A Framework for Context-Aware Applications for Smart Spaces

M. Mohsin Saleemi, Natalia Díaz Rodríguez, Johan Lilius, and Iván Porres

Turku Centre for Computer Science (TUCS)
Department of Information Technologies, Åbo Akademi University
Turku, Finland
{msaleemi,ndiaz,jlilius,iporres}@abo.fi

Abstract. This paper presents an approach for developing context-aware intelligent applications for Smart Space-based infrastructure. The goal is to model and process context information using our development tool and Nokia's Smart-M3 architecture. We propose an adaptable and scalable context ontology, an ambient computing framework based on Smart Spaces and a rule based reasoning to infer high level context. Our approach deals with key issues in context aware ubiquitous computing such as adaptive and proactive changes in the environment, incorporation of novel sources of context information and automatic code generation from the context ontology to provide seamless interoperability.

1 Introduction

Context consists of any information that can be used to characterize the situation or state of an entity [7]. Entities can include anything e.g. a person, a physical object, an application or a device that is used to interact with the user. The concept of context has very broad view which can essentially consider anything as a context such as physical objects, applications, environment and the users. For example, physical context of a person can include his location, time etc; his social context can include his social relations e.g. family and friends etc, his activity context can include any tasks he performs in his daily life (watching TV, listening music, talking on phone, etc). In pervasive and context-aware computing, a user should be able to readily accomplish an action which possibly can include cooperation and collaboration with others using multiple devices and networks as he moves in the environment. In this way, a whole new universe of intelligent applications would automatically adapt to the user's intention. For example, your smart phone could notice that your favorite program starts in 5 min. based on your profile information or a fan page on Facebook and the TV guide available on the broadcaster's web page. Then it could use GPS to find that you are not at home and deduce that it needs to start the PVR (Personal Video Recorder) at home. This kind of intelligent applications need the context information from different sources to adapt to the user's preferences without involving human interactions. The context-aware intelligent applications can be realized by exposing the context information, internal data and functionality of

S. Balandin et al. (Eds.): NEW2AN/ruSMART 2011, LNCS 6869, pp. 14–25, 2011.

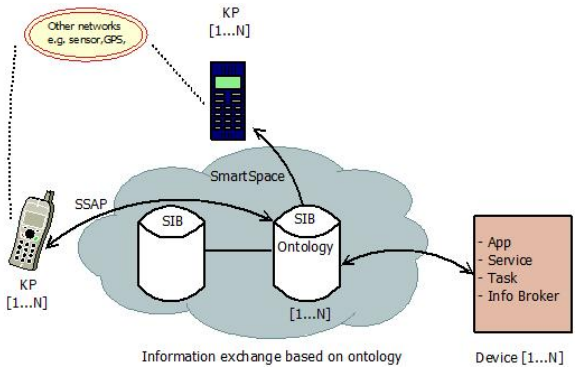

Fig. 1. Smart-M3 Architecture

the devices and ensuring data interoperability between them. This requirement is due to the variety of devices to be used and the need for interacting with each other within the context.

To enable the above mentioned cross-domain scenario and to solve the interoperability issue, one way is through the notion of *Smart Space*. A Smart Space is an abstraction of space that encapsulates both the information in a physical space and the access to this information allowing devices to join and leave the space. In this way the Smart Space becomes a dynamic environment whose identity changes over time when a set of entities interact with it to share information. For example, communication between the mobile phone and the PVR in the above scenario does not happen point-to-point but through the Smart Space whose members are the mobile phone and the PVR. We have developed a programming interoperability solution for rapid application development in Smart Spaces that can be extended to support context-aware intelligent applications [10]. Our solution is based on Nokia open source Smart-M3 architecture [4], an ideal choice for developing pervasive applications as it includes: 1) A blackboard software architecture which is cross-domain, cross-platform and enables knowledge share and reuse. 2) Ontology governance process (information stored in RDF) ensuring seamless information interoperability.

2 Smart-M3 Architecture

The Smart-M3 (Multi part, Multi device and Multi vendor) architecture [12][4] provides a particular implementation of Smart Space where the central repository of information is the Semantic Information Broker (SIB). The Smart-M3 space is composed of one or more SIBs where information may be distributed over several SIBs for the later case. The devices see the same information, hence it does not matter to which particular SIB in a M3 space a device is connected. The information is accessed and processed by entities called Knowledge Processors (KPs). KPs interact with the M3 space by inserting, retrieving or querying

information in any of the participating SIBs using access methods defined by the Smart Space Access Protocol (SSAP). Smart-M3 provides information level interoperability to the objects and devices in the physical space by defining common information representation models such as the Resource Description Framework (RDF). Since Smart-M3 does not constrain to a specific structure of information, it enables the use of ontologies to express the information and relations in a KP application providing multi domain support. Figure 1 shows the overall Smart-M3 architecture.

3 Application Development for Smart-M3

We use an ontology-driven development approach for Smart-M3 for mapping ontologies to Object Oriented Programming[3][10]. Our approach consists of two parts. The first part is the generator that creates a static API from an OWL ontology. This mapping generates native Python classes, methods and variable declarations which can then be used by the KP developer to access the data in the SIB as structured and specified in the OWL ontology. The second part is the middleware layer which abstracts the communication with the SIB. Its functionality is the handling of RDF Triples (*Subject, Predicate, Object*) with the generated API. This consists of inserting, removing and updating Triples and committing changes to the Smart Space. It also provides functionality for synchronous and asynchronous queries. Our approximation enables application developers to use the generated API to develop new KPs and applications without worrying about the SIB interface as the generated API takes care of the connection to the SIB each time an object is created.

In this application development approach, the concept of application is not the traditional control-oriented application running on a single device but a number of independently operated KPs which may run on different devices and are grouped together to be perceived as a single application. For instance, chat, calendar synchronization and multi-player games are examples of applications using this approach where a set of KPs, each handling a specific task, run on multiple smart devices and coordinate and interact with each other through the SIB to make a complete application. This coordination between KPs is done in the form of data exchange through the SIB where KPs subscribe to or query for specific data to perform a specified task. Application ontologies are used to describe data in the SIB and directs the KPs to access and manipulate data related to their functionality.

4 Context Ontology Model

Context information can be modeled using a variety of approaches such as key-value models, graphical models, object oriented models, logic based models and ontology based models [5]. We have chosen to use ontology based context modeling because of several reasons. Firstly, as our Smart-M3 architecture provides an interoperability solution based on ontology models, we can benefit from automatic code generation and ontology reasoning. Secondly, ontologies are the most

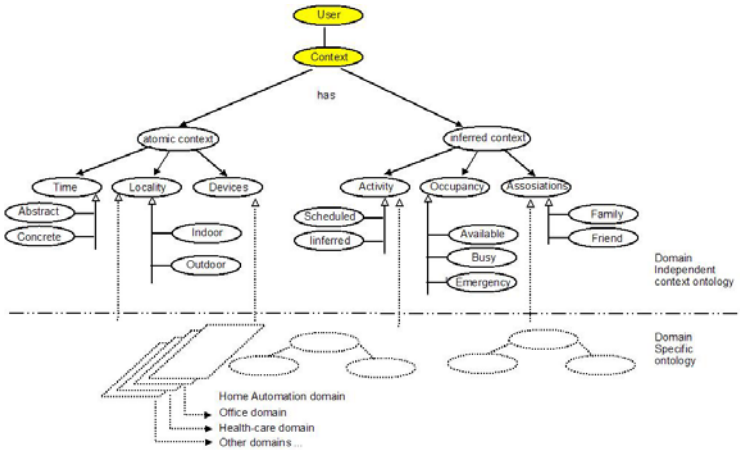

Fig. 2. Context Ontology of a user

promising and expressive models [5] fulfilling requirements for modeling context information. Thirdly, ontology based models provide advantages of flexibility, extendibility, genericity and expressiveness which are key factors in context-aware systems.

Information about the user's context is significant if enables the ambient system and applications to adapt to the user's preferences. In this paper we refer to all information that characterize the situation of a user as his *context*. In order to make the system more adaptive to the user's behavior, we propose to use multiple dimensions as the context of a user. Figure 2 shows these dimensions in an example context ontology. We divided the user's context in two broad categories, namely *atomic context* and *inferred context*. Atomic context refers to the context data acquired directly from context providers. The sources of atomic context can be any source providing relevant information to describe a user's situation. The inferred context refers to the information deduced from the given context data. We modeled user's context using six context dimensions: *Time*, *Locality*, *Devices*, *Activity*, *Occupancy* and *Associations*. Although this is not the only set, we believe that it is enough to capture most of the concepts. The user's context can be fully described in most of the domains by using these dimensions.

As the context ontology defines the basic terms and relations related to the user context that are applicable in every possible domain, we have defined it as an independent layer as shown in Figure 2. The user interactions involve a number of devices and appliances available in the environment which make the context dimensions of the ontology consider their activities and associations as domain independent types. The upper ontology in Figure 2 represents the core concepts to model user's situations in the environment and it appears feasible to have an unified upper context ontology capable of dealing large communities of users in wide range of domains. The lower part of Figure 2 shows the domain specific ontologies which describe concepts related to the domain in question by

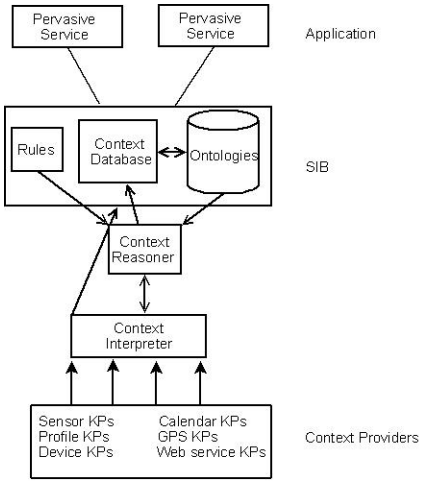

Fig. 3. System architecture

enumerating the concepts presented in the upper context ontology. For example, activity is a domain independent type of context but the tasks which are performed under this concept in the office domain such as *meeting, presenting* etc. are different when compared with tasks in the home-automation domain. Each specific domain has its own definition of the user's activity in that particular domain and even for the particular applications in that domain.

The system can thus, deduce information not explicitly given in the ontology, e.g. if the user is in the living room and the TV is ON then it implies that the activity is watching TV. Similarly, the system can deduce that the user is busy if he is talking on the phone, even if his calendar shows no activity at that time. In this way, by using the context information from different dimensions, the system can adapt to the user's current behavior and make the decisions rather dynamically.

5 System Architecture

Figure 3 shows the overall architecture of our context-aware system that supports pervasive applications from different domains. It consists of the following components.

Context Providers: A range of context provider KPs give atomic information about the user's context. Context providers cover from low level data obtained from sensors, GPS, RFID, WiFi, etc. to data from web services or user profiles. Atomic context information from these providers is used to infer new context information at higher level using inference rules and the context reasoner. The system provides two levels of inferences. At the first level, atomic context information infers new information while in the second the inferred context information is used to infer higher-level information.

Context DataType Interpreter: Context information from different data sources has diverse chronological features and data formats. The system needs a type conversion to map the value of one input type to another value of another input type to allow new and innovative information sources to be used. E.g. the temperature sensor's integer value at a given timestamp must be converted to the actual temperature in Celsius. GPS coordinates should be mapped to give the corresponding building number. The context interpreter is responsible for converting raw data to meaningful context information that can be put into ontology for use by other components of the system. In this way the context ontology is extended automatically. For this purpose all atomic data sources should specify their functionality in terms of input and output data types as well as other meta information such as how long is the validity of its data, how accurate is the data, its nature i.e data is sensed or defined etc. This can be done by using Web Ontology Language for Services (OWL-S) [1] which has capability to specify characteristics and functionalities of all the information sources. OWL-S would facilitate Context DataType Interpreter to easily map data values from heterogeneous devices.

Context Reasoner/Rules Interpreter: This module is responsible for inferring new higher level context from given atomic context information. The context reasoning is based on inference rules defined by KP developers which are then provided to the Python Rule Module. Also other information sources and inference rules can be provided as separate libraries. This module may trigger the execution of rules based on the current context information which in turn infers new contexts. It enables the context-aware system to be tailored for specific application scenarios. For example, if a user is in his bedroom, the bed sensorOn is true and the lightOn is false, the reasoner can infer that the user's activity is *sleeping* and put this inferred context information in the Semantic Information Broker. The context reasoner can also infer context properties using ontology reasoning by specifying inter-ontology relationships. This means inferring hierarchy classes and properties related to objects in the context (a sensor is attached to a sofa and the sofa is in the living room; thus, the sensor is in the living room).

Ontologies: OWL ontologies define the context. They represent both data directly sensed from context providers and information inferred from this data using inference rules. The context ontology describes generic concepts related to the individual and consists of atomic information from the information sources, inferred information from the atomic information using the inference rules and the inference rules. The core context ontology of an individual is extended when the new inferred information using the inference rules is added to it as shown in Figure 4.

Inference Rules: The inference process by the reasoner requires a set of inference rules that are used to infer new context related to an individual. These inference rules are created using a predefined format (section VI) and provided to the system as a set of imported libraries. The rules are domain specific and deterministic and the rule interpreter uses them to combine instances of context to infer new context information.

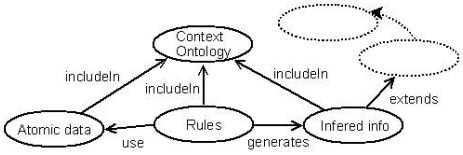

Fig. 4. Ontology extension based on Inferred context

5.1 Inference Rules and Context Reasoning

In context-aware ubiquitous systems where the emergence of increasing number of devices are used to perform desired services, we need to impose control constraining the participating devices' behavior. Rules can define how to react when a phone rings during a meeting; how to handle multiple requests to play different channels on a single TV at the same time; how to infer the user's activity using the active context information from multiple KPs. The inference rules, based on logic programming, allow context information origination from the provided set of ontologies. Its evolution/adaptation is caused by KPs taking part in the application. Figure 4 shows the ontological model after inferring context information. Dotted ellipses represent the application specific part of the context ontology.

We have defined an inference rule using a 3-clauses pattern:

With clause models declarations and assertions.

When clause contains events that trigger the rule.

Then clause includes conclusions representing inferred information after the rule is triggered.

Following there are few examples which illustrate our approach to define inference rules. The first rule states that if the user is in room B4050 at a specific time 13:20, and the room is occupied between 13:00 to 15:00 having more than one person there, then the user's activity is inferred as *busy in a meeting.*

```
with U:- User(id="1", role="Student", name="Mohsin"),
     R:- Room(room No.="B4050", location= "ICT"),
     P:- projector (id="101", type="ProModel")
when U.locatedIn (R, atTime"13:20"),
     R.occupied("13:00-15:00"),     P.locatedIn:- R,
     R.number of people > 1, P.statusON
then U.busyInMeeting
```

The inferred context information can then be used by another rule to infer other level information or to perform a task when some event is triggered. For example, the following rule states that when the user is in the meeting then forward incoming calls to his voice mail without interrupting him.

```
with U:- User(id="1", role="Student", name="Mohsin"),
     Ph:- phone(id="10", type="Iphone", model="4G")
when Ph.incomingCall, U.busyInMeeting, Ph.owner:- U
then Ph.activeVoicemail
```

There might be some emergency cases when the user does not want to ignore incoming calls. The following rule overwrites the result of the previous one when the calling person is the user's wife giving a beep to the user's phone. The user's relationship with the caller can be obtained from existing ontologies given to the system, such as Friend of a Friend (Foaf) ontology in this case.

```
with U:- User(id="1", role="Student", name="Mohsin"),
     Ph:- phone(id="10", type="Iphone", model="4G"),
     C:- Caller(name="Samra", association="wife")
when Ph.incomingCall,  U.busyInMeeting,
     Ph.owner:- U,  U.relation:- C
then Ph.beepOnce
```

The following rule infers based on the user's context information that the user is out for lunch.

```
with   U:- User(id="1", role="Student", name="Mohsin"),
     Res:- Restaurant(name="Unica", location= "ICT")
when ResOpen:- true, U.locatedIn(Res, atTime"12:15"),
     timeInterval > 5 min
then   U.havingLunch
```

A set of rules are to be specified for each domain. The rules above represent office domain. Based on the context information from multiple context dimensions, the system triggers, when certain changes happen, the activation of the associated rules.

6 Implementation

We use Python's meta-programming features to enable writing Python code which includes logic programming statements representing inference rules. These are inspired from the event-condition-action (ECA) rules model which is a technology from active databases for supporting dynamic functionality [9]. The interpreter for the inference rules is in its early stage of development.

6.1 Development Framework

The first task is defining a script language to use OWL2 allowing the user to express rules as section V showed. The second task is the integration of those logic expressions to work with ontologies into a functional Object-Oriented language. Because of its versatility, metaprogramming opportunities and ease of prototyping (easy to learn and use) we chose Python. Thus, given a context, the programmer could define in a simple way and beforehand the underlying rules that pervasively help the user daily in his Smart Space.

The third task is the integration of first and second approaches with the *Smart-M3 Ontology to Python API Generator* framework [3], which makes more intuitive to the programmer the definition of pervasive applications. This tool

provides automatic generation of a Python API for every OWL class as well as setters and getters among other methods to interact effortlessly with the common SIB through which all KPs communicate with each other.

In this section we focus on the second and third tasks and its motivation. Given the functionality provided by the Smart-M3 Ontology-Python framework, there is a need for designing a rule syntax language that allows users -with basic programming skills- easy definition of rules to model DIEM applications. In this way the need for learning OWL or query languages is minimized or null.

6.2 Programming Knowledge Processors in Python

The main feature of the Python Rules Module is to encapsulate, acting like a bind, the SIB interface. Our implementation approach is inspired by *Pythologic, Prolog syntax in Python* [2].

A Rule is structured as follows:

```
With() |= When() >> Then()
```

- **With()** Clause represents *assumptions* about existence of individuals.
- **When()** Clause represents *conditions*, when the KP must execute.
- **Then()** Clause represents *actions* to trigger.

In this way, the application programmer does not deal with RDF Triples directly but mainly with logic Python expressions. Therefore, the programmer could embed into Python code expressions like:

```
1  condition1 = lambda: user.isBusy()
2  condition2 = lambda: room.getOccupied()
3  conditions = [condition1, condition2]
4  action =  lambda: user.setVoiceMail(True)
5  myRule = With([user, room]) // When(conditions) >> Then(
       action)
6  diem.addRule(myRule)
```

Listing 1.1. Rule definition with Python Rules Module

The underlying implementation of the Python Rules Module translates Python logic expressions to the SIB API main interface: *Query, Subscribe, Insert, Remove, Update*. Thus, the Python Rules Module just needs to be imported to be used with the KP class where the DIEM application is coded:

- **With()**: If instances in *With()* exist in the SIB (SIB-Query), proceeds to evaluate *When()*. The check includes the ontology's Python object declaration, this is, other KPs know about it.
- **When()**: If *When()* is true (SIB-Query), executes *Then()*. If not, sets a SIB-Subscription to the attributes in *When* clause. The subscription capability provided by the Smart-M3 SIB allows knowing when the value of certain attribute has changed so that the rule can be evaluated again avoiding, in this way, unnecessary infinite query loops or traffic bottlenecks.

– **Then()**: If *With()* & *When()* satisfy, executes *Then()*, which translates into SIB-Update/ SIB-Add/ SIB-Remove/ SIB-Unsubscribe (results may update RDF Triples).

A Knowledge Processor can be located e.g. in any smart phone or device and can be for example a phone application for getting the local temperature from the Internet or a sauna/thermostat activator. All the KPs can be created and connected to the Smart Space (called 'x' in this example) in the following way:

```
7     def main(args):
8     app = QtGui.QApplication(sys.argv)
9     smartSpace = ('x', (TCPConnector, ('127.0.0.1', 10010))
          )
10    KP = PhoneKP.create(smartSpace)
11    # Definition of Rules
12    sys.exit(app.exec_())
```

Listing 1.2. KP Programming and Connection to the Smart Space 'x'

Straight after the KP is created, the user could define the Python rules related to the existing KPs. Then, connect the KPs to the Smart Space and run them is the only thing left.

If `EmptyKP.py` (provided by the Ontology-Python Generator) is used, instance declarations will automatically translate to insertions of Triples into the SIB. This allows other KP applications connected to the same Smart Space to know about those individuals' existence to interact with them. In the Python Rules Module, every KP application contains a `TripleStore` instance (produced by Ontology-Python Generator) representing the Smart Space' SIB. At last, the *With()*, *When()* and *Then()* clauses translate its Python statements to one of the implementation options given by the Ontology-Python Generator. These are SIB calls in *RDF* or *WQL* language. Our approach shows that learning OWL or query languages is not needed for interconnections with the SIB and interactions with other devices' KPs.

7 Related Work

The research in context-aware computing provides a wide number of context-aware systems and approaches for application development. Starting with context modeling, there are plenty of different points of view, but since the ontology model wins the rest regarding simplicity, flexibility, extensibility and expressiveness, we focus in comparison with ontology based systems and similar systems to ours. A good compendium of pros and cons in relation to design architectures and context models [5] shows that, already in 2004, similar advanced ontology-based context models were presented.

CoBrA and SOCAM are some of them using their own OWL-based approach for context processing while others like Context Managing Toolkit describe context in RDF. CoBrA [6] is proposed as an agent-based infrastructure for context modeling, context reasoning and knowledge sharing using context acquisition

components. The Soupa and CoBrA-Ont Ontologies are some of the related tools. They also provide techniques for user's privacy control. SOCAM [8] introduces another architecture for building context-aware services focused on information sensing and context providers. A central server or context interpreter gains context data through distributed context providers and processes it before sending it to the clients. These and other projects as [14] focus basically on creating ontologies for context-representation but they do not intend to build a framework for creating location- and context-aware development of services or applications based on the semantic back end.

In [13], the authors extend typical operating system concepts to include context-awareness. Gaia presents a similar representation to our RDF Triples with 4-ary predicates (the 4th one is context-type) and it does not use OWL but DAML + OIL. Gaia's MVC model also differs from our blackboard architecture. In [11] and [15], the authors presented a framework that targets only smart phone platform and technologies.

Context Toolkit [7] presents an approach to enable application development by using reusable components. However, its attribute-value tuples are not clearly meaningful enough making the application programming restricted. All these systems use SQL to access the central database. Contrasting to our RDF/WQL queries, we makes the queries to be restricted to a smaller set of statements.

With the purpose of facilitating the creation of services we can see that diverse technology-specific frameworks have been created but none of them outcomes with a clear functional programming tool. In comparison to the previous systems, our approach for modeling and processing context addresses the challenge of context-aware ubiquitous computing in smart environments using automated ontology code generation for Python. And our idea is expressive enough to represent domain specific context as abstract context data.

8 Conclusions and Future Work

In this paper, we expressed our ideas for context-aware applications for Smart Spaces. We presented our contextual ontology for modeling context information and the overall system architecture. The structure and syntax of inference rules are described with office domain scenario. We conclude that Smart Spaces are well suited for ambient applications to adapt to the user's preferences because they can provide information about the physical environment which can be shared and reused by many dynamic applications. In the future, we aim to implement a context manipulation library for our Smart-M3 tool to process contextual information. Next challenges to be tackled are e.g. consistency related issues as the insertion of individuals' properties into the SIB for the whole Smart Space could create ambiguity if not checked. Moreover, extra functionality to efficiently implement subscriptions to individuals' attributes is needed. After the basic functionality is consistent, new environments's use cases could be applied.

Acknowledgment. The research work presented in this paper is funded through the ICT-SHOCK DIEM project.

References

1. OWL-S Services, http://www.daml.org/services/owl-s/
2. Pythologic, Prolog Syntax in Python, http://code.activestate.com/recipes/303057-pythologic-prolog-syntax-in-python/
3. Smart-M3 Ontology to Python API Generator, http://sourceforge.net/projects/smart-m3/files/smart-m3-ontology_to_python-api_generator_v0.9.1beta.tar.gz/
4. Smart-M3 software at sourceforge.net, release 0.9.4beta (May 2010), http://sourceforge.net/projects/smart-m3/
5. Baldauf, M., Dustdar, S., Rosenberg, F.: A Survey on Context-Aware systems. International Journal of Ad Hoc and Ubiquitous Computing 2 (2007)
6. Chen, H., Finin, T., Joshi, A.: An Ontology for context-aware pervasive computing environments. In: Proceedings of the Workshop on Ontologies in Agent Systems (2003)
7. Dey, A.K., Abowd, G.D.: The Context Toolkit: Aiding the development of context-aware applications. In: Workshop on Software Eng. for Wearable and Pervasive Computing (2000)
8. Gu, T., Pung, H.K., Zhang, D.Q.: A middleware for building context-aware mobile services. In: Proceedings of IEEE Vehicular Technology Conference (2004)
9. Bailey, P.T.W.J., Poulovassilis, A.: An event-condition-action language for XML. In: Proceedings of the 11th International Conference on World Wide Web (2002)
10. Kaustell, A., Saleemi, M.M., Rosqvist, T., Jokiniemi, J., Lilius, J., Porres, I.: Framework for Smart Space Application Development. In: Proceedings of the International Workshop on Semantic Interoperability, IWSI (2011)
11. Matetelki, P., Pataki, B., Kovacs, L.: Service-oriented context-aware framework. In: Young Researchers Workshop on Service-Oriented Computing (2009)
12. Oliver, I., Honkola, J.: Personal semantic web through a space based computing environment. In: Proceedings of the 2nd Interntional Conference on Semantic Computing (2008)
13. Roman, M., Hess, C., Cerqueira, R., Ranganathan, A.: A middleware infrastructure for active spaces. IEEE Pervasive Computing (2002)
14. Wang, X.H., Zhang, D.Q., Gu, T., Pung, H.K.: Ontology based context modeling and reasoning using OWL. In: Workshop Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications (2004)
15. Jin, Y., Her, Y., Kim, S.-K.: A context-aware framework using ontology for smart phone platform. International Journal of Digital Content Technology and its Applications 4(5) (2010)