Dynamic Ontology Enrichment and Reasoningin AmI Environments

Aitor Almeida[1] Diego Lopez-de-Ipiña[2] Unai Aguilera[1]
Iker Larizgoitia[1] Xabier Laiseca[1] Pablo Orduña[1]
Ander Barbier[2]

Abstract. Context in an environment usually suffers from sudden changes; people and devices move from one place to another, previously unknown objects appear and existing ones disappear. AmI-enhanced environments must be flexible enough to adapt and respond to these changes. This work describes a semantic infrastructure which manages context, reasons over it and, more significantly, it is capable of dynamically enriching the ontology that models context.

1 INTRODUCTION

The management of environment and user context is one of the key aspects of Ambient Intelligence (AmI). Without context knowledge, an intelligent environment cannot adapt itself to user needs. This explains why management of context should be one of the main activities in any Ambient Intelligence system. Smartlab is a semantic middleware platform that aims to facilitate this task while making explicit the previously hidden context. Semantic technologies are used to represent context and to infer from it reactions to be taken. The use of a context ontology allows Smartlab to integrate information coming from heterogeneous sources and to share it easily.

The most remarkable feature of the SmartLab middleware is its ability to dynamically extent the context ontology with new context-modelling concepts and rules provided by non-previously available devices within an environment. Dynamic discovery and installation of previously non-available semantic devices is automatically performed by this middleware. Furthermore, its context ontology is enriched taking into account the semantic knowledge and behavioural information in the form of rules provided by these newly discovered devices, which were not taken into account in the design phase.

The following sections describe the context managing and semantic reasoning mechanisms used by this middleware. In section 2 previous work on semantic context management is discussed. Section 3 explains the system architecture and the dynamic context enrichment mechanisms it offers. Section 4 describes the Smartlab Context Ontology created to model the context. In section 5 the semantic reasoning capabilities of this middleware over the context are explained. Finally in section 6 conclusions are given and possible future work is suggested.

2 RELATED WORK

In recent years some projects [1][2][3] have created ontologies that model the context. SOUPA [4] is a set of ontologies

oriented to ubiquitous and pervasive applications used by the COBRA project[1]. It is composed by two sub-sets SOUPA Core and SOUPA Extensions. The SOUPA Core defines the elements that are present in any ubiquitous application while SOUPA Extensions gives support to more specific applications. CONON [5] is used by the SOCAM [2] project to model the context of pervasive computing applications. It is also divided in to sets, one with the general information shared between all the applications and the other one domain specific. CODONT [6] is used by the CODAMOS [3] project and its main aim is to create an adaptable and flexible infrastructure for AmI applications. These three ontologies have in common some similar elements (see Table 1) like information about location, time, people/users, actions/activities and devices. There are also some elements unique for each system, like the information about the environmental conditions, events, policies and services.

Table 1. Context modeling ontologies

	SOUPA	CONON	CODONT
Similar elements	Person, Agent, BDI, Policy, Event, Action, Time, Space	Location, Person, Activity, Computing, Entity	User, Service Platform, Location, Time, Environmental condition

While all these ontologies are easily extensible, this process takes place offline. In Smartlab this extension of the ontology is done dynamically, on the fly, when a new device is discovered in the environment. The device is able to add new concepts to the ontology creating new classes that are able to model new context information. This new concepts are checked before adding them to the ontology. Moreover new domain rules are also added to the system with the new devices, adapting the system reactions to the new knowledge.

This mechanism enables the system to be more flexible and adaptable. wOther frameworks [7] have also used ontologies to describe networked sensor and actuators. But while they use OWL-S to describe the capabilities of the devices the Smartlab approach describe the context captured by the devices, emphasizing the “What, When and Where” and not the “How”.

Another distinguishing feature of Smartlab is the decoupling of the devices from the context managing service. This is achieved with the use of OSGi [8] and an event-based architecture. Events generated in the ontology are translated to OSGi events and propagated to the devices. Other frameworks have also used events to decouple services. For example agents in the Open Agent Architecture (OAA) [9] are structured around the propagation of and reaction to events. OAA also minimizes the effort required for the aggregation of new agents to the system via “facilitator agents” which use metadata to match the requests from different agents. Although this type of coordinator may exist in Smartlab their presence is not mandatory, each service knows how to react to the context changes represented in the events

[1]Tecnológico Fundación Deusto, [2]Universidad de Deusto, Spain. E-mail: [1]{aalmeida, uaguiler, ilarizgo, xlaiseca, orduna}@tecnologico.deusto.es, [2]{dipina, barbier}@eside.deusto.es

3 THE SMARTLAB ARCHITECTURE

The system has a multilayer architecture composed by four layers: Sensing and Actuation Layer, Service Abstraction Layer, Semantic Context and Service Management Layer and Programming, Management and Interaction Layer.

1. **Sensing and Actuation Layer:** This layer consists on the different devices that exist in the system environment. Representatives of these devices exist on Layer 3. Currently a wide range of devices have been implemented: EIB devices, IP Cameras, VoIP devices, SMS gateways, sensor netwoks, Indoor Location Systems and some custom-make devices like sensorized chairs and wristbands with displays and wireless capabilities.
2. **Service Abstraction Layer:** This layer encapsulates the devices functionality into OSGi bundles. Devices are discovered dynamically by the third layer and their information added to the ontology.
3. **Semantic Context and Service Management Layer:** In this layer resides all the context management and reasoning functionality of the system. It is divided in three modules:
    - *Continuous and Dynamic Environment Service Management Module*: This module constantly monitors the environment searching for new devices and manages addition of these new devices to the system.
    - *Semantic Context Management Module*: This module manages the context information stored in the ontology and provides an inference mechanism over it. When a new device is discovered all its metadata is stored in the ontology and new concepts are added if they did not exists previously.
    - *Web Gateway Module:* Offers a Web gateway that allows controlling the devices of the system.
4. **Management and Interaction Layer:** In this final layer reside the applications that take advantage of the registered services and the ontology. One example application is the Environment Controller, which uses the Web Gateway Module to offer a Web Page where all devices in the system can be controlled.

3.1 The Dynamic Ontology Enrichment

Every device bundle in Smartlab must implement the `ISmartlabService` interface. This interface exposes the methods that allow the ContextManager to query the bundle about its metadata, the new ontology concepts and the new domain rules related to that device. The interface methods are the following:

```
public interface ISmartlabService {
        public String getOntology();
        public String getIndividual();
        public String getRules();
        public String[]
        getEventsToRegister();
        public void startUpdatingContext();
        public void stopUpdatingContext();
        public String getInterface();
}
```

- *getOntology*: returns the new ontology concepts in RDF/OWL to be added. When new concepts are added to the ontology the consistency of the resulting ontology is checked. If the new ontology is inconsistent the new additions are discarded.
- *getIndividual*: returns the service instance with the device metadata to be added to the ontology in RDF/OWL. As with concepts the consistency of the new individuals is also checked before adding them to the ontology.
- *getRules*: services can also add new domain rules to the system.
- *getEventsToRegistes*: returns an String array with all the events the service is interested in.
- *startUpdatingContext*: Once all the concepts, individuals and rules have been added to the system the service can start updating the ontology with its measures. Measure consistency is also checked when it is inserted in the ontology.
- *stopUpdatingContext*: This method is called to stop the updating of the context.
- *getInterface*: returns the bundle interface name.

Currently Smartlab supports two types of devices: physical devices and logical devices. The concepts and rules provided by the physical devices (lights, windows, doors…) are not domain specific. These devices are not required to know what other devices and rules exist in the knowledge base. Domain specific concept and rules are supplied by logical devices (for example the MeetingManager described in Section 5.2) that are familiar with the other existing devices and can model the domain behavior.

4 THE SMARTLAB CONTEXT ONTOLOGY

The context in Smartlab is stored in an ontology using RDF/OWL. There are several advantages in this approach:

- The knowledge is modelled using standard languages (XML/RDF[10]/OWL[11]) and knowledge can be easily shared between applications.
- Knowledge can be easily extended adding new entities. Smartlab take advantage of this feature to dynamically enrich the ontology with new concepts.
- The semantic relations of the ontology allow inferring new context making explicit the implicit context.
- Several ontologies can be united providing they share some common entities.

The ontology is based in the systems previously discussed in the Related Work section. There are three key elements in those ontologies: who, when and where. For this reason the main entities in the ontology are the space, time and actors (devices and people):

- *TimeItem*: This class models the time and chronological relations in the ontology. Time can be expressed in two ways, as an instant or as a time period composed by two instants. Chronological

relations express how different TimeItems relate to each other: before, during, after…

- *SpatialItem*: This class models the location and spatial relations in the ontology. SpatialItems can be points, two dimensional areas, rooms or buildings. Spatial relations express how SpatialItems and LocableItems relate to each other: subsumedBy, locatedIn, contains, limits, isAdjacent …
- *LocableItem*: This class models the actors in the ontology. All the actors in the Smartlab Context Ontology have a spatial location. LocableItem have three subclases: DeviceItems which models the devices, PersonItem which models the people and ValueItem which models the measures taken by the devices. Each measure is taken in a given time and has a location associated.
- *Event*: An event is a reaction to a change in the context. These events are created by the domain rules of the system and are translated to OSGi events. The device bundles can subscribe to these events using OSGi Event Admin [12]

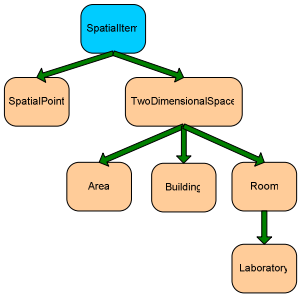

Figure 1. SpatialItem hierarchy.

5 SEMANTIC CONTEXT MANAGER

This module located in Layer 3 of the system architecture is the element in charge of managing the knowledge of the system. It can be subdivided in three other sub-modules (see Figure 2)

- *Context Manager*: The measures taken by the devices do not have per se any semantic meaning. This sub-module "semantizes" this data giving meaning to the measures and inserting them into the ontology. An example of this would be the location data given by the Indoor Location System. This system provides the X and Y coordinates of a LocableItem. The Context Manager interprets this data adding semantic information, identifying the object to which those values belong and their reference point:

```
<BinnaryLight rdf:ID="LightArea2">
<placedIn rdf:resource="#LockerRoom"/>
<name rdf:datatype="XMLSchema#string">
LightArea2
</name>
<x>23.3</x> <y>45.5</y>
</BinnaryLight>
```

- *Context Reasoner*: This module infers new context information based on the measures provided by the devices. This is done making explicit the implicit context and using domain specific rules.
- *Event Reasoner*: This module periodically checks via SPARQL [13] if a new event has been generated in the ontology responding to a change in the context. These Event entities are translated to non-semantic OSGi events and propagated using the OSGi Event Manager.

Typical context reasoning in Smartlab follows these stepts:

1. A device provides new measures that are analyzed to check their consistency and added to the ontology by the Context Manager.
2. This knowledge is processed by the Context Reasoner and new knowledge is generated by the Knowledge Eliciting rules (see Section 5.1).
3. All the knowledge is processed using the Domain Rules (see Section 5.2) and events are created in the ontology.
4. The Event Reasoner discovers those events in the ontology and extracts them from the ontology. These events are translated to OSGi Events and propagated by the OSGi Event Manager.
5. The devices subscribed to those events receive them and act accordingly.

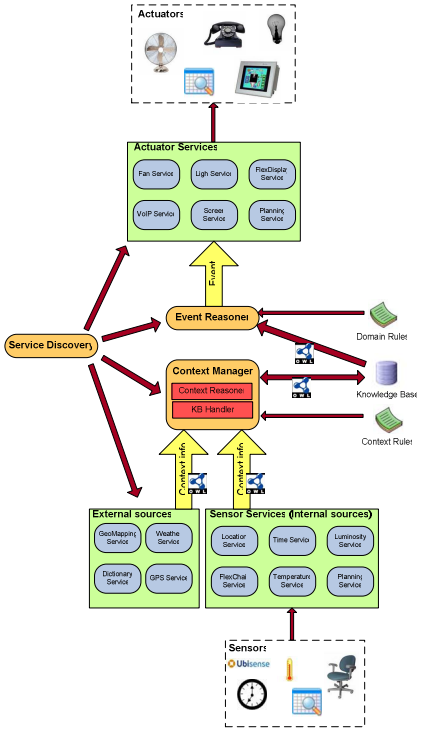

Figure 2. Semantic Context Manager Architecture

Two types of rules can be found in the system, the knowledge eliciting rules and the domain rules (see Image 3).

5.1 Knowledge eliciting rules

In this category are placed the rules that expand the ontology making explicit the hidden implicit knowledge in the ontology. There are two subcategories of knowledge eliciting rules: the semantic rules and the heuristic rules.

The semantic rules implement partially the RDF Model Theory [14] and the OWL Model Theory [15]:

```
OWL-InverseOf:
(?x ?prop1 ?y),
(?prop1 owl:inverseOf ?prop2)
->
(?y ?prop2 ?x)
]
```

The heuristic rules infer new spatial and temporal knowledge. One example would be to infer the relation between two time instants, reasoning which one take place before:

```
[TEMPORAL1-eventInstantBefore:
(?ev1 rdf:type smart:Event),
(?ev1 smart:time ?ins1),
(?ins1 smart:value ?v1),
(?ev2 rdf:type smart:Event),
(?ev2 smart:time ?ins2),
(?ins2 smart:value ?v2),
lessThan(?v1,?v2)
->
(?ev1 smart:before ?ev2)
]
```

An example of spatial reasoning would be to infer if a person is located in an area:

```
[SPATIAL10-PersonItemIsLocatedIn:
(?thg1 rdf:type smart:PersonItem),
(?thg1 smart:isLocatedIn ?loc1),
(?loc1 rdf:type smart:SpatialPoint),
(?loc1 smart:isContainedBy ?area1)
->
(?thg1 smart:isLocatedIn ?area1)
]
```

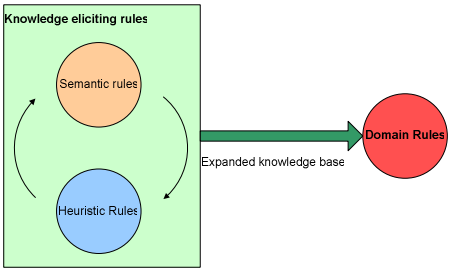

Figure 3. System rules

5.2 Domain rules

Domain rules define the behavior of the system, modeling the reactions to the changes of the context. These rules are domain specific and change from application to application. We have defined a prototype for an intelligent laboratory that is currently deployed. First the system is able to detect a meeting if three or more people are at the same time on a meeting area:

```
[EVENT-Meeting:
(?meetingArea rdf:type smart:Area),
(?meetingArea smart:containsPersonItem
?p1),
(?meetingArea smart:containsPersonItem
?p2),
(?meetingArea smart:containsPersonItem
?p3),
(?meetingArea smart:name ?meetingAreaName),
(?p1 smart:name ?name1),
(?p2 smart:name ?name2),
(?p3 smart:name ?name3),
notEqual(?name1, ?name2),
notEqual(?name1, ?name3),
notEqual(?name2, ?name3),
makeTemp(?meetingEvent)
->
(?meetingEvent rdf:type
smart:LocableMeetingEvent),
(?meetingEvent smart:meetingAreaName
?meetingAreaName)
]
```

When a meeting is detected the MeetingBundle receives the MeetingEvent and automatically displays in the projector the presentation that is planned for that meeting. Responding to the MeetingEvent more domain rules are fired. The lights of the meeting area are turned on:

```
[EVENT-SwitchOnLightsForMeeting:
(?meetingEvent rdf:type
smart:LocableMeetingEvent),
(?meetingEvent smart:meetingAreaName
?nameArea)
(?light rdf:type smart:DimmableLight>),
(?light smart:placedIn ?area),
(?area smart:name ?nameArea),
(?light smart:name ?lightName),
makeTemp(?lightOnEvent)
->
(?lightOnEvent rdf:type
smart:LightSwitchOnEvent),
(?lightOnEvent smart:lightName ?lightName),
(?lightOnEvent smart:placeName ?nameArea),
]
```

A SMS is send to the people not present in the area alerting them about the meeting:

```
[EVENT-SendMeetingSMS:
(?meetingEvent rdf:type
smart:LocableMeetingEvent),
(?meetingEvent smart:meetingAreaName
?nameArea),
(?p1 smart:isLocatedIn ?area2),
(?area2 smart:name ?name2),
notEqual(?nameArea, ?name2),
(?p1 smart:mobile ?mobile),
makeTemp(?smsEvent)
->
(?smsEvent rdf:type smart:SMSEvent),
(?smsEvent smart:areaName ?nameArea),
(?smsEvent smart:mobile ?mobile),
(?smsEvent smart:smsText 'You are late'),
]
```

And people in the meeting area receive a message in their SmartDisplay (a bracelet with an integrated display and wireless capabilities) telling them to turn of their mobile phone:

```
[EVENT-SendMeetingAlert:
(?meetingEvent rdf:type
smart:LocableMeetingEvent),
(?meetingEvent smart:meetingAreaName
?nameArea),
(?p1 smart:isLocatedIn ?area2),
(?area2 smart:name ?nameArea),
(?display rdf:type smart:smartDisplay),
(?display smart:owner ?p1),
(?display smart:name ?nameDisplay),
makeTemp(?displayEvent)
->
(?displayEvent rdf:type
smart:DisplayMessageEvent),
(?displayEvent smart:messageCode
'SwitchOffTheMobile'),
(?displayEvent smart:targetDevice
?nameDisplay)
]
```

6 CONCLUSIONS AND FUTURE WORK

This work has presented various mechanisms to manage the context and use it to infer reactions. The environments change over time (people and devices move from one place to another, new devices appear, existing devices disappear…) that's why any attempt to model a complex environment must tackle this problem. A solution would be to dynamically enrich the ontology that models the environment and the system rules to adapt on the fly to all this changes.

Further work will try to identify patterns in user's behavior and the changes in the context that take place when that patterns occur to create rules dynamically and to predict future behavior. Another improvement will be to semantize the capabilities of the services to enable the use of automatic service composition to dynamically create complex responses to context changes.

ACKNOWLEDGEMENTS

Thanks to the Industry, Commerce and Tourism Department of Basque Government for sponsoring this work through grant S-PE06FD02 of the SAIOTEK 2006 program.

REFERENCES

[1] H. Chen. An Intelligent Broker Architecture for Pervasive Context-Aware Systems. PhD thesis, University of Maryland, Baltimore County, 2004.

[2] Tao Gu, Hung Keng Pung, Da Qing Zhang. Toward an OSGi-Based Infrastructure for Context-Aware Applications. Pervasive Computing, 2004.

[3] The CoDAMoS Project: Context-Driven Adaptation of Mobile Services. http://www.cs.kuleuven.ac.be/distrinet/projects/CoDAMoS/

[4] Harry Chen, Filip Perich, Tim Finin, and Anupam Joshi. SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications, In Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (Mobiquitous 2004), Boston, MA, August 22-26, 2004.

[5] Wang XH, Zhang DQ, Gu T, Pung HK. Ontology Based Context Modeling and Reasoning using OWL. Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004.

[6] Davy Preuveneers, Jan Van den Bergh, Dennis Wagelaar, Andy Georges, Peter Rigole, Tim Clerckx, Yolande Berbers, Karin Coninx, Viviane Jonckers, Koen De Bosschere. Towards an Extensible Context Ontology for Ambient Intelligence. Proceedings of Ambient Intelligence: Second European Symposium, EUSAI 2004.

[7] Bernard Horan. The Use of Capability Descriptions in a Wireless Transducer Network. White Paper, Sun Microsystems. 2005.

[8] OSGi Alliance, OSGi Alliance Home Site; http://www.osgi.org/Main/HomePage, April. 2008.

[9] Martin, David L. and Cheyer, Adam J. and Moran, Douglas B. *The Open Agent Architecture: A Framework for Building Distributed Software Systems*. Applied Artificial Intelligence, vol. 13, no. 1-2, pp. 91-128, January-March 1999.

[10] RDF, Resource Description Framework. ONLINE. http://www.w3.org/RDF/, April 2008

[11] OWL Web Ontology Language Overview. ONLINE. http://www.w3.org/TR/owl-features/, April 2008

[12] OSGi Event Admin. ONLINE. http://www2.osgi.org/javadoc/r4/org/osgi/service/event/EventAdmin.html, April 2008.

[13] SPARQL Query Language for RDF. ONLINE. http://www.w3.org/TR/rdf-sparql-query/, April 2008.

[14] RDF Model Theory. ONLINE. http://www.w3.org/TR/2001/WD-rdf-mt- 0010925/, April 2008.

[15] OWL Model Theory. ONLINE. http://www.w3.org/TR/2002/WD-owl-semantics-20021108/, April 2008.