DYNAMIC DISCOVERY AND SEMANTIC REASONING FOR NEXT GENERATION INTELLIGENT ENVIRONMENTS

D. López-de-Ipiña*, A. Almeida$^+$, U. Aguilera$^+$, I. Larizgoitia$^+$, X. Laiseca$^+$, P. Orduña$^+$, A. Barbier*, J.I. Vazquez*

*University of Deusto, {dipina, barbier, ivazquez}@eside.deusto.es, $^+$Tecnológico Fundación Deusto, {aalmeida, uaguiler, ilarizgo, xlaiseca, porduna}@tecnologico.deusto.es – Avda. Universidades 24, Bilbao, SPAIN

Keywords: AmI, OSGi, Service Discovery, Semantic Web, Context Modelling and Reasoning.

Abstract

This work describes an OSGi-based middleware platform to enable more scalable, future-proof, cost-efficient and standard-following intelligent environments. It complements OSGi with two main features which make it even more suitable for intelligent environment management: a) dynamic discovery and monitoring of distributed semantic services and b) semantic context modelling and reasoning for intelligent service provision. Furthermore, it evaluates the suitability of applying semantic technologies in the construction of intelligent environment middleware.

1 Introduction

In Ambient Intelligence (AmI) [17], context monitoring or perception of user's current situation is paramount in order to adapt the environment's behaviour to user needs. However, it is necessary to previously augment with computational, communicational or sensorial capabilities our working and living environments if environment dynamic adaptability wants to be achieved. In other words, AmI forces us to instrument locations, devices and everyday objects so that the surrounding environment appears to be intelligent.

Environment intelligence, i.e. the environment's ability to match the user's current context and needs with the services offered by surrounding devices, may be delegated either to the mobile devices which the users wear or to the nearly imperceptible infrastructure deployed in such environments. In previous work [11] we explored the first scenario where the user carries intelligent devices which assist them in their daily activities intermediating with the environment. On the contrary, this work explores how to make our living and working environments more intelligent by adopting standard-based middleware addressing the flexibility, future-proof, programmability and intelligence requirements demanded by next generation intelligent environments.

The "SmartLab[1]: Cooperative and Programmable Intelligent Working Environment" project aims to provide a semantic middleware platform to improve the instrumentation, management and control of current intelligent environments. In this case, the environment considered is our heavily instrumented research lab SMARTLAB (http://www.smartlab.deusto.es), enriched by a wide assortment of interaction, communication and computation devices in order to improve the daily working processes carried out by researchers in it.

This work differs from previous intelligent middleware platform proposals in its focus on being reliable, future-proof and self-configurable in its intelligence capabilities, giving place to next generation intelligent environments, i.e. programmable, scalable, cost-efficient, evolvable and based on standards. In a nutshell, SmartLab provides the software and hardware infrastructure required to add evolvable intelligence to a living, working or leisure environment in a simple, extensible and scalable manner.

The following sections describe the SmartLab platform which integrates sensing, reasoning and actuation over heterogeneous equipment in an intelligent environment, independently of its concrete application domain. Section 2 offers an overview of previous related work on intelligent and semantic middleware for environment automation. Section 3 reviews the multi-layer architecture of the SmartLab middleware platform. Section 4 focuses on the dynamic discovery module of SmartLab. Section 5 describes the ontological and rule-based reasoning features of SmartLab and analyses its performance. Finally, section 6 draws some conclusions and suggests further work.

2 Related Work

There have been quite a few attempts [2][3][6][7] to create middleware which aims to simplify intelligent environment deployment, configuration and management. Among them, quite a few have considered using OSGi [12], since it is a software infrastructure which ensures platform and producer independence and easy

[1] From now on, the notation SmartLab refers to the research project, whilst SMARTLAB refers to the research lab.

programmability. It provides an execution environment and service interfaces to allow the discovery and dynamic cooperation of heterogeneous devices and services to guarantee evolution and external connectivity, allowing us remote control, diagnosis and management. This feature set explains why OSGi is emerging as a *de facto* industry standard [9] for gateways that monitor, control and coordinate the heterogeneous devices (sensor and actuators) dynamically deployed in any kind of environment (house, factory, car or even a research lab as in the case of SMARTLAB).

However, OSGi does have its limitations and needs to adapt to the context gathering, reasoning and timely actuation requirements of ubiquitous computing environments. For instance, OSGi proposes a centralised architecture which needs cooperation with network discovery protocols [5] in order to be aware of remotely available services. Although OSGi does provide mechanisms to dynamically install, uninstall or update the executing services within an OSGi framework, it is not capable by itself of discovering newly emerging devices with their associated services. Thus, research efforts such as R-OSGi [16] have given answer to this issue, by enabling seamless cooperation among services deployed in different OSGi servers. The suggested SLP-based discovery mechanism is in our opinion more resource demanding than the approach presented in section 4. Furthermore, although they provide mechanisms to dynamically discover and consume services they do not provide mechanisms to enable the automatic cooperation of several OSGi services without previous knowledge among them. SmartLab caters with this situation complementing OSGi bundles with semantic features.

Other researchers have noticed the importance of adding reasoning capabilities to OSGi servers in order to make them more suitable for AmI environment management. A current trend of this regard is to adopt OWL-based [13] semantic ontologies [1][18] as knowledge repositories upon which reasoning [10] takes place. Thus, Diaz-Redondo et al. [3] point out the problems associated to OSGi and other distributed systems middleware to enable the spontaneous collaboration of software services without previous knowledge among them. It is hardly realistic to suppose that a specific provider knows, a priori, the interfaces that match other providers' services. Their solution approach is to describe OSGi services with their properties and capabilities through OWL-S[14] so that other software elements in a residential gateway can automatically determine their purpose and know how to invoke them. SmartLab has restricted to model semantically the inputs and outputs of device representing OSGi bundles. Although service semantization is a powerful mechanism it imposes heavy processing demands not suitable for the embedded devices which usually populate AmI environments such as SMARTLAB. Furthermore, it still requires the application programmer to be aware of the types of services available in an environment in order to issue appropriate service composition requests.

Gu et al. [7] decouple OSGi services from the context they produce and consume. They provide an architecture named SOCAM which enables the easy construction and cooperation of context-aware applications or bundles through context sharing. Inference over the context published in semantic format (RDF/OWL) is carried out using both ontological and user provided rule-based reasoning. The main limitation of their approach is that they do not leverage on OSGi's already existing service registry and even notification mechanisms to register with context sources and notify events. A key feature that distinguishes SmartLab from SOCAM and other works is the ability of augmenting, on the fly, the domain ontology on which its inference process is performed. Furthermore, we adopt a more loosely coupled event-based component programmability model based on OSGi standard services.

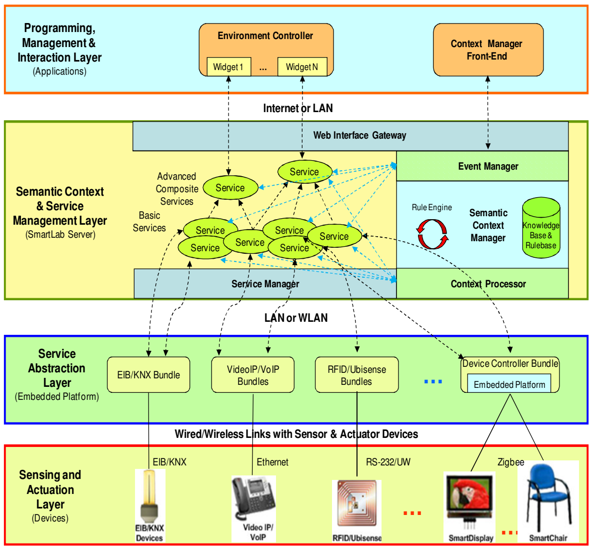

Fig. 1. SmartLab multi-layer architecture.

3 SmartLab Semantic Middleware

The main contribution of this work has been to define a multi-layer architecture, as shown in Fig. 1, which enables to configure and deploy intelligent environments more easily. The following sections describe the four layers in which such software architecture is divided and the embedded platform designed to augment everyday objects with a degree of intelligence and allow its integration with our software infrastructure.

3.1 Layer 1: Sensing and Actuation

This layer is composed by the set of devices (sensor and actuators) which populate an environment and constitute the hardware of the services offered to the user. SmartLab combines the capabilities of all those devices in layer 1 to offer advanced intelligent services to the end user in layer 3. Both industry standard and custom-built sensing and actuation devices have been used:

- *EIB/KNX*. SMARTLAB has been equipped with an EIB/KNX automation bus to which lighting, air conditioning/heating, presence detection and door and window opening sensors and actuators were attached. Thus, this bus allows for lighting, temperature or security automation.

- *VoIP and VideoIP*. Several Asterisk[4]-compatible VoIP phones have been installed in SMARTLAB so that telephony related actuations can be triggered. Likewise, several IP video cameras have been scattered throughout the lab, so that surveillance and security related services can be performed.
- *Indoor Location Systems*. The Ubisense (http://www.ubisense.net/) indoor location system has been deployed in SMARLAB. Such location system allows tracking users and objects within 20 cm accuracy. Furthermore, this location system has been complemented with several Mifare® 13.56 MHz RFID readers for detecting the presence of people not wearing Ubisense tags.
- *SmartDisplay*. This custom-built intelligent device produced combines an organic screen (μOLED-96-G1 of 4D Systems) with a WSN mote based on Mica2DOT operating on band 433 MHz capable of displaying messages broadcasted by other nearby sensing motes (see bottom left hand side of Fig. 2).
- *SmartChair*. This custom-built intelligent chair is the result of combining two pressure sensors at the base and back of a conventional chair with a Mica2 mote which broadcasts the pressure values (see top left hand side of Fig. 2).
- *SmartContainer*. This custom-built container resulted from combining a liquid level sensor with a Mica2 mote alerts every time the water goes over some limits (see bottom right hand side of Fig. 2).

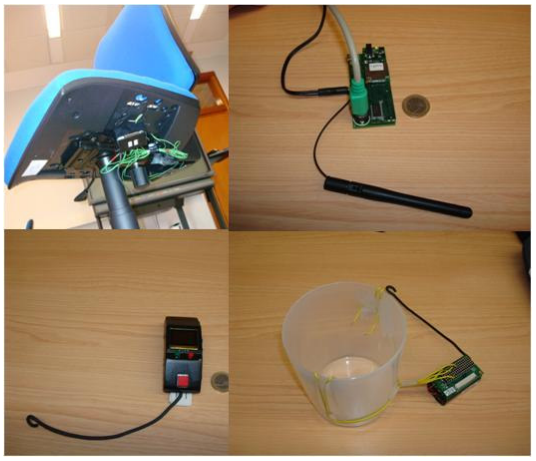
Fig. 2. SmartChair (top left), SmartLab Embedded Platform (top right), SmartDisplay (bottom left) and SmartContainer (bottom right).

3.2 Layer 2: Service Abstraction

This layer transforms the functionality of the devices of layer 1 into software services. An OSGi bundle implementing such services has been provided for each of the device types deployed within SMARTLAB. Thus, in our model every device type is encapsulated in the form of a bundle or execution unit within the OSGi environment. Such bundles are .jar archives which include several binary files and a manifest file with certain configuration properties for each bundle. In fact, such bundles rather than encapsulating the real functionality, they act as proxies or delegates of such functionality and are downloadable and installable into OSGi environments where they process commands issued by other bundles or third party applications which are propagated to the devices in layer 1.

A key aspect to consider is where to host the bundles which act as proxies of physical devices. The obvious location is the devices themselves. However, this requires that devices have to have the capability of publishing and announcing bundles, and, more importantly, serve the requests received from such bundles once proxies have been installed in a client OSGi environment. Given that is not common to encounter physical devices with such advanced computation capabilities, usually the software representatives of such devices will be executed in servers which mediate between the real devices through home automation buses, serial ports or any other communication mechanism and the clients of such bundles. Next, the OSGi bundles corresponding to the sensing and actuation devices of section 3.1 are described:

- *EIB/KNX Bundle*. It is located in a PC connected to the EIB/KNX bus through a BCU (Bus Coupling Unit) USB interface and executing the KNXnet to IP tunneling server Tweety (https://www.auto.tuwien.ac.at/a-lab/eibdtweety.html). The main mission of this bundle is to allow message passing from and to the home automation bus. It exports a range of services representing each of the EIB devices plugged into the bus. Thus, if the environment has binary lights it will register an OSGi service with the interface `IBinaryLight` for each of the existing lights in an environment. Other types of services used are `DimmableLight`, `Alarm`, `OpeningSensor` or `Heating`. The implementation of such bundles has made use of the Calimero Java library which allows the communication with the Tweety server earlier mentioned. This and the following bundles are exported through the dynamic service discovery and installation protocol described in Section 4.
- *VideoIP Bundle*. This bundle registers a service for each of the four D-Link DCS-5300G PTZ IP cameras deployed trough SMARTLAB. The service exported offers methods to capture image stills, move the camera objective or activate the camera built-in movement detection mechanism, which uploads the captured images into an FTP server every time that movement is detected.
- *VoiceIP Bundle*. This bundle is located in a PC equipped with an Ambient MD3200 card which allows the communication between the Asterisk [4] software and the telephone system. In fact, this bundle communicates with a web service in the Asterisk machine offering functionality such as: a) user management, b) call management for registered

users, c) text synthesis from text, d) automatic pre-recorded message alert calls or e) call waiting message recording.

- *Ubisense Bundle*. This bundle is hosted in the machine where the Ubisense location server is installed. When exported it acts as proxy of a web service wrapper for the Ubisense DCOM component. It offers the following functionality to third applications: a) list people present in a given location, b) list objects whose location has changed from last reading, c) notification of appearance, disappearance and movement of Ubisense tagged-objects.
- *Device Controller Bundle*. It is responsible of deploying each SmartLab custom-built device (remember Fig. 2) as an OSGi service. Such bundles are published by the SmartLab Mote Communication server executed within the embedded platform selected to mediate with our custom devices.

3.3 SmartLab Embedded Platform

SmartLab vision is that small footprint low-cost almost invisible embedded platforms attached to everyday objects will foster intelligent environment deployment. These devices will facilitate the integration of computation and communication capabilities into everyday objects, which can then be exported, downloadable and easily consumable from third party applications. SmartLab has addressed this idea by assembling an embedded platform constituted by the following elements:

- A Gumstix Connex 400xm with a 400 MHz processor, 16 MB Flash memory and 64 MB RAM.
- A Wifistix module which provides Wi-Fi communication capabilities to the Gumstix module so that it can connect with a IP to wireless sensor network (WSN) bridge based on the Crossbow Mica 2 platform, executing in one of the servers of SMARTLAB.
- A set of Mica2 sensor nodes integrated into the set of proprietary sensing (SmartChair and SmartContainer) and actuation (SmartDisplay) devices reviewed.

The Gumstix module acts as gateway between a Wi-Fi network and a Crossbow Mica2-based WSN constituted by the objects to which sensor motes have been stuck. These motes are capable of communicating at 38.4 Kbaudios imposing minimum energy consumption requirements and exposing a reduced size, which makes them ideal for integration with everyday objects. The software image installed in them uses JamVM, compatible with Java 2, but optimized for devices with reduced computing resources.

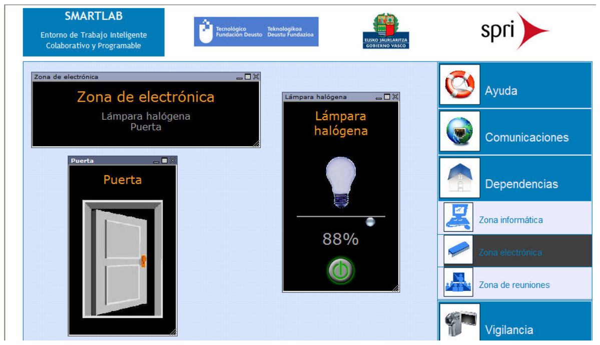

Fig. 3. Environment control of the electronics area within SMARTLAB: lighting and door opening/closing.

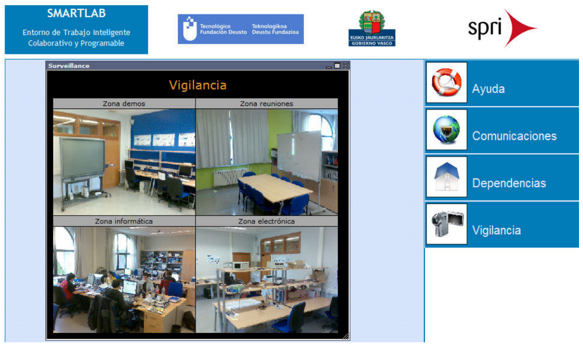

Fig. 4. Surveillance control in SMARTLAB.

3.4 Layer 3: Semantic Context Modelling and Service Management

The Knopflerfish-based OSGi server which implements layer 3, namely SmartLab Server, acts as a gateway between the advanced OSGi services resulted from combining, often through the OSGi Wire Admin Service, the functionality offered by the equipment and smart objects deployed in SMARTLAB and the set of third party agents which want to exploit such services. Such gateway is composed of the following three core modules (review Fig. 1):

- *Continuous and Dynamic Environment Service Management Module*. Implemented by the Service Manager component, it monitors the environment continuously to determine the appearance and disappearance of new services provided through layer 2 as result of the activation and deactivation of equipment within an intelligent environment.
- *Semantic Context Management Module*. Implemented through the Semantic Context Manager component, it provides a semantic inference component which operates over an ontology where all context knowledge is gathered. Whenever a new device bundle is discovered all its semantic metadata is gathered and added to the ontology constituting the knowledge base and the domain rules set kept in this module. It can happen that the rules published by a device depend on knowledge provided by another device representing service. It is not required that a device has a priori knowledge of another device. However, occasionally it will be needed in case that knowledge gathered by two or more different device

types wants to be combined and reason upon. This is the only point when certain coupling among devices can take place within SmartLab.

- *Web Gateway Module*. Implemented through the Web Interface Gateway component, it offers a web based interface for third party application wanting to interact with the environment services managed by SmartLab Server. It behaves both as a web-to-OSGi service gateway and as an HTTP web controller for the requests issued by the web gadgets that constitute the front-end to the advanced services provided by layer 3. This element translates HTTP requests generated from interactions over the gadgets controls into method invocations of OSGi services.

3.5 Layer 4: Service Programmability, Management and Interaction

It is the top layer of the architecture where the final applications which make use of the SmartLab infrastructure lie. Furthermore, it provides two generic applications which allow controlling and managing an intelligent environment, namely Environment Controller and Context-Manager Front-End. Next, these two generic applications are described:

- *Environment Controller*. This component offers an intuitive interface based on the web gadget concept to control the software services associated to instrumented objects in the environment or the advanced services which coordinate the operation of several of those elements in SMARTLAB. This environment administrator makes use of small size web gadgets in order to be accessible from different clients such as web browsers, tactile screens or mobile devices.

  The web interface developed requires user login. Once a user has logged in, they can interact with a right hand side menu, see Fig. 3 and Fig. 4, which brings on the left hand side a set of gadgets representing the services available in the environment. This gadget-based system developed over the JavaScript library X (http://www.cross-browser.com/) is completely modular, easing the creation and incorporation of new user interfaces for the management and control of new devices connected to the system.

  Any browser both in a PC or mobile device respecting web standards will be suitable to render this interface. Developing new gadgets for this application is also simple. It only requires knowledge about common web technologies such as HTML, CSS and JavaScript.
- *Context-Manager Front-End*. This component offers a web interface which allows the following management actions:
  a. Management of device-associated OSGi bundles (activation, deactivation, elimination) and aggregated services currently active within SmartLab Server.
  b. Modification of the ontology which constitutes the knowledge base for SmartLab's reasoning mechanism.
  c. Management of environments' rule behaviour.
  d. Tracking of system log and statistics to verify SmartLab server's continuous correct behaviour.

4 Dynamic Discovery of Semantic Bundles

This section explains how conventional OSGi bundles enhanced with semantic information are dynamically discovered, analysed, downloaded and installed into SmartLab Server. That way, this server is always aware of the most up-to-date range of active environment services.

4.1 Semantically-enhanced OSGi Bundles

Something remarkable about SmartLab is that the bundles discovered augment semantically the context-modelling ontology regulating SmartLab Server behaviour with new concepts (context classes and events) and instances of such concepts. The operations fulfilled by every SmartLab OSGi bundle on installation in the server are:

1. The ontology modelling context knowledge within SmartLab Server is enriched with new device, context and event representing classes.
2. New behaviour semantic rules are fed into the inference engine built in within SmartLab Server. Such engine reasons over the semantic data provided by the different bundles installed in the server.
3. A set of aggregated event types [10] is specified which the rules defined by a device bundle may generate and to which other bundles can subscribe.

On the other hand, during its lifetime, the services exported by a bundle are continuously publishing instances of the ontology classes earlier defined, which as a result fire the rule engine and generate aggregated event instances.

Every device representing bundle in SmartLab must implement the `ISmartlabService` interface for each service it exports. This interface defines the contract between the ContextManager component of layer 3 and the discovered bundle services. The structure of such interface is 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();
}
```

The meaning of the methods provided by such interface which are invoked by the Context Manager is:

- `getOntology` – returns an ontology with the device classes, their values and events in RDF/XML format.

- `getIndividual` – returns an instance of the device class with current values for such device in RDF/XML.
- `getRules` – returns behaviour rules exported by the device in Jena [8] rule format.
- `getEventsToRegister` – returns the names of types of events in which the bundle is interested.
- `startUpdatingContext` – the ContextManager calls this method to request a bundle to start updating its measures. Such context update is undertaken by the semantic bundle by invoking the `UpdateContext` method of ContextManager.
- `stopUpdatingContext` – the ContextManager calls this method whenever a bundle should stop updating context.
- `getInterface` – returns the name of the bundle interface.

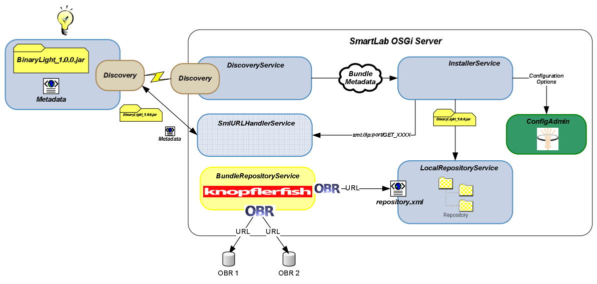

Fig. 5. Service Manager Internal Architecture.

4.2 Dynamic Service Management Module

This module allows the continuous and dynamic publication, announcement, discovery, download, installation and configuration of distributed services without requiring server reboot. The following OSGi bundles cooperate in such process and constitute the internal architecture of this module shown in Fig. 5:

- *Discovery Service Bundle*. It implements the discovery protocol used by both SmartLab Server and the devices willing to export semantics enhanced bundles. A simple multicast protocol (see Fig. 6) is used to announce, discover, obtain bundle and service metadata and download the corresponding bundle.
- *Installer Service Bundle*. It decides whether a candidate bundle should be installed or not. For that, it uses the meta-information provided by the discovery mechanism. Given that an OSGi bundle can define a set of dependencies for their correct behaviour, this bundle is responsible of ensuring such dependencies are fulfilled. Such process is undertaken with the help of an OSGi Bundle Repository (OBR), which is a repository where the bundle jars are placed together with an XML description of each of them and their dependencies.
- *SmlURLHandlerService Bundle*. Whenever the installation service discovers a candidate bundle, it must process the bundle metadata to decide whether to proceed or not. The discovery and installation processes are dependent on the protocol supported by the target device to export the bundle's .jar file. In order to make the discovery protocol agnostic to the code download mechanism used, a URL handler has been defined, named SmlURLHandlerService which is compatible with the standard OSGi UrlService. Thus, the installation service can obtain both the metadata and the .jar file through an URL (starting by `sml://`), without having to know the internal details about how that data is really gathered.
- *LocalRepositoryService Bundle*. It stores the bundles downloaded. This repository is updated every time a new bundle .jar is received or an earlier discovered bundle has stopped sending heartbeats (`ANNOUNCE` protocol message) generating a new version of the XML file describing its contents. The basic functionality of an OBR is already provided by Knopflerfish. This OBR has only been complemented by the Installer Service with the LocalRepositoryService bundle. The installer service decides whether to install or not a specific bundle by checking the LocalRepositoryService XML file.
- *OSGi Configuration Service*. Given that bundles are installed (an uninstalled) automatically and, in most cases, they manage a remote device or service, very often some dynamically changing configuration details (IPs, ports and so on) must be supplied to update the internal configuration of the services registered by a bundle. With that objective the parameters which may change are declared in the bundle manifest. Those parameters are loaded and managed from then on by the OSGi Configuration Admin.

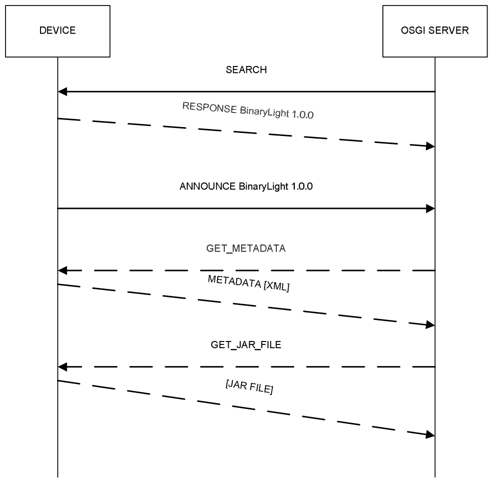

Fig. 6. Service Discovery Protocol Messages.

5 Semantic Context Management

The Semantic Context Manager component, shown in layer 3 of Fig. 1, consists of three main elements, which are further detailed in Fig. 7:

- *Context Processor*. This component gathers the context supplied in semantic format by the deployed services and populates with it SmartLab Server's context ontology knowledge base. An example of this is the data gathered from a location system.

Initially, these data are only several real numbers, but the semantization process within each SmartLab OSGi bundle interprets them and adds semantic information, indicating that X and Y refer to the position on an object and that those values are relative to a given 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*. It augments the context information making explicit hidden implicit information. For that, it uses both the semantic relations of the SmartLab ontology and the rules designed for a specific domain (e.g. SMARTLAB research lab). After new context information is inferred new reactive behaviour is produced triggering events such as: “there is a meeting”, “there is flood”, “person X entered in room Y” and so on.
- *Event Manager*. It transforms the notifications generated on the right hand side (RHS) of SMARTLAB domain rules into non-semantic OSGi EventAdmin-compatible events. Besides, with the help of OSGi EventAdmin, it maintains a registry of all events for which interest has been expressed. Internally, it checks continuously SmartLab’s RDF knowledge base through SPARQL to find out newly generated events.

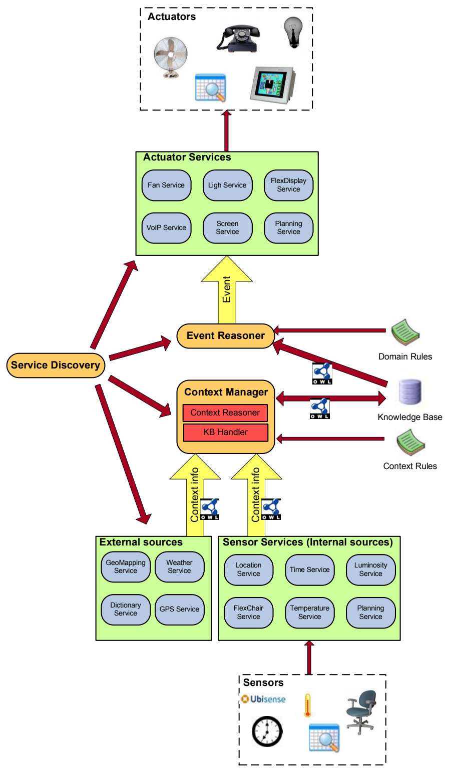

Fig. 7. Semantic Context Manager Architecture.

The following steps illustrate the context reasoning process undertaken within SmartLab Server:

1. SmartLab knowledge base is fed with context information in semantic format provided by the different services installed in SmartLab Server.
2. After updating such semantic model, the inference engine provided by Jena firstly enhances the RDF triples currently available with newly inferred ones and secondly finds domain rules which trigger in their RHS the generation of aggregated events associated to reactive behaviour to be performed by the devices deployed in SMARTLAB.
3. Aggregated event notification is produced after mapping the semantic events generated into OSGi events. Such events are broadcasted automatically by OSGi EventAdmin to the SmartLab bundles that registered interest on the type of events generated.
4. The notified bundles use the event information to generate the associated reactions. For instance, make a call through Asterisk, switch on the lights through EIB and so forth.

5.1 SmartLab Generic Context Ontology

The knowledge base in SmartLab is modelled by an ontology, which offers the following advantages:

- The knowledge base is expressed in XML/RDF/OWL, a standard set of languages which can be easily processed by applications.
- An ontology can be easily extended with new entities without affecting the already existing ones.
- Several ontologies can be combined in an easy manner providing there are entities in common. Thus, other applications can expand this knowledge base, enhancing and improving it in this manner.
- The semantic relations expressed in the ontology allow making explicit the implicit knowledge in a simple manner.

This ontology has been designed taking into account earlier semantic AmI systems [1][7]. Three key elements within the ontology have been identified which model context for any intelligent environment: “where”, “when” and “who”. Therefore, the main elements of this ontology are space, time, actors (people, agents or devices) and events:

- `TimeItem` – class which models time.
- `SpatialItem` – class which models locations.
- `LocableItem` – assigns locations to entities.
- `Event` – context changes which occur at a specified time at certain location.

5.2 Knowledge Generation in SmartLab

SmartLab uses two knowledge generation mechanisms based on reasoning over its generic context modelling ontology: a) ontological reasoning and b) rule-based reasoning. Ontological reasoning makes use of the RDF predicates: `rdf:domain`, `rdf:range`, and `rdf:type`; the RDFS predicates:

`rdfs:subPropertyOf`, `rdfs:subPropertyOf`; and the OWL ones: `owl:TransitiveProperty`, `owl:SymmetricProperty` and `owl:inverseOf`. Such predicates are used to infer implicit knowledge from the explicit knowledge gathered in the form of RDF triples in the SmartLab Server's knowledge base. On the other hand, rule-based reasoning can be used to define relations among entities in the generic ontology which relates space, time and actors and which cannot be expressed through RDF, RDFS and OWL predicates. For example, the following rule specifies when an event `?ev1` should be considered to have happened before event `?ev2`.

```
[TEMPORAL1-eventInstantBefore:
   (?ev1                              rdf:type
<http://deusto.es/smartlab.owl#Event>),
   (?ev1     <http://deusto.es/smartlab.owl#time>
?ins1),
   (?ins1   <http://deusto.es/smartlab.owl#value>
?v1),
   (?ev2                              rdf:type
<http://deusto.es/smartlab.owl#Event>),
   (?ev2     <http://deusto.es/smartlab.owl#time>
?ins2),
   (?ins2   <http://deusto.es/smartlab.owl#value>
?v2),
   lessThan(?v1,?v2)
->
   (?ev1   <http://deusto.es/smartlab.owl#before>
?ev2)
]
```

Finally, it is very important to model the knowledge and behaviour of a concrete application domain as is the case of SMARTLAB, which is equipped with a range of device types to facilitate the activities of researchers. For instance, the following rule would determine that a meeting is taking place in SMARTLAB whenever there are at least two people in the meeting area. Observe that the SmartLab OSGi bundle responsible of activating the elements to run a meeting (show presentation in projector, switch on heating, regulate lighting, and so on) would not only register the following rule with the Context Manager but it would also enrich the generic ontology with concepts such as `MeetingArea` which hierarchically depends on `SpatialItem` and more concretely in the concept of `Area` shown in Fig. 8.

```
[EVENT-Meeting:
(?meetingArea                         rdf:type
<http://deusto.es/smartlab.owl#MeetingArea>),
(?meetingArea
<http://deusto.es/smartlab.owl#containsPersonItem
> ?p1),
(?meetingArea
<http://deusto.es/smartlab.owl#containsPersonItem
> ?p2),
 (?p1        <http://deusto.es/smartlab.owl#name>
?name1),
(?p2         <http://deusto.es/smartlab.owl#name>
?name2),
notEqual(?name1, ?name2),
makeTemp(?meetingEvent)
->
(?meetingEvent
rdf:type<http://deusto.es/smartlab.owl#LocableMee
tingEvent>),
(?meetingEvent
<http://deusto.es/smartlab.owl#place>
?meetingArea)
(?meetingEvent
<http://deusto.es/smartlab.owl#name> ?name1)
(?meetingEvent
<http://deusto.es/smartlab.owl#name> ?name2)
]
```

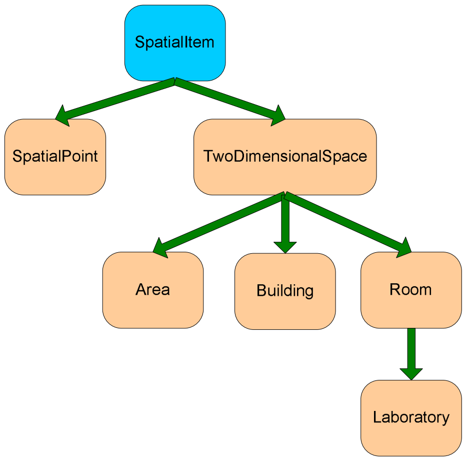

Fig. 8. SpatialItem Class Hierarchy.

5.3 Knowledge Reasoning in SmartLab

As mentioned, SmartLab undertakes two types of context reasoning: ontology- and rule-based reasoning. Therefore an important design decision was to choose whether to combine both types of reasoning in the same reasoning engine or use two different engines. The main problem of using separate engines is that knowledge must be continuously exchanged between both engines, there is no common knowledge base. On the other hand, using an embedded reasoning engine within the ontological engine would not allow the ontological engine to observe the changes undertaken by the rule-based engine over the latest ontological knowledge base snapshot. The main advantage of using a single engine is that no knowledge interchange is required. Besides, only a reduced set of OWL predicates may be regarded for ontological reasoning if the whole set is not required, so reducing the overall resource demands. However, the main restriction is that the OWL semantic predicates have to be re-expressed in the chosen rule format.

The following example shows how to express the OWL predicate `owl:inverseOf` as a semantic rule in our chosen (Jena) rule format. In order to simplify the definition of such transformations we have made use of the theoretic models of RDF, RDFS and OWL[13].

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

Jena rule engine's forward chaining inference capabilities allow the Context Manager to undertake three types of inference divided in two phases. In the first phase, the engine extracts the knowledge implicit in the ontology and processes its semantic relations to expand the knowledge base. Then, in the second phase, making use of the expanded knowledge base it infers actions to be carried out by the system, i.e., it generates the events that the actuation bundles respond to. Three types of rules have been defined:

- *Semantic rules* – they are divided in two subcategories: RDF and OWL. In order to implement such rules the theoretical models of both specifications have been analysed. Only rules that extend the knowledge base using semantic relations have been considered.
- *Knowledge extraction rules* – they enable extracting high level knowledge from the ontology implicit knowledge. The three main aspects modelled on the system are *what*, *when* and *where*. Therefore, the knowledge inferred is based on time and location. Among other things, these rules order events and find out the physical relations among them.
- *Event-inferring rules* – they generate aggregated events from the context currently stored in the knowledge base. They reason over the existing explicit or generated knowledge to reason upon it and infer what events should be launched. Although the knowledge extraction rules can be extrapolated to any AmI application and domain (factory, car, home, laboratory), the event inferring rules depend on the specific domain considered. These rules define the behaviour of a specific type of environment. Thus, to illustrate this point two types of rules have been defined related to the SMARTLAB domain: meeting rules and security/safety rules. These rules alert the bundles interested on easing meetings, or increasing the environment safety (prevent floods) by switching off equipment or closing doors. The previously reviewed `EVENT-Meeting` rule shows how to generate a `MeetingEvent` whenever there are at least two people in the meeting area.

5.4 Performance of Knowledge Inferring Process

Adopting a semantics-based reasoning mechanism is very powerful. However, currently available reasoning engines still do not offer great performance. In order to assess this point we have carried out several tests to determine the performance of ContextManager in two situations: a) the inclusion of new semantic information and b) the rule-based reasoning process. It is important to note that whenever the SmartLab ontology is upgraded with new concepts such as classes or events, a validation process is carried out that ensures the continuous validity of it. This process obviously delays the reasoning process but it is compulsory to guarantee the ontology integrity.

Concretely, the discovery of several devices which register their ontology extensions, individuals and rules has been emulated. The tests have been repeated considering 5, 10, 20, 50 and 100 different devices. The following measurements have been taken:

- *Registry time* – represents the time invested in introducing the semantic information added by a device to the default SmartLab ontology and the device's behaviour rules to the SmartLab rule base.
- *Inference time* – represents the time required to introduce the context information, e.g. temperature value change, and the rule based inference process resulted from this change is concluded.
- *Average triples, classes, individuals and rules* – gives an idea of the size of the knowledge base in every case considered.

	5 Devices	**10 Devices**	**20 Devices**	**50 Devices**	**100 Devices**
Average Registry Time (ms)	506	527	415	535	1000
Average Inference Time (ms)	1854	2620	3671	9944	27827
Average triples	866	965	1164	1656	2329
Average clases	112	123	143	203	303
Average individuals	66	84	121	194	269
Average rules	29	35	45	75	125

Table 1: Context Manager Performance Tests

Table 1 illustrates the results obtained by the Context Manager component when registering new semantic concepts and performing its reasoning process in average, taking into consideration the cases where 5, 10, 20, 50 and 100 devices are simultaneously generating new context inputs. Obviously, the more services the system has to cope with the worse performance it is obtained. The average reasoning time obtained after a context change has been produced ranges from 1.8 (acceptable) to 27.8 (non-acceptable) seconds when passing from 5 to 100 concurrent active devices.

6 Conclusion and Further Work

This work has presented some middleware extensions to the OSGi framework in order to cater for the evolution and self-configuration needs of Intelligent Environments. An environment configuration changes continuously across time and the management middleware must adapt to those changes seamlessly. The SmartLab Service Manager component proposed is capable of discovering newly emerging exporting services devices and to find out when earlier discovered services become unavailable. Furthermore, this service manager is capable of dealing with SmartLab bundles, i.e. standard OSGi bundles augmented with semantic metadata. Such metadata allows both the knowledge and rule bases of an environment, such as SMARTLAB, to be upgraded on the fly through the SmartLab Semantic Context Manager component. Consequently, the concepts modelled and the behaviour rules which regulate the reactivity of the environment are automatically and dynamically adapted to the latest configuration, i.e. devices deployed, of an environment, without user intervention.

Further work will tackle the performance issues identified in section 5.4, regarding semantic rule-based reasoning. These performance problems are partly due to the use of a non-optimized rule engine in our experiments. Adopting semantic technologies to model and reason about environment context and services is undoubtedly very flexible and powerful, but unfortunately imposes higher resource requirements than other more conventional modelling and reasoning techniques such as RDBMS or traditional rule engines (CLIPS, Jess). In future work, we will replace the Jena reasoning engine by other more sophisticated rule engines such as Pellet [15] and work on the optimization of the results obtained. In addition, another area of improvement will be to semantize not only the context provided by bundles but also the services they offer. Thus, we will enable the automatic composition of OSGi services' functionality without user supervision but attending to the specific needs of third-party users.

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] A.K. Dey. Providing Architectural Support for Building Context-Aware Applications, *PhD thesis, Georgia Institute of Technology*, 2000.

[3] R. P. Díaz Redondo, A.F. Vilas, M.R. Cabrer, J.J. Pazos Arias, J. García Duque, A. Gil Solla. Enhancing Residential Gateways: A Semantic OSGi Platform. *IEEE Intelligent Systems*, vol. 23, no. 1, January/February 2008, pp. 32-40

[4] Digium Inc. Asterisk: The Open Source PBX & Telephony Platform. *http://www.asterisk.org/*, February 2008

[5] W. K. Edwards. Discovery Systems in Ubiquitous Computing. *IEEE Pervasive Computing*, vol. 5, pp. 70-77, 2006.

[6] A. Helal, W. Mann, H. Elzabadani, J. King, Y. Kaddourah and E. Jansen. Gator Tech Smart House: A Programmable Pervasive Space. *IEEE Computer magazine*, March 2005, pp 64-74.

[7] T. Gu, H.K Pung, D.Q. Zhang. Toward an OSGi-Based Infrastructure for Context-Aware Applications. *IEEE Pervasive Computing*, vol.3, no. 4, October 2004, pp. 66 - 74, ISSN:1536-1268

[8] Jena Semantic Web Framework. *http://jena.sourceforge.net/*, February 2008.

[9] C. Lee, D. Nordstedt, and S.Helal. Enabling Smart Spaces with OSGi. *IEEE Pervasive Computing*, vol. 2, no. 3, July-Sept. 2003, pp. 89 – 94

[10] D. López-de-Ipiña et al. An ECA Rule-Matching Service for Simpler Development of Reactive Applications, *IEEE Distributed Systems Online*, vol. 2, no. 7, 2001.

[11] D. López de Ipiña, J.I. Vázquez, D. García, J. Fernández, I. García, D. Sainz and A. Almeida. A Middleware for the Deployment of Ambient Intelligent Spaces. *Ambient Intelligence in Everyday Life, Springer, LNAI 3864*, ISSN 0302-9743, pp. 239-255, 2006

[12] OSGi Alliance, OSGi Alliance Home Site; *http://www.osgi.org/Main/HomePage*, February. 2008.

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

[14] OWL-S: Semantic Markup for Web Service. *http://www.w3.org/Submission/OWL-S/*, February 2008

[15] Pellet: The Open Source OWL DL Reasoner. http://pellet.owldl.com/, February 2008

[16] J. S. Rellermeyer, G. Alonso, T. Roscoe. R-OSGi: Distributed Applications through Software Modularization. *Proceedings of the ACM/IFIP/USENIX 8th International Middleware Conference Conference (Middleware 2007)*, Newport Beach, CA, 2007

[17] N. Shadbolt, Ambient Intelligence, *IEEE Intelligent Systems*, vol. 2, no.3, 2003

[18] 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.