from well defined context, and as every device or component understands how the context influences its own domain, the system can produce emergent behavior.

In pervasive computing a smart space is understood as a cloud of devices existing and cooperating in this space. In an architecture described in [7] all configurations between devices are established automatically, data and functionalities are distributed and available for different devices to enable cooperation. Devices can appear or disappear from a particular space at any time, so devices have to be able to work autonomously or form subnetworks called virtual devices [7]. A device can participate in many virtual devices; virtual devices are created for a particular task and can be destroyed if a different configuration is requested. A virtual device can be created by a user request or some scheduled task. Devices can join or leave a virtual device at any time; this operation can influence the behavior of a virtual device. The way a virtual device is formed is described in the knowledgebase by use of a list of actions to perform. Devices can use any protocol to form a virtual device, the protocol specification is described in a device's knowledgebase. Therefore depending on configuration, a device will have clear instructions how to react to a message from a particular protocol.

II. Knowledgebase Structure

A knowledgebase consists of layers of knowledge (Figure 1). Layers are divided into T-Box and A-Box. T-Box consists of class taxonomy and property definitions that we name in this paper ontology models, like Dolce [9] or any other upper ontology, A-Box contains instances corresponding to classes described in T-Box.

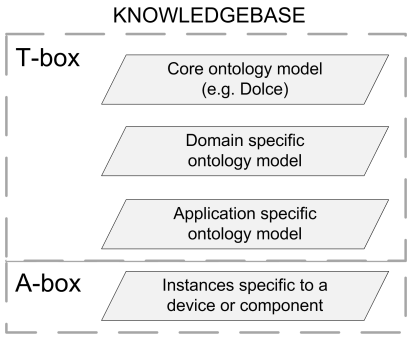

Fig. 1. Layered structure of a knowledgebase.

The top level of the knowledgebase is a meta model of the core ontology. The meta model of the ontology has to be fixed, this project is following the Sofia approach by choosing Dolce as a core ontology. Dolce is an upper ontology that addresses very general domains [9]. The next layer in Figure 1 is a domain ontology that is created for a specific area. For example if the device is a sensor, its domain ontology describes classes and properties that are associated with sensor domains. This way the knowledge of the surrounding world is narrowed to the domain the device or component belongs to. The next layer, Application specific ontology model, is a model for instances that describes a device's behavior, capabilities and services that the device can provide. In the last layer of the knowledgebase, categorized as an A-Box, contains instances describing a particular device according to rules provided by T-Box ontology models. Similar devices will have similar T-Box part of the knowledgebase, but different instances in A-

Box. T-Box can be automatically generated and used when building A-Box for a particular device.

A. Resource Description Framework

The knowledgebase structure can be fixed using RDF (Resource Description Framework) [11] triples. RDF is a framework for representing information in the World Wide Web. RDF can describe a simple graph-based data model [11]. An RDF triple consists of subject, predicate and object, where the subject and predicate are resources defined by an URI (Uniform Resource Identifier) while the object is a literal or a resource. The structure of the knowledgebase will consist of triples:

$$(Subject, Predicate, Object)$$

For example:

$$(apple\_tree, is-a, tree)$$

This triple uses predicate *is-a* to associate subject *apple_tree* with object *tree*. Ontologies are designed by domain experts and represent universal and domain-specific truths.

For the purpose of the World Wide Web, ontologies are represented by use of a markup scheme like XML. Unfortunately XML adds complexity to database format by introducing tags that need to be interpreted. XML is not designed for data storage or efficient retrieval of data [12] the additional time to process data exists due to parsing and text conversion [13]. For devices with a limited processing capability it is important to keep any database structure as simple as possible. Therefore the knowledgebase format is kept to the simple three field structure of RDF.

RDF for World Wide Web can use many external files of different ontologies, therefore it is not essential to define all concepts in one RDF file. In many cases referencing by use of URL to external ontology files is not necesarry. Devices from pervasive systems usually have a well defined functionality and often do not need to connect to the Internet. Therefore entries in a knowledgebase need only reference other entries in it.

The A-Box part of knowledgebase consists of data necessary for the system to make decisions, process requests, interpret messages, learn and react to commands. A knowledgebase consists of six parts:

1) Device description: Description of the device using the ontology. This part of the knowledgebase will consist of two parts: a core ontology necessary to characterize the device universally, and a domain specific ontology describing entities and relations that are restricted to one type of device (for example: lighting, sensors, multimedia devices).

2) Device capabilities: The description of device capabilities that are provided by the device manufacturer. For example, capabilities are simple tasks that a device can physically perform. A simple lamp can only turn on, turn off and dim to some level. It is impossible for lamp to play music or accept coins because these tasks are beyond a lamp's physical capabilities.

3) Context and users: The context describes the actual situation of the whole system, or part of it. Every device can understand contexts that are described in its own knowledgebase. Contexts might be *night*, *lunch_time*, *reading* or *watching_tv* and are associated with environment conditions or actions performed by users. Understanding context and acting upon it is one of the most important tasks of the proposed system. Context is a very broad concept and customization for different users is important. For example *watching_tv* for *Anna* means that their personal phone should be off, window blinds shut and light dimmed. The same context can mean something different to *Tomas*, he wants his phone in loud mode, lights turned off, blinds closed. Devices participating in one context can be the same but settings are different. Reasons for this are different preferences and priorities for particular users. The system presented in this paper, built on concepts from [7], does not have formal central control, so contexts are saved on devices that are required to perform some part of the function required in a context. Therefore the context *watching_tv* for *Anna* in a lamp or in a TV specify totally different actions.

4) Configuration and state: The information about the state and configuration of a device is very important and should be accessible by the device. For example, to work as a group, devices need to be formed in a virtual device [7]. The configuration holds information in which virtual devices the device participates and what is the current state of the configuration. A device that is a part of a virtual device should keep on functioning without any breaks, if a new request arrives a device has to be able to determine if it should drop its current task and respond to the request, by checking its current configuration settings. The state of a device is also included in the knowledgebase. For example a lamp can store information about its dim level or color in the knowledgebase where it can be easily accessed.

5) Recipes: Recipes are used to describe a procedure for the behavior of every device. Recipes are very detailed and specify the procedure that a device has to follow for a given context. The idea is to describe all important steps in a recipe, so the device only has to read the recipe and act upon it. Recipes are associated with context, people and request types. Since devices forming a pervasive systems have limited resources, the number of requests, that a device can serve, is also limited. Therefore a device can decide itself if it can execute recipe or not. The concept of a recipe is a main subject of this paper and is described in the next section.

B. Recipes

Recipes in a knowledgebase are sequential and represent steps that a device should follow in a particular virtual device. Every recipe has a header to describe the recipe's type, context and other conditions that are guarding access to this recipe. The recipe shown in Figure 2 consists of *n* steps. Steps contain actions that need to be performed in this particular step. If the device fails to finish a step, the recipe is dropped.

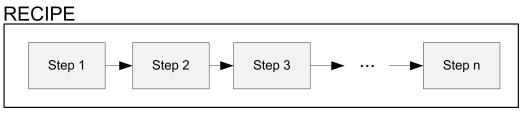

Fig. 2. Recipe structure.

Recipes are described using ontology terms, so the concept of a recipe has to be present in the ontology model. Classes and properties associated with a structure of a recipe are described in the T-box part of the knowledgebase in Application specific ontology model layers are shown in Figure 1.

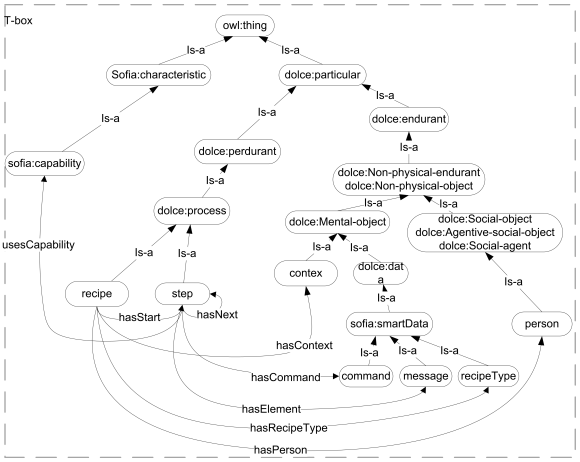

Fig. 3. An ontological view of the concept of a recipe in T-box.

A simple example of an ontological model to represent recipes combined with Dolce and Sofia models is presented in Figure 3. This model is representing recipes and steps as a subclass of Dolce processes. Property *hasStart* associates recipe with a first step from this recipe, and property *hasNext* creates a sequence of steps in a recipe.

Let's consider a lamp and a switch that can cooperate when grouped in one virtual device. The mechanism to group and configure these devices requires one device to start a configuration by sending a search message. When a device receives a search message it checks if it desires to participate in a virtual device that the message propagates and if it has a recipe that it can use in this particular configuration. If so, the recipe is executed, information about the configuration is saved; device sends an acknowledgment massage back and acts upon the instructions provided by this recipe. If a switch initiates a virtual device with a lamp, lamp has to have a recipe that describes, in detail, how to act upon a particular request. A recipe consist of header, that describes a recipe and steps that device has to perform. The header consist of recipe type, service and context that recipe is associated with. A recipe can be associated with a person and customized for this person.

subject	predicate	object
recipe1	is-a	recipe
recipe1	hasContext	Reading
recipe1	hasPerson	Anna
recipe1	hasRecipeType	configureVD
recipe1	hasStart	step1
step1	is-a	step
step1	usesCapability	startConfiguration
step1	hasNext	step2
step2	is-a	step
step2	usesCapability	output
message1	is-a	message
step2	hasElement	message1
message1	hasMessageType	2
message1	hasAckResponse	0
step2	hasNext	step3
step3	is-a	step
step3	usesCapability	deviceFunctionality
step3	hasCommand	turnOnOff
step3	hasValue	1
step3	hasNext	step4
step4	is-a	step
step4	usesCapability	deviceFunctionality
step4	hasCommand	dim
step4	hasValue	70
step4	hasNext	null
null	is-a	step

Recipe's **harder**. This recipe is associated with context reading and person Anna.

Step 1 saves all data needed for a particular configuration, therefore creates entries in local memory about the virtual device that the lamp is about to join.

Step 2 uses capability output to send an acknowledgment to a device that requested joining a new virtual device, in this case this message is sent to light switch.

Step 3 sends command to turn on the light.

In **step 4** lamp dims to 70 percent. A step called null indicates end of recipe. If a device reach an end of a recipe, a task is finished.

Fig. 4. Example of a recipe in knowledgebase, expressed in RDF triples.

An example of a recipe for a lamp expressed in RDF format is shown in Figure 4. The *recipe1* consists of a header and four steps. Steps in a recipe represents a sequence of actions that the device has to perform. If a device reach the end of a recipe, a task is finished.

A device that interprets a knowledgebase needs to be extended with a Semantic Interpretation component (Figure 5) and use a communication link to receive and send messages necessary to establish virtual devices and cooperate with other devices. To influence a device's functionality, the Semantic Interpretation component uses the Device Interface component.

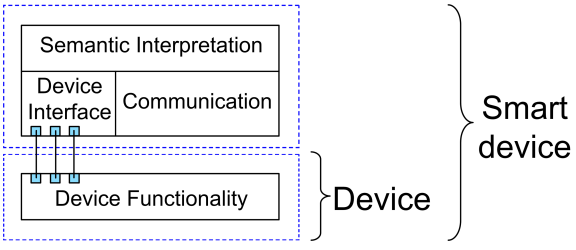

Fig. 5. Device's architecture.

The addition of components specified in Figure 5 and a device specific knowledgebase enables semantic interoperability between devices in a smart space. Recipes are used to inform a device what exact steps

it should undertake in a particular context or when receiving data from different devices. Since devices may have many recipe , choice on which recipe to run is made by the device using the content of the received message. If there are more than one recipes that are usable in a certain situation, the the message content is used to further refine a recipe choice. This is done by matching RDF triples of the recipe to fields of the message. This way we can make conditional choices on which recipe to run and there by achieve different device behavior.

III. Implementation

The smart lighting system chosen to illustrate a context-aware distributed system consists of a light switch, different kinds of light sources and sensors (Figure 6). The switch has a User Interface (UI) and it can accept requests from the user. Light sources are different types of lamps, with different light emission source types. There can be many sensors present in the space: light sensors, motion sensors, devices recognizing users' identity (for example RFID readers) and any other sensors that can be beneficial for the system. As shown on Figure 6 the system has no central control. The request comes from the UI embedded in light switch, but this does not make the light switch a controller, it takes the role of a request center. All messages sent in the system are broadcast messages, every device receives the message and decides if it should accept or discard it.

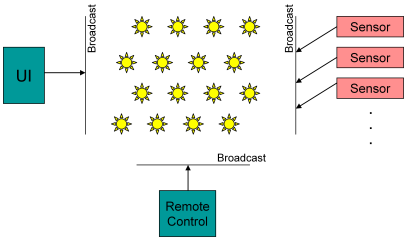

Fig. 6. The lighting system architecture.

Any device that can increase or reduce light level in the room may consider itself as a provider of a lighting service. Therefore lamps are not the only source of the illumination in a space but also window blinds that can block light from outside or a mirror that reflects exterior light onto the ceiling are also light sources. The idea is to integrate and engage all of those elements of the system that can influence illumination in a particular space. Applying this approach can save energy by using natural light, rather than using only light sources.

The simulation was implemented in JCSP (Communicating Sequential Processes for Java) [14]. This language supports concurrent programming, it is useful when simulating devices running simultaneously. Devices are presented as CSP processes and communication links are implemented using CSP channels. Use of JCSP enables simulating a network of many devices running in parallel and communicating by sending messages.

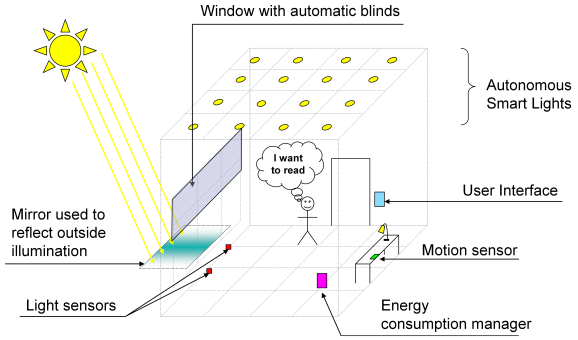

Fig. 7. Proposed scenario.

The scenario chosen to describe the system is presented on Figure 7, which shows a model of a room that is used as an office. The space is equipped with a light switch with UI, ceiling lamps, lamp on a desk, automatic blinds on the window, mirror used to reflect daylight to illuminate the room, light and motion sensors. The user is the center of attention and can set preferences manually in the UI. The user can set lights manually or choose a context available in a particular space. When the user indicates a need to control light levels or the context changes, the switch performs a search for devices that provide service: lighting for this user. All the lamps, blinds and mirror responds to this request, because all of them are able to provide light for the space.

The other task that the proposed set-up can perform is to keep the light at the same level in the whole room, when the outside light conditions change. This configuration uses communication between light and sensors to use as much natural light as it is possible, while attaining a stable level of light in the room.

IV. Results and Conclusions

A complex system consisting of sensors and actuators was designed and implemented. The system controlling a space consists of autonomous devices collaborating to achieve lighting conditions desired by the user. There are sixteen ceiling lamps, two light sensors, light switch, mirror and user defined contexts. The context plays a crucial role in the system, as it influences behavior of devices to suit different users needs.

The simulation is run on PC with Intel Core2 CPU 6300@1.86GHz, 1GB RAM and Windows XP OS. Java code with JCSP libraries is run to simulate devices working in parallel, knowledgebases are binary files containing RDF triples.

There are two available contexts in the system: reading and meeting, devices have different recipes for different contexts.A lamp in a simulation consists of Semantic Interoperability and Communication

components, device functionality and knowledgebase binary file. A lamp has 214 triples in its knowledgebase and 5 recipes that are responsible for different actions. During configuration of a virtual device for context reading, the knowledgebase in the lamp is accessed 305 times and local memory, used to keep temporary data, is accessed 806 times.

The simulated scenario consisting on 20 devices runs 148 Java threads. Average number of messages sent per configuration is 53 and messages received by a device is 954. All messages are broadcast, every device receives all messages, but most of them are dropped and do not trigger any actions.

The simulation presented in this paper runs in real time in a users perception. The goal of this simulation is is to show functional behavior of a distributed system using RDF based recipes. The numbers mentioned here should be treated as a reference. we do not attempt to to compare these numbers to another implementation.

The simulation is a proof of concept that a pervasive system can be constructed out of autonomous devices without any central control. Recipes are described in previous sections and in [7] can made in RDF format and influence the device's functionality and behavior. The interoperability is achieved at a semantic level by the use of ontologies. Ontologies are also used to construct a knowledgebase that can be reused or transferred and understood in different devices.

V. Further work

Recipes are built according to a model taken from common ontology, therefore devices can process any recipe following these rules. The next step in developing the architecture is to include an extension that allows new recipes to be added to existing knowledgebase in devices from newly installed devices. Adding recipes enables updatability, both forward and backward. When a new device is added to a network, it can update existing devices with a recipe that will guide devices how to use a new device or component.

The adaptation to a changing environment, set of devices in a network and user's needs can be done by adding a new recipe to a knowledgebase. More sophisticated technique can update and adapt recipes themselves, learning to adjust devices behavior depending on situation and improving recipes that are in device's memory.

Acknowledgment

The authors would like to acknowledge Perada, EU network of Excellence, for supporting knowledge exchange. This work is being carried out in the Sofia project [8].

References

[1] T. Strang and C. Linnhoff-Popien, "Service interoperability on context level in ubiquitous computing environments," L'Aquila, Italy, 2003.

[2] R. Orfali, D. Harkey, and J. Edwards, *The essential distributed objects survival guide*. John Wiley and Sons, Inc., 1995. [Online]. Available: http://portal.acm.org/citation.cfm?id=210712

[3] E. Christensen, F. Curbera, F. Curbera, G. Meredith, and S. Weerawarana.

[4] N. Georgantas, "Semantics-based interoperability for pervasive computing systems," 2008.

[5] D. O'Sullivan and D. Lewis, "Semantically driven service interoperability for pervasive computing," in *Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access*. San Diego, CA, USA: ACM, 2003, pp. 17–24.

[6] T. Gruber, "Ontolingua: A mechanism to support portable ontologies," Stanford University, Knowledge Systems Laboratory, Tech. Rep., 1992.

[7] A. A. Syed, J. Lukkien, and R. Frunza, "An ad hoc networking architecture for pervasive systems based on distributed knowledge," in *Proceedings of Date2010, Dresden*, 2010. [Online]. Available: http://www.date-conference.com/proceedings/PAPERS/2010

[8] Sofia, "Smart objects for intelligent applications," 2009. [Online]. Available: http://www.sofia-project.eu

[9] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari, "WonderWeb deliverable d18," 2001.

[10] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. Swartout, "Enabling technology for knowledge sharing," *AI Magazine*, vol. 12, no. 3, pp. 36–56, August 1991.

[11] O. Lassila and R. R. Swick, "Resource description framework (rdf) model and syntax specification," 1998.

[12] L. Khan and Y. Rao, "A performance evaluation of storing XML data in relational database management systems," in *Proceedings of the 3rd international workshop on Web information and data management*. Atlanta, Georgia, USA: ACM, 2001, pp. 31–38. [Online]. Available: http://portal.acm.org/citation.cfm?id=502939

[13] R. Bourret, "XML and databases," 2005. [Online]. Available: http://www.rpbourret.com/xml/XMLAndDatabases.htm

[14] P. Welch and N. Brown, "Communicating sequential processes for java (jcsp)," 2009.