A MDD Strategy for developing Context-Aware Pervasive Systems

Master Thesis – Postgraduate studies 2007/2008
"Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información"

UNIVERSIDAD POLITECNICA DE VALENCIA

Author: Estefanía Serral Asensio

Supervisors: Dr. Vicente Pelechano Ferragud
Dr. Pedro J. Valderas Aranda

Contents

1. Introduction

The work presented in this master thesis deals with the problem of developing context-aware pervasive systems. The term ‘pervasive’ introduced first by Weiser[1] refers to the seamless integration of devices into the users’ everyday life. Weiser defined pervasive systems as those that “*weave themselves into the fabric of everyday life until they are indistinguishable from it.”* To make this vision a reality, systems should vanish into the background to make the user and his actions the central focus rather than computing devices and technical issues. One field in the wide range of pervasive computing is the research of context-aware systems. A context-aware system *“use context to provide relevant information and/or services to the user, where relevancy depends on the user’s task”* [2]. These systems are able to adapt to the current context, making that user intervention is sought only when it is absolutely required. Thus, these systems aim at increasing usability and effectiveness by taking context into account.

Several efforts have been developed in order to support context-aware pervasive system development through software infrastructures, frameworks and models for describing context information. However, developing such systems is still a complex task because of the lack of adequate software abstractions, programming models, methodologies and efficient frameworks. This master thesis proposes a methodological approach to develop context-aware pervasive systems based on ontologies and the Model-Driven Development (MDD) guidelines [3]. This approach makes four main contributions: 1) a set of models to capture context at modelling time, 2) a context ontology and an OWL context repository based on this ontology to capture context at runtime; 3) a framework for automatically storing, managing and processing the context information at runtime, and 4) an infrastructure for adapting the pervasive system in order to improve user life. Additionally, we integrate our approach with a MDD method to develop pervasive systems that has code generation capabilities. This integration facilitates the development of a MDD method that allows us to automatically obtain functional context-aware pervasive systems from their modelling.

The rest of this chapter is organized as follows: Section 1.1 explains the purpose of this thesis. In Section 1.2, the problem that this thesis resolves is stated in detail. Next, the main contributions of this thesis are summarized in Section 1.3. The research methodology that we have followed to develop this master thesis is presented in Section 1.4. Next, Section 1.5 explains the context in which this master thesis has been developed. Finally, Section 1.6 describes the structure of this master thesis.

1.1 Motivation

Nowadays, a number of leading technological organisations are exploring pervasive computing; however, the solutions developed up to now are far from becoming reality the Weiser's vision. A step further towards this vision is the research in context-aware systems development. *By improving the computer's access to context, we increase the richness of communication in human-computer interaction and make it possible to produce more useful computational services* [2]. Thus, Context-awareness becomes a key issue if we want to create systems that are invisible and disappear in terms of the user's perception.

Context-aware systems not only must capture context information, but also they must understand context and adapt their behaviour according to it. To achieve these requirements, *it is necessary a context model to define and store context data in a machine processable form* [4]. Different context models have been proposed until now; for instance the projects CORTEX [5] and Hydrogen [10] propose object oriented models, the Context Toolkit [2] proposed by Dey model context by using Key-Value models; ContextUML [8], CML [7] and [9] are Graphical modelling languages; etc.

Several studies [4] [12] [23] state that the use of ontologies to model context is one of the best choices. They state that this model guarantees a high degree of expressiveness, formality and semantic richness. Ontologies also exhibit prominent advantages for reasoning and reusing context as well as making the integration of pervasive environments easier. Some examples of ontology-based approaches are CoBrA[13], SOCAM [14] and COMANTO [15]. However, in spite of the numerous approaches that deal with context modelling, the development of a flexible and useable context model that covers the wide range of possible context information in pervasive systems is still a challenging task.

In addition, as we have said, context-aware systems, as well as having a suitable context model, must understand context and adapt to it. To do this, they must first have sufficient intelligence and knowledge-awareness to react appropriately according to context. To achieve this goal it is not enough to acquire context that can be directly captured via sensors, device status, user profiles, etc.; it is necessary reasoning about context at semantic level to interpret this information and obtain knowledge from it. According to this knowledge, the system must adapt its behaviour to meet the needs of users within ever-changing context. This adaptation of applications requires a clear architecture and a well founded explicit relationship between the context as a key element and the adaptation strategy in pervasive computing. Many research efforts, such us [7], [13] and [21], have been made to address these issues, but the solution is yet missing.

Finally, achieving the Weiser's vision by means of the development of a context-aware pervasive system that accomplishes the requirements explained above can be a difficult task without a suitable methodology. However, most of current approaches to develop context-aware pervasive systems use ad-hoc solutions in spite of modelling context. Thus, these approaches do not usually take into account any software engineering method. This makes that developers are more focused on solving technological problems than on satisfying systems requirements. Besides, manual developments are error-prone and the resulting product use to be buggy and hard to maintenance and evolve.

If we also take into account the permanent technological innovations and the complexity and dynamism of pervasive computing systems, their manual implementation and maintenance render impractical. Solid engineering methods are needed in order to produce robust systems in an efficient way.

In this context, a model-driven approach seems a good choice due to several reasons (they are further explained in Section 2.3). Since the current technology is evolving fast, the domain knowledge is captured in models which are independent of technology and, therefore, are not affected by platform evolution. In addition, this approach allows developers to focus on the requirements of the system and facilitate the maintenance of the systems developed by following it. Moreover, if a new technology appears, it is not necessary to describe again the system, but just a specific (and reusable) generator must be developed.

1.2 The Problem Statement

The development of Context-Aware Pervasive Systems is not a closed research topic. The above discussion indicates that some problems still need to be considered.

The work that has been done in this master thesis is an attempt to improve the development of Context-Aware Pervasive Systems by considering these problems, which can be stated by the following four research questions:

- **Research Question 1.** How should context of pervasive systems be represented at conceptual level?
- **Research Question 2.** How should context information that is only available at run time be managed by the system in order to understand and interpret it at semantic level?
- **Research Question 3.** How should the system act and react according to context to give a better support to system users?
- **Research Question 4**. How should context support mechanisms be included within the development process of pervasive systems?

1.3 Main Contributions

We present in this subsection the main contributions that have been developed to resolve the four research questions presented 1.2:

1. We propose a set of models that provides high-level abstractions to manage and handle context information of pervasive systems at conceptual level. This contribution is obtained by achieving the following goals:
   1.1. Studying the concept of Context and identifying the context information that can be represented at modelling time.
   1.2. Studying the different approaches that are proposed to specify context information.
   1.3. Identifying the main abstractions that characterize the context information and proposing a set of models to represent these abstractions.
2. We define a context ontology and an OWL context repository based in this ontology for being capable of store in a machine processable language both current context information and the historical context information. In addition, we present a Java framework in order to manage the OWL context repository and interpret this information at runtime. To do this, this framework automatically updates the OWL context repository according to the changes produced in context information at runtime. Moreover, it allows us to reason about this information at semantic level in order to interpret it and derive

knowledge from this information. The following goals are faced to carry out this contribution:

2.1. Studying the existent approaches that are used to capture context at runtime.
2.2. Studying the different mechanisms that can be used to manage and understand context at runtime.
2.3. Implement the suitable mechanisms to manage context and derive knowledge from it at runtime.

3. We improve the effectiveness and usability of the developed system. To do this we extend the proposed framework with mechanisms in order to anticipate the next user action. In addition, we provide an end-user tool to manage the private user information and mechanisms to ensure the privacy and the security of the system. This contribution is obtained by achieving the following goals:

3.1. Studying the requirements that a context-aware system must fulfil.
3.2. Studying the different approaches that have been proposed in order to both adapt systems to user behaviour and ensure the privacy and the security of the system.
3.3. Implementing mechanisms for both anticipating the next user action and ensure the privacy and the security of the system.

4. We integrate our approach with a MDD method [24] previously implemented by our research group what allows us obtain fully functional context-aware pervasive systems. This MDD method provides us with PervML, which is a modelling language for specifying pervasive systems, and a code generation strategy that allows us generate the functional system from the specified models. We also define a methodological guidance that guides developers from the description of the system by means of the models that we propose to its deployment. To do this, the following tasks are carried out:

4.1. Studying the abstractions proposed by PervML and how they can be integrated with the defined context models.
4.2. Implementing an automatic transformation to automatically transform the PervML models extended with Context support into OWL.
    4.2.1. Identifying the mappings between the abstractions of the PervML models and the primitives of the OWL metamodel.
    4.2.2. Defining these correspondences by using the ATL tool.
4.3. Integrating our framework with the code generation strategy of the MDD method.

4.4. Defining a precise sequence steps to follow in order to develop a context-aware pervasive system.

Finally, it is worth noting that the code generation strategy is one of the main reasons for using the MDD method in this master thesis. Another reason for this choice is the extensive knowledge that the research centre in which this master thesis has been developed has about this method. This aspect has facilitated an exhaustive analysis of the best way to carry out the integration.

1.4 Research Methodology

In order to perform the work of this master thesis, we have carried out a research project following the design methodology for performing research in information systems as described by [25] and [26]. Design research involves the analysis of the use and performance of designed artefacts to understand, explain and, very frequently, to improve on the behaviour of aspects of Information Systems [26].

The design cycle consists of 5 process steps: (1) problem statement, (2) solution suggestion, (3) design and development, (4) evaluation, and (5) conclusion. The design cycle is an iterative process; knowledge produced in the process by constructing and evaluating new artefacts is used as input for a better awareness of the problem.

Following the cycle defined in the design research methodology, we started with the statement of the problem (see step 1 of Figure 1): We identified the problem to be resolved and we stated it clearly.

Next, we performed the second step which is comprised of the suggestion of a solution to the problem, and comparing the improvements that this solution introduces with already existing solutions. To do this, the most relevant approaches for developing context-aware pervasive systems were studied in detail.

Once the solution to the problem was described, we design and developed this solution (step 3). To do this, we design a set of models, a context ontology and the mechanisms necessaries to give support to context. Next, we implement a context repository based on the proposed ontology, a Java framework to mange context at runtime, and end-user tools to support the privacy and security of the system.

Once the solution to the problem was developed, we validate our approach by developing several case studies (step 4). The results obtained from the case studies are used to improve the proposed solution.

Finally, we analyzed the results of our research work in order to obtain several conclusions as well as to delimitate areas for further research (step 5).

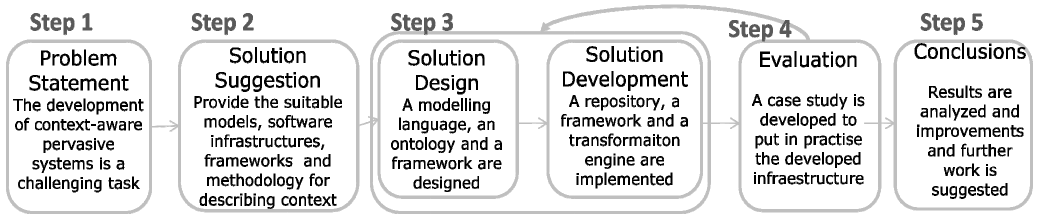

Step 1
Problem Statement
The development of context-aware pervasive systems is a challenging task
Step 2
Solution Suggestion
Provide the suitable models, software infrastructures, frameworks and methodology for describing context
Step 3
Solution Design
A modelling language, an ontology and a framework are designed
Solution Development
A repository, a framework and a transformaiton engine are implemented
Step 4
Evaluation
A case study is developed to put in practise the developed infraestructure
Step 5
Conclusions
Results are analyzed and improvements and further work is suggested

Fig. 1 Research methodology followed in this master thesis

1.5 Thesis Context

This Master's Thesis was developed in the context of the research centre *Centro de Investigación en Métodos de Producción de Software* of the Technical University of Valencia. The work that has made the development of this master thesis possible is in the context of the following research government projects:

- DESTINO: Desarrollo de e-Servicios para la nueva sociedad digital. CYCIT project referenced as TIN2004-03534.
- SESAMO: Construcción de Servicios Software a partir de Modelos. CYCIT project referenced as TIN2007-62894.
- OSAMI Commons: Open Source Ambient Intelligence Commons. ITEA 2 project referenced as TSI-020400-2008-114.
- Atenea: Arquitectura, Middleware y Herramientas. ProFIT project referenced as FIT-340503-2006-5.
- Ingenieria del Software para el Desarrollo de Sistemas Pervasivos. Explorando Nuevas Vias en la Interaccion Hombre-Máquina. Proyecto financiado por Care Technologies, S.A

1.6 Master Thesis Structure

The rest of this thesis is organized as follows:

- *Section 2* introduces the background and technical overview, in which the concepts and technologies related to the work presented in this thesis are briefly explained.
- Chapter 3 presents a critical analysis of the most well-known approaches for the development of context-aware pervasive systems. In particular, we focus on how these approaches deal with context management. We identify the main limitations of these techniques and conclude with a summary of the main drawbacks in context-aware pervasive systems that should be improved. We take these drawbacks as a base in order to develop the work of this thesis.
- Chapter 4 presents a method for developing pervasive systems. We have integrated our approach with this method in order to develop whole context aware functional pervasive systems
- Chapter 5 explains how we give support to Context management in pervasive systems, both at modelling time and at runtime.
- Chapter 6 describes the adaptation that we carry out in order to according to Context in order to satisfy the needs of users within ever-changing context.
- Chapter 7 presents the global execution strategy that is followed when a context-aware pervasive system is put in execution.
- Chapter 8 introduces the methodological approach for the development of context-aware pervasive systems that is proposed in this master thesis.
- Chapter 9 validates the work of this master thesis by the development of a complete case study by following the proposed approach.
- Chapter 10 presents the conclusions and future work of this master thesis. To do this, this Chapter first summarizes the main master thesis contributions, the publications that this master thesis has produced are next enumerated, and the future work is described.

2. Background and Technical Overview

Research in Pervasive Computing is very diverse since the field itself has not yet been clearly defined. Researchers from different communities make efforts to understand and improve concepts, technologies and applications for research in Pervasive Computing. In this Chapter we try to clarify the definition of the concepts used in context-aware pervasive computing. In order to facilitate the understanding of this master thesis we also briefly explain the technologies used to carry out this work.

2.1 Ubiquitous Computing vs Pervasive System vs Ambient Intelligence

Several terms are used in the published literature for talking about similar concepts. The main differences depend on the context of use: for instance, EEUU vs Europe or Academy vs Industry. According to Mattern [27], while Weiser saw the term ubiquitous computing in a more academic and idealistic sense as an unobtrusive, human-centric technology vision that will not be realized for many years yet, industry (IBM) has coined the term pervasive computing with a slightly different slant [28][29]. In [30] is stated that while researchers in the United States were working on the vision of ubiquitous computing, the European Union began promoting a similar vision for its research and development agenda. The term adopted in Europe is ambient intelligence (coined by Emile Aarts of Philips) which has a lot in common with Weiser's ubiquitous computing vision. This point of view is confirmed by the great number of events and research projects that are organized and/or funded in Europe under this term whose topics clearly matches the ones that are inside the scope of the ubiquitous computing area.

Although subtle differentiations could be done between these terms according to their etymological meanings (nor ubiquitous implies intelligence, neither intelligence implies pervasiveness, etc.), we can state in general that the main idea or vision behind them is the same and, therefore, they can be equally used in this master thesis.

2.2 What is Context?

The term Context is widely used in Computer Science with different meanings. The meaning of Context depends on the work area in which it is defined. For instance:

- In Artificial Intelligence [31], it is defined as everything that affects the computation except from the explicit input and output of the application. According to this definition, we need to precisely determine what we consider explicit and implicit in the system. All what is considered implicit constitute the Context. In this way, Context of a System changes depending on the initial consideration of explicit and implicit elements.
- In Natural Language Processing [32], it is understood as all the knowledge that surrounds a specific statement or assertion. In this area, it is important to avoid the danger of taking things “out of context”. Assertions true in one context might be false in another. Thus, Context in this area is related to the meaning of one sentence with regards to the meaning of other sentences.
- In Operating Systems [33], it defines the minimal set of data used by an operating system task, and which need to be saved in order to allow the task to be interrupted at a given date, and to be resumed at the point it was interrupted.

In AmI computing, the definition of Context is different from the ones presented above. In addition, there is no a generally accepted definition. Some definitions are the following:

- In [34], Context is characterized by the location of use, the collection of nearby people, hosts, and accessible devices as well as such as things over time. Thus, context-aware systems are those that are able to adapt themselves to these aspects.
- In [35], the concept of Context awareness is described as the ability of the computer to sense and act upon information about its environment, such as location, time, temperature, or user identity. Thus, the concept of Context is mainly centred on the characteristics of the user location.
- In [2], the most used definition of Context in AmI systems is presented. Dey defines Context as any information that can be used to characterize the situation of an entity. And an entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves.

- In [36], two classes of Context are identified, namely personal and environmental Context. Examples of environmental context include: the time of the day, the opening times of attractions and the current weather forecast. Personal Context refers to user profiles in which information such as user's interest, attitudes, or beliefs are considered.
- In [37], a distinction between the user's context and the system's context is done. The user's context provides the means to determine what to observe and how to interpret the observations. The system's context provides a means to compose the federation of components that observe the user's context.
- In [12], Context is defines as the information about a location, its environmental attributes (e.g., noise level, light intensity, temperature, and movement) and the people, devices, objects and software agents that it contains. Context may also include: system capabilities, services offered and sought, the activities and tasks in which people and computing entities are engaged, and their situational roles, beliefs, and intentions.
- In [38], authors states that Context refers to the physical and social situation in which computational devices are embedded.

As we can see, different definitions of Context are provided in the area of Ambient Intelligence. However, all of them share some ideas about the information that must be considered as Context. Basically, this information includes aspects related to the system and the environment in which it is executed as well as characteristics of the users that interact with it. More specifically, after studying the definition of Context stated by the different authors, we define the context of pervasive systems as the following information:

- Information about system users: name, age, address, native language, user preferences or attitudes, etc.
- Privacy and security policies: information that indicates what actions each user can execute.
- Space information: spatial description of the place where the system is deployed.
- System services: services that the system provides to users.
- System devices: devices deployed in the environment that are used by the system services.
- Temporal information: date and time, holiday, working day, etc.
- Services available for a user in the current time and their state;

- User mobility: where users are and where they can go.
- User actions: what the user is doing at present moment and what the user has done in the past. The actions performed in the past are necessary to predict the next action or to memorize and to reproduce scenes (repetitive sequences of actions within a certain time interval) at the opportune moment.

In the above context information two types of context information can be identified:

1. Those that are available at design time (information about users, privacy and security policies, space information, system services and system devices);
2. Those that are only available at runtime (temporal information, available services and their state, user mobility, and actions that users perform).

2.3 Model Driven Development

The method that we propose in this work to develop context-aware pervasive systems applies the Model-Driven Development (MDD) guidelines [3]. Thus, the systems developed by our method take advantage of the benefits provided by MDD.

MDD is an approach to software design and development that strongly focuses on models and claims that these models can be refined and finally be transformed into a technical implementation.

Interest and focus on models arise today with further emphasis due to recent developments that resulted into the establishment of important, widely known, and recognized standards, particularly those originated from the Object Management Group (OMG) such as the Model-Driven Architecture (MDA) [3] initiative and the Unified Modeling Language (UML) specification [39]; and those originated from Microsoft such as the Software Factories [40] and the DSL tools [41].

On the one hand, MDA tries to raise de abstraction level at which main development occurs, essentially shifting the focus from coding to modelling. In essence, fundamentally relying on models, MDA proposes for the system development life cycle the development of a Platform Independent Model (PIM) of the system that, free from specific platform technological issues, details the structure and behaviour of the system by using UML, which specifies notation and semantics for representing models. Given a chosen technological platform, this PIM is transformed into a Platform Specific Model (PSM) that incorporates all the necessary technological details inherent to the chosen technological platform on which the system is to be implemented. From this PSM, system code foundations are generated for the target technological platform.

This separation of concerns between PIM and PSM allows that with no further modification to the PIM itself, other technological platforms can be easily targeted, since the PIM still represents the desired system structure and functionality with no contamination of technological details. These standards enable reuse of knowledge and artefacts, tools' specialization and interoperation.

On the other hand, a Software Factory, as defined in [40], is *a software product line that configures extensible tools, processes and content […] to automate the development and maintenance of variants of an archetypal product by adapting, assembling and configuring framework-based components.* Thus, Software Factories (SF) focus on the development of similar systems encouraging the reuse of architectures, components and know-how. In order to achieve this reuse, Software Factories integrate several existing techniques. The main activities promoted by Software Factories are:

- Building families of similar software. This activity involves the analysis and design of a common architecture for a set of systems and the development of a framework to support this architecture.
- Assembling components. The construction of a new system implies the use, assembly and/or configuration of the components provided by the framework.
- Developing domain specific languages and tools. Developers use this language in order to describe the specific requirements of a member of the system family. Then, the specific source code is automatically generated.
- Using constraint based scheduling and active guidance. All the steps of the development project must be taken according to a well-defined process properly adapted to the domain.

The main detected differences between MDA and SF are the following:

- SF provide a more precise design process than MDA, since SF explicitly suggests the use of product lines, implementation frameworks, design patterns, etc.
- SF promote the use of Domain Specific Languages, whereas MDA argues the use of UML. While SF recommend the creation of languages for providing to the developers conceptual primitives suitable for a specific kind of systems, the OMG suggests that

UML must be the common language, using its extension capabilities (profiles) when needed.

- MDA, unlike SF, provides languages for model management (UML, MOF, QVT, etc.).
- MDA explicitly promotes the rise in the abstraction level of the modelling languages. In the SF approach this possibility is also took into account, but not is a requirement.

2.3.1 Benefits of applying MDD

MDD has many benefits that are historically well known in the software engineering community. The most important are the following:

- *Productivity*: The code generation tool builds code in a fraction of the time that it takes an engineer to produce the equivalent amount of code. Furthermore, since handwritten code tends to have a great number of syntax errors that engineers must detect and correct a lot of time is wasted. By following a MDD strategy, code is systematically generated free of errors by a tool.
- *Quality*: Large volumes of handwritten code tend to have inconsistent quality because engineers find newer or better approaches as they work. Generated code always increases in quality over time because, when bugs or shortcomings are found, they can be fixed in the code generation tool and automatically applied to the next applications.
- *Adaptability*: By following a MDD strategy a software product is developed independently of implementation technologies. This software product is developed by describing it in a model. To obtain the code for a specific technology a code generator tool is used. If the implementation technology changes, the code generation tool needs to be adapted. Developed applications do not need to be changed because they are abstractly developed by means of models.
- *Maintenance*: The maintenance of handwritten code is frequently a very tedious task: a very simple change in the requirements of an application requires engineers to modify a lot of software artefacts such as user interfaces, logical classes, database tables, documentation, etc. By following a MDD strategy, only the models need to be changed. Changes at implementation level are automatically done.

2.4 Ontology, ontology languages and ontology reasoners

In this work we propose an ontology-based strategy to give support to the development of context-aware pervasive systems.

In Philosophy, Ontology is the study of being or existence and its basic categories and relationships. It seeks to determine what entities can be said to "exist", and how these entities can be grouped according to similarities and differences. We have used ontologies for millennia to understand and explain our rationale and environment. However, only recently have ontologies become a research topic of interest in computer and information science.

In computer science and information science, an ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to define the domain. The philosopher-ontologist, in principle at least, has only one goal: to establish the truth about reality by finding an answer to the question: what exists. In the world of information systems, in contrast, an ontology is a software (or formal language) artefact designed with a specific set of uses and computational environments in mind. An ontology is often something that is ordered by a specific client in a specific context and in relation to specific practical needs and resources. In a widely-quoted definition, an ontology is "… a specification of a conceptualization" [42]. An ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. Thus, it allows a programmer to specify, in an open, meaningful, way the concepts and relationships that collectively characterize some domain.

An ontology mainly contains the following elements:

- Classes: represents the concepts of the ontology
- Attributes: properties that objects (and classes) can have
- Relations: ways that classes and objects can be related to one another
- Individuals: instances or objects of the defined classes

The terms Abox and Tbox are also used to refer to the elements of an ontology. These terms describe two different types of statements in ontologies. Tbox statements describe a system in terms of controlled vocabularies, for example, a set of classes and properties. Abox are Tbox-compliant statements about that vocabulary.Tbox statements are sometimes associated with object-oriented classes and Abox statements associated with instances of those classes. Together Abox and Tbox statements make up a knowledge base (a special kind of database for knowledge management).

Thus, an ontology is an explicit, first-class description. This description can be specified in different languages, such us RDF or OWL, and can be used by different reasoners, such as Racer or Pellet. This is our main reason for building an ontology-based application: we can use a reasoner to derive additional truths about the concepts that we are modelling. There are many different styles of automated reasoner and many different reasoning algorithms for ontologies.

2.4.1 Web Ontology Language (OWL)

Web Ontology Language (OWL) [43] is a semantic markup language for publishing and sharing ontologies on the World Wide Web. In this work we have selected OWL to implement the ontology proposed for context-aware pervasive systems for the following reasons.

The OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

OWL DL does not permit some constructions allowed in OWL Full, and OWL Lite has all the constraints of OWL DL plus some more. The intent for OWL Lite and OWL DL is to make the task of reasoning with expressions in that subset more tractable. Specifically, OWL DL is intended to be able to be processed efficiently by a description logic reasoner. OWL Lite is intended to be amenable to processing by a variety of reasonably simple inference algorithms, though experts in the field have challenged how successfully this has been achieved.

We have select OWL because several reasons:

- It enables automated reasoning
- It has the capability of supporting semantic interoperability to exchange and share context knowledge between different systems, i.e., contexts can be exchanged and understood between different systems in various domains.

- It is also more expressive than other ontology languages such as RDFS
- It is an open W3C standard.

2.4.2 Semantic Web Rule Language (SWRL)

In order to enable the automatic reasoning in our ontology we have used the Semantic Web Rule Language (SWRL) (44). SWRL is a proposal based on a combination of the OWL DL and OWL Lite sublanguages of the OWL Web Ontology Language with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language. The proposal extends the set of OWL axioms to include **Horn-like rules**. It thus enables Horn-like rules to be combined with an OWL knowledge base. A high-level abstract syntax is provided that extends the OWL abstract syntax described in the OWL Semantics and Abstract Syntax document.

The proposed rules are of the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Both the antecedent (body) and consequent (head) consist of zero or more atoms. An empty antecedent is treated as trivially true (i.e. satisfied by every interpretation), so the consequent must also be satisfied by every interpretation; an empty consequent is treated as trivially false (i.e., not satisfied by any interpretation), so the antecedent must also not be satisfied by any interpretation. Multiple atoms are treated as a conjunction. Note that rules with conjunctive consequents could easily be transformed into multiple rules each with an atomic consequent.

Atoms in these rules can be of the form C(x), P(x,y), sameAs(x,y) or differentFrom(x,y), where *C* is an OWL description, *P* is an OWL property, and *x* and *y* are either variables, OWL individuals or OWL data values. For instance, Fig. 2 shows at the top an example of SWRL rule in a human readable syntax, and at the bottom, the same rule in OWL syntax. This rule implies that if two people have the same parent, they will be siblings.

```
antecedent ⇒ consequent
parent(?x,?y) ∧ parent (?z,?y) ⇒ siblings (?x,?z)
--------------------------------------------------------------------
<swrl:Variable rdf:about="#x"/>
<swrl:Variable rdf:about="#y"/>
<swrl:Variable rdf:about="#z"/>
     <swrl:Imp rdf:about="#SiblingRule">
             <swrl:head rdf:parseType="Collection">
             <swrl:IndividualPropertyAtom>
```

```
                <swrl:propertyPredicate rdf:resource="#sibling"/>
                        <swrl:argument1 rdf:resource="#x"/>
                        <swrl:argument2 rdf:resource="#z"/>
                </swrl:IndividualPropertyAtom>
                </swrl:head>
                <swrl:body rdf:parseType="Collection">
                <swrl:IndividualPropertyAtom>
                <swrl:propertyPredicate rdf:resource="#parent"/>
                <swrl:argument1 rdf:resource="#x"/>
                <swrl:argument2 rdf:resource="#y"/>
                 </swrl:IndividualPropertyAtom>
                 <swrl:IndividualPropertyAtom>
                        <swrl:propertyPredicate rdf:resource="#parent"/>
                        <swrl:argument1 rdf:resource="#z"/>
                        <swrl:argument2 rdf:resource="#y"/>
                 </swrl:IndividualPropertyAtom>
                 <swrl:DifferentIndividualsAtom>
                        <swrl:argument1 rdf:resource="#x"/>
                        <swrl:argument2 rdf:resource="#z"/>
                 </swrl:DifferentIndividualsAtom>
                </swrl:body>
</swrl:Imp>
```

Fig. 2 SWRL rule

2.4.3 Pellet: an OWL-DL Reasoner

In this approach, we use Pellet [45] to derive additional truths about the modelled context information. Pellet is a complete and capable OWL-DL reasoner with acceptable to very good performance, extensive middleware, and a number of unique features. It is written in Java and is open source under a liberal license. It is used in a number of projects, from pure research to industrial settings.

Pellet is the first implementation of the full decision procedure for OWL-DL (including instances) and has extensive support for reasoning with individuals (including conjunctive query over assertions), user-defined datatypes, and debugging ontologies. It implements several extensions to OWL-DL including a combination formalism for OWL-DL ontologies, a non-monotonic operator, and preliminary support for OWL/Rule hybrid reasoning. It has proven to be a reliable tool for working with OWL-DL ontologies and experimenting with OWL extensions.

2.5 Eclipse Platform

We have developed this work in the Eclipse Platform [46]. Eclipse is an open source community, whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. A large ecosystem of major technology vendors, innovative start-ups, universities, research institutions and individuals extend, complement and support the Eclipse platform.

Eclipse has over 60 open source projects that can be conceptually organized into seven different "pillars" or categories:

3. Enterprise Development
4. Embedded and Device Development
5. Rich Client Platform
6. Rich Internet Applications
7. Application Frameworks
8. Application Lifecycle Management (ALM)
9. Service Oriented Architecture (SOA)

Eclipse was initially the IBM IDE for Java development, which was released as free software. Currently, it is the base platform for many other technologies and projects due to its very powerful modular structure and its open nature. Eclipse is organized in a set of first level thematic projects which guides the evolution of more concrete projects. Most of the Eclipse plug-ins are related to software development, for instance, the Eclipse Modeling Project. It is the first level project that unifies the modelling related projects by focusing on the evolution and promotion of model-based development technologies within the Eclipse community by providing a unified set of modelling frameworks, tooling, and standards implementations. All the plug-ins used in this work have been developed in this project. These are the following: the Eclipse Modelling Framework (EMF), the ATLAS Transformation Language Project, the MOFScript Tool and the EMF Ontology Definition Metamodel (EODM).

2.5.1 Eclipse Modelling Framework (EMF)

EMF is a modelling framework and code generation facility for building tools and other applications based on a structured data model. From a model specification described in XMI, EMF provides tools and runtime support to produce a set of Java classes for the model, a set of adapter classes that enable viewing and command-based editing of the model, and a basic editor. Models can be specified using annotated Java, XML documents, or modelling tools like Rational Rose, then imported into EMF. Most important of all, EMF provides the foundation for interoperability with other EMF-based tools and applications. Thus, EMF consists of three fundamental pieces:

- **EMF** - The core EMF framework includes a meta model (Ecore) for describing models and runtime support for the models including change notification, persistence support with default XMI serialization, and a very efficient reflective API for manipulating EMF objects generically.
- **EMF.Edit -** The EMF.Edit framework includes generic reusable classes for building editors for EMF models.
- **EMF.Codegen** - The EMF code generation facility is capable of generating everything needed to build a complete editor for an EMF model.

Besides, three levels of code generation are supported:

- **Model** - provides Java interfaces and implementation classes for all the classes in the model, plus a factory and package implementation class.
- **Adapters** - generates implementation classes that adapt the model classes for editing and display.
- **Editor** - produces a properly structured editor that conforms to the recommended style for Eclipse EMF model editors and serves as a starting point from which to start customizing.

All generators support regeneration of code while preserving user modifications. The generators can be invoked either through the GUI or headless from a command line.

2.5.2 ATL

ATL (ATLAS Transformation Language) is a model transformation language and toolkit developed by the ATLAS Group. In the field of Model-Driven Engineering (MDE), ATL provides ways to produce a set of target models from a set of source models.

Developed on top of the Eclipse platform, the ATL Integrated Environment (IDE) provides a number of standard development tools (syntax highlighting, debugger, etc.) that aims to ease development of ATL transformations. The ATL project includes also a library of ATL transformations.

2.5.3 Mofscript

The MOFScript tool is included in the Generative ModeLling Technologies (GMT) Eclipse project. The objective of this project is "*to produce a set of research tools in the area of MDE (Model Driven Engineering)*". In this context, the MOFScript project "*aims at developing tools and frameworks for supporting model to text transformation*". This subproject has been developed in a development community at SINTEF, supported and tested by the European Integrated Project MODELWARE.

The MOFScript tool is an implementation of the MOFScript model to text transformation language. It provides mechanisms for generating text from MOF-based models, controlling the sequence of execution, string manipulation, easy production of files, traceability of models and generated text, etc. and it is based on the QVT standard.

2.5.4 EODM

EODM (EMF Ontology Definition Metamodel) is built on top of EMF and conforms to the ODM (Ontology Definition Metamodel) standard of OMG (www.OMG.org). It provides a set of programming APIs for programmers and IT specialists. User can create, modify, and navigate RDF/OWL models using EODM, just like other programming implementations of semantic models.

Compared with others tools, the most important differentiation of EODM is that all ontology objects are also EMF model objects which gives EODM the interoperability between RDF/OWL model with other EMF supported models, including models defined using XML Schema, Rational Rose, and annotated Java classes. All these will enable Semantic Web Application developers to develop ontologies using their favourite building tools, import them into EMF, and utilize the comprehensive development facility of Eclipse and EMF. EODM includes transformers to transform between Ecore and RDF/OWL. It also facilitates other tools to treat RDF/OWL ontology models as an EMF model and process and store them as usual.

2.6 OSGi

The *Open Services Gateway initiative (OSGi)* [47] is an "independent, non-profit corporation working to define and promote open specifications for the delivery of managed broadband services to networks in homes, cars, and other environments". The

OSGi specification defines **standardized primitives** that allow applications to be constructed from **small, reusable and collaborative components** (called bundles in OSGi). These components can be composed into an application and deployed.

The core component of the OSGi Specifications is the OSGi Framework. This Framework is divided in four layers:

1. **L0 Execution Environment**: is the specification of the Java environment.
2. **L1 Modules**: defines the class loading policies. The OSGi Framework is a powerful and rigidly specified class-loading model.
3. **L2 Life Cycle Management:** adds bundles that can be dynamically installed, started, stopped, updated and uninstalled.
4. **L3 Service Registry**: provides a comprehensive model to share objects between bundles.

Thus, this framework provides constructs and services that can support many requirements needs for developing context-aware pervasive systems, such as the followings:

- Discovery. OSGi relies device discovery on low-level protocols as Jini, Lonworks or UPnP. The standard specifies the interfaces for Jini and UPnP driver services. When the devices are discovered they can be coupled to device drivers and then used for the system services.
- Adaptation. Adaptation is achieved through dynamic bundle loading and updating, and service lookup. When a new device or service is registered in the framework by a bundle, any other running service can use it. The link is done in runtime.
- Integration. The integration of the software representation of a device and the physical environment relies on low-level technologies. Basically, OSGi uses bridges to the final device drivers. The native device drivers are in charge of the comunication with the physical device.
- Programming Framework. OSGi provides a well defined programming framework around the service concept that separates service description from any possible implementations. As OSGi is Java-based, it is operative system independent. For complex applications, there is a proposal and implementation, of a component model built on top of OSGi.

- Robustness. Dynamic coupling of services and devices is a guarantee of robustness. If a service runs out or a device fails they can be automatically replaced by other elements that provide the same functionality.
- Security. The framework security model is based on the Java 2 specification. OSGi defines a standard service for permission administration. In the framework, a bundle can have a single set of permissions. These permissions are used to verify that a bundle is authorized to execute privileged code. For example, a *FilePermission* defines what files can be used and in what way.

3. State of the Art

In this Chapter, we present an analysis of the most important approaches to develop context-aware systems that have been proposed, paying special attention to context modelling and context management. Therefore, we present in detail those approaches that carry out a context modelling in its development process. They are presented grouped according to division done by Strang and Linnhoff-Popien [4] based on the type of context model that is used in the system. We have also studied other approaches that are not considered in this detailed analysis because they do not provide a context model. These approaches are introduced in a schematic way in Section 3.2.

3.1 Research works that model Context

We present in this Section some of the most important research works that support Context in Pervasive Systems by using a context modelling. We have grouped them according to division done by Strang and Linnhoff-Popien. They divide six context modelling approaches: key-value modelling, markup scheme modelling, graphical modelling (UML, ER, etc.), object oriented modelling, logic-based modelling and ontology-based modelling.

3.1.1 Key-Value models

These models represent the simplest data structure for context modelling. They are frequently used in various service frameworks, where the key-value pairs are used to describe the capabilities of a service. Service discovery is then applied by using matching algorithms which use these key-value pairs. Various approaches exist where contextual aspects are modelled in by using this type of model:

- The **Context Toolkit** [2] handles context in simple attribute-value-tuples, which are encoded using XML for transmission. It takes a step towards a peer-to-peer architecture but it still needs a centralised discoverer where distributed sensor units (called widgets), interpreters and aggregators are registered in order to be found by client applications. The toolkits object-oriented API provides a superclass called *BaseObject* which offers generic communication abilities to ease the creation of own components. The Context Toolkit has a discovery mechanism to search for and find appropriate sensors at runtime. It also can handle information from different data sources and offers facilities for both context aggregation and context interpretation. The context aggregators (former called context servers) are responsible for composing context of particular entities by subscribing to relevant widgets, context interpreters provide the possibility of transforming context, e.g., in a simple case returning the corresponding e-mail address to a passed name. Like widgets aggregators and interpreters inherit communication methods from the *BaseObject* supperclass and have to be registered at the discoverer in order to be found.

3.1.2 Markup scheme models

All markup based models use a hierarchical data structure consisting of markup tags with attributes and content. *Profiles* represent typical markup-scheme models. CSCP [19] (which is explained further) use this type of models. FAWIS [6] also use a markup scheme models, however, although the context model is flexible enough to be applied to different scenarios, its methodology is focused on the adaptation of Web-based Information Systems via the transformation of the presentation and navigation.. This context model specifies context by a set of profiles, each one describes an autonomous aspect of the context itself (e.g., the user, the location, the device, etc.). A profile is characterized by a set of simple or complex attributes, and each instantiation of a profile has a fixed set of attributes, assuming also the presence of null values. Profiles can be combined to represent a context at different levels of detail; however, the model does not allow the expression of constraints between sets of attributes or set of profiles to avoid meaningless combinations. The system mainly considers the user-profiling issues of the context modelling problem, while leaving all the other aspects not formally described.

3.1.3 Graphical models

Graphical models as the Unified Modelling Language (UML) or the Object-Role Modelling (ORM) are suitable for modelling context by graphical models. Various approaches exist where contextual aspects are modelled in by using a graphical model:

- CML: (Pervasive, Autonomic, Context-aware Environments) project [7] proposes an extension to ORM for context modelling purposes. The context model has a graphical notation (CML): the possible contexts for a target application are rendered by a directed graph composed by a set of entities, describing objects, and their attributes, representing the entity properties. As Fig. 3 shows, different kinds of associations connect an entity to its attributes or to other entities.

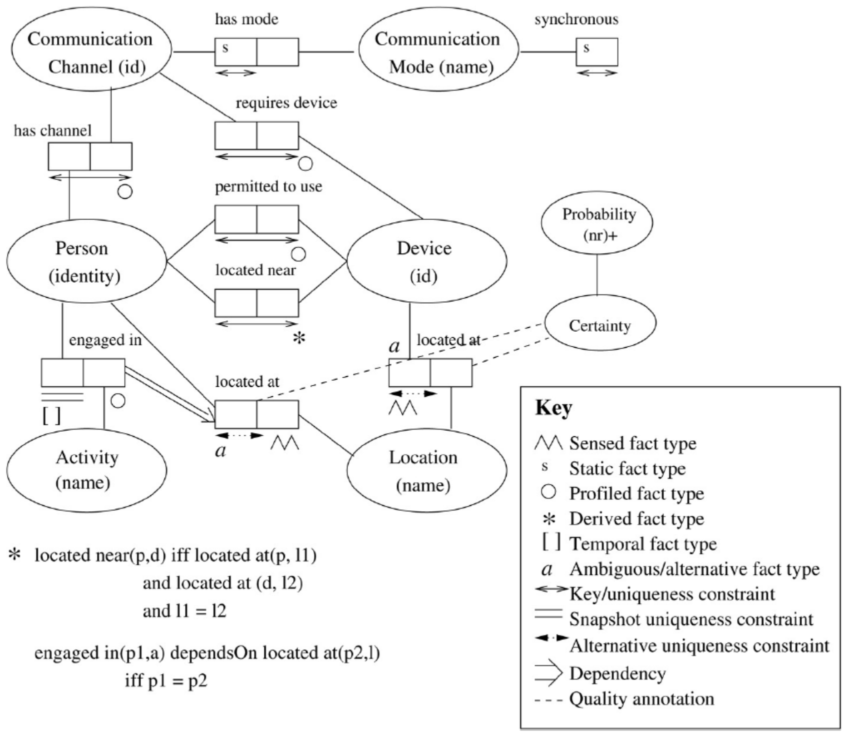


Fig. 3 Example of CML model

Besides, it presents a different architecture in which context data are stored in a database. The meta-data (temporality, quality, etc.) are added either to context data or to relations between them. The authors indicate clearly that they did not have a look at issues such as scalability or performance. In addition, the authors proposed a model driven approach to develop context-aware applications based on their object-oriented Context Modelling Language (CML). They proposed a semi-automated procedure to map their context models to context management systems based on relational databases.

- ContextUML [8] is a UML-based model for the specification and model-driven development of *Context-aware Web Services (CAS).* It provides meta-models for context, services and context-aware mechanisms that associate both with each other. The meta-model for context information is centred around the *Context* class that represents generic context information. It is further sub-classed into *AtomicContext* (simple, low-level context, directly provided by a context source) and *CompositeContext* (high-level context, aggregates multiple atomic or composite contexts). The classes ContextSource, ContextService and *ContextServiceCommunity* model the resources from which context information is retrieved. Context UML allows the modelling of derivation rules, but does not include means to model user privacy. Although this model is integrated into a complete meta-model for implementing CAS, it is restricted to them and lacks grounding in real application scenarios.

- Ayed introduces in [9] a MDD approach for the development of Context-Aware applications which is based on six phases that are shown in Figure 4. They propose a UML profile that allows the designers to apply this MDD approach. The profile is highly based on class diagram extensions and most of the stereotypes (Context, ContextState, Optional, VariableStructure, PhysicalSensor, etc.) extend the Class meta-class. They split up transformations according to technical concerns (a separate transformation for each PSM) and they also recommend decompose transformations according to non-functional concerns. Ideally, each transformation should address only one non-functional concern. As a result, they propose a chain of transformations that need to be applied subsequently to weave all non-functional concerns into the application model.

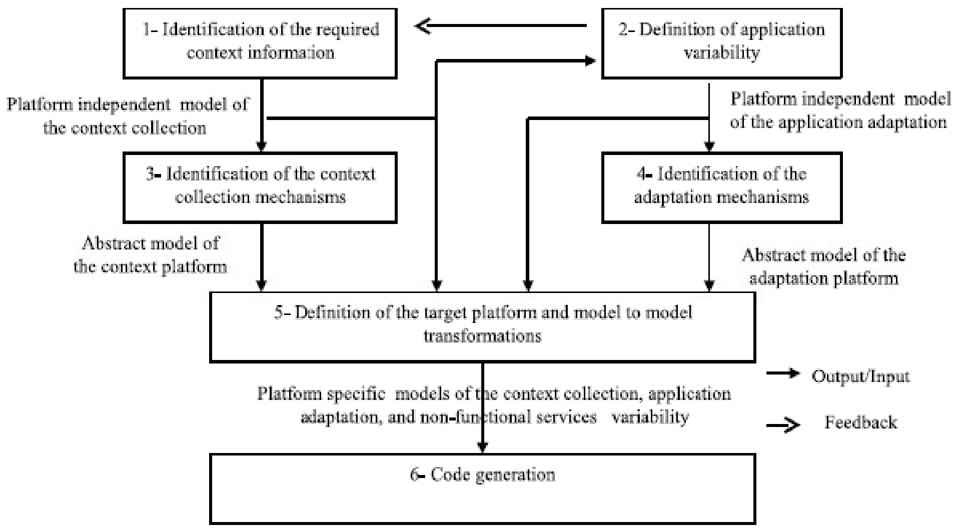

1- Identification of the required context information
2- Definition of application variability
Platform independent model of the context collection
Platform independent model of the application adaptation
3- Identification of the context collection mechanisms
4- Identification of the adaptation mechanisms
Abstract model of the context platform
Abstract model of the adaptation platform
5- Definition of the target platform and model to model transformations
Output/Input
Platform specific models of the context collection, application adaptation, and non-functional services variability
Feedback
6- Code generation

Fig. 4 MDD phases for the development of Context-Aware applications proposed by Ayed et al.

3.1.4 Object oriented models

Modelling context by using object-oriented techniques offers to use the full power of object orientation (e.g., encapsulation, reusability, inheritance). Existing approaches use various objects to represent different context types (such as temperature, location, etc.), and encapsulate the details of context processing and representation. Access the context and the context processing logic is provided by well-defined interfaces. Some projects that use this model are the followings:

- Hydrogen [10]: Its context acquisition approach is specialised for mobile devices. While the availability of a centralised component is essential in the majority of existent distributed content-aware systems, the Hydrogen system tries to avoid this dependency. It distinguishes between a remote and a local context. The remote context is information that another device knows while the local context is knowledge that our own device is aware of. When the devices are in physical proximity they are able to exchange these contexts in a peer-to-peer manner via WLAN, Bluetooth, etc. Both local and remote context are made up of context objects. *Hydrogen* cannot offer persistent storage possibilities due to limited memory resources a peer-to-peer network of mobile devices.

  The architecture consists of four layers which are all located on the same device:
  - The *Adaptor layer* is responsible for retrieving raw context data by querying sensors. This layer permits a sensor's concurrent use by different applications.
  - The *Management layer* makes use of the Adaptor layer to gain sensor data and is responsible for providing and retrieving contexts.
  - The *Context server* offers the stored information via synchronous and asynchronous methods to the client applications.
  - *Application layer*, where the appliance code is implemented to react on specific context changes reported by the context manager. Due to platform and language independency, all inter-layer communication is based on a XML-protocol.
- The *CORTEX [5]* system is a context-aware middleware. The architecture is based on the *Sentient Object Model* which was designed for the development of context-aware applications in an ad-hoc mobile environment. The model special suitability for mobile applications depends on the use of STEAM, a location-aware event-based middleware service designed specifically for ad-hoc wireless networking environments. A sentient

object is an encapsulated entity consisting of three main parts, *Sensory capture*, *Context hierarchy* and *Inference engine*. Via interfaces a sentient object communicates with sensors which produce software events and actuators which consume software events. As Figure 5 shows, sentient objects can be both producer and consumer of another sentient object. Own sensors and actuators are programmed using STEAM. For building sentient objects, a graphical development tool is available which allows developers to specify relevant sensors and actuators, define fusion networks, specify context hierarchies and production rules, without the need to write any code. Since only one context is active at any point in time (concept of the *active context*) the number of rules that have to be evaluated are limited. Thus efficiency of the inference process is increased. The inference engine component is based on C Language Integrated Production System (CLIPS). It is responsible for changing application behaviour according to the current context by using conditional rules.

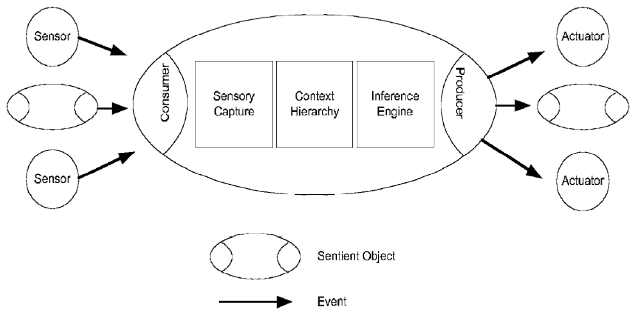


Fig. 5 Example of Sentient Object Model

3.1.5 **Logic based models**

Logic-based models have a high degree of formality. Typically, facts, expressions and rules are used to define a context model. A logic based system is then used to manage the aforementioned terms and allows adding, updating and removing new facts. The inference (also called reasoning) process can be used to derive new facts based on existing rules in the systems. The contextual information needs to be represented in a formal way as facts. CASS is centralized middleware for small portable devices that use logic-based models. CASS [11] offers a high-level abstraction on context sensed by appropriate distributed low-level sensors. The middleware contains an Interpreter, a ContextRetriever, a Rule Engine, a SensorListener and a knowledge base. By means of these components CASS manages both time and space, taking into account the context history. Besides, this middleware can derive high-level context basing on its rule engine and its knowledge base. Its knowledge base contains rules that are queried by the rule engine to find goals using the so-called forward chaining technique. As these rules are stored in a database separated from the interpreter neither recompiling nor restarting of components is necessary when rules change.

3.1.6 Ontology based models

Ontologies represent a description of set of concepts and the relationships among them. Therefore, ontologies are a very promising instrument for modelling contextual information due to their high and formal expressiveness and the possibilities for applying ontology reasoning techniques. Some of the most relevant systems that use ontologies to model context are the followings:

- Context Broker Architecture (*CoBrA*) [13] uses an own OWL-based ontology approach, namely COBRA-Ont that extends the SOUPA ontology [12]. CoBrA is an agent-based architecture for supporting context-aware computing in smart meeting rooms. This Context Broker is composed by a Context Knowledge Base, a Context Inference Engine, a Context Acquisition Module and a Privacy Management Module. *CoBrA* maintains and manages a shared contextual model on the behalf of a community of agents. These agents can be applications hosted by mobile devices that a user carries or wears (e.g., cell phones, PDAs and headphones), services that are provided by devices in a room (e.g., projector service, light controller and room temperature controller) and web services that provide a web presence for people, places and things in the physical world (e.g., services keeping track of peoples and object whereabouts).
- The SOCAM (Service-oriented Context-Aware Middleware) project introduced by Gu et al. [14] is an architecture for building context-aware mobile services. The SOCAM authors divide a pervasive computing domain into several sub-domains and then define each sub-domain in OWL to reduce the complexity of context processing. The richness and flexibility of this model is not complemented by a proper constraining mechanism; the model does not offer explicit ways to limit the number of expressible contexts. The context-aware mobile services are located on top of the architecture, thus, they make use of the different levels of context and adapt their behaviour according to the current context. SOCAM has also implemented a context reasoning engine that reasons over

the knowledge base. The tasks of engine include inferring deduced contexts, resolving context conflicts and maintaining the consistency of the context knowledge base. Different inference rules used by the reasoning engine can be specified.

- CoDaMoS [15]: The CoDaMoS propose a extremely general ontology-based context model. Sets of extensible ontologies are exploited to express contextual information about user, environment and platform in both systems. CoDaMos adds also support for service description. As SOCAM, the richness and flexibility of this model is not complemented by a proper constraining mechanism; the model does not offer explicit ways to limit the number of expressible contexts.
- U-Learn3 [16]: this ontology-based context-model is focused on the support of learning. The learner and the learning content are described by two ontologies (learner ontology and content metadata) and a rule-based system provides a content-to-learner matching mechanism. The content can be a service or a set of data. The data can be enriched by adding content metadata, the user's context described by the learner ontology and the matching can be used to select the relevant data depending on the context. Yet, the system seems at an early stage of development, and the formalization not complete: the learner and learning-content ontologies seem very general and not clearly specified, while the matching rules are not described in the available papers. The authors claim to support sensor integration without providing enough details to actually evaluate the contribution.
- The Context Managing Framework presented by Korpipää et al. [17] is comprised in four main functional entities: the context manager, the resource servers, the context recognition services and the application. Whereas the resource servers and the context recognition services are distributed components, the context manager represents a centralised server managing a blackboard. It stores context data and provides this information to the client applications. This framework also offers various processing facilities. The resource servers' tasks are complex. First they gather raw context information by connecting to various data sources. After the preprocessing and feature abstraction crip limits and fuzzy sets are used for quantisation. But now the data are delivered by posting it to the context manager's blackboard. The context recognition services are used by the context manager to create higher-level context object out of context atoms. In this vein new recognition services are easy to add.

3.1.7 Hybrid models

The hybrid models use more than one of the above presented models.

- The Gaia project [18] is a middleware infrastructure that extends typical operating system concepts to include context-awareness. Thus, it is designed to facilitate the construction of applications for smart spaces, such as smart homes and meeting rooms. In Gaia, the context is represented in a special manner, namely through 4-ary predicates (<ContextType>, <Subject>, <Relater>, <Object>) written in DAML+OIL. Besides, the Gaia's context is processed by performing first-order logic operations. It aims at supporting the development and execution of portable applications for active spaces. *Gaia* exports services to query and utilize existing resources to access and use current context. It consists of a set of core services and a framework for building distributed context-aware applications. Gaia's event manager service enables applications to be developed as loosely coupled components, and can provide basic fault tolerance by allowing failed event producers to be automatically replaced. Gaia's remaining four services support various forms of context-awareness, and include: (i) a context service, which allows applications to find providers for the context information they require, (ii) a presence service, which monitors the entities entering and leaving a smart space (including people as well as hardware and software components), (iii) a space repository, which maintains descriptions of hardware and software components, and (iv) a context file system, which associates files with relevant context information and dynamically constructs virtual directory hierarchies according to the current context. Heterogeneity, mobility and component configuration can all be supported by Gaia in limited forms; however, privacy is not addressed by any of the basic services.
- CSCP [19] model the context by a Markup scheme model. In CSCP the authors present a Mobility Portal: a web portal providing an adaptive web interface, reacting to user channel, device and user profile. The context model represents profile sessions and is based on RDF; it does not impose any fixed hierarchical structure for the context notion, thus inherits the full flexibility and expressive power of RDF. The instantiation of the model allows one to represent a single structured session profile (i.e., a point in the space of possible contexts) with information about the device, the network, and the user of the considered session.
- COMANTO [20]: the authors propose a hybrid context modelling approach to handle context objects and context knowledge. For the first purpose, a location-based context

model is formalized for considering both fixed (e.g., regions, streets, etc.) and mobile location data (e.g., people, vehicles). For the second purpose the general COMANTO ontology is proposed as a public context semantic vocabulary supporting efficient reasoning on contextual concepts (such as users, activities, tools, etc.) and their associations. The ontology is used to collect a structured semantic representation about generic context information and is not domain-, or application-oriented. COMANTO provides a general purpose and very expressive formal model, although lacking the possibility to discard useless contexts.

3.2 Other Studied Approaches

In order to perform the detailed analysis presented in this chapter, we have studied other approaches that do not consider context modelling in their development process. These approaches are mainly focused on developing context-aware pervasive systems by using ad-hoc solutions. From these approaches it is worth mentioning the following ones:

- **The AMIGO project** [21] aims to develop open, standardized, interoperable middleware for pervasive systems, restricting the focus solely to the networked home environment. This project seeks to combine multi-agent techniques with semantic web services to create a system that can adapt to the user’s current context. The project proposes that groups of agents may join in a context-aware service composition process. The project follows the paradigm of Service Orientation, which allows developing software as services that are delivered and consumed on demand. The components in the Amigo Open Source Software can be divided into three main parts:
    - The Base Middleware contains the functionality that is needed to facilitate a networked environment. It provides independence for existing hard- and software, and allows that new services can be discovered and composed
    - The Intelligent User Services provide services for users, context information, combine multiple sources of information and make pattern-based predictions. This part allows that information is tailored to user profiles and adapts to the user's situation and changes in the context.
    - The Programming and Deployment Framework contains modules that facilitate the development of Amigo-aware services.

- **The MavHome project** [22] is focused on creating a home that acts as an intelligent agent. The MavHome architecture is designed with modular components and has four cooperating layers: The decision layer, information layer, communication and physical layer. The decision layer is in charge of selecting actions to take based on information it receives from other layers. The information layer works to collect, maintain, and generate information useful for decision making. The communication layer exists to route communications between the users and the house and the house and external resources. Finally, the physical layer is the actual hardware devices in the house. Together, these layers work to create a proactive smart environment. The most common data source is low-level sensor information. The project suggests the role of prediction algorithms based on the inhabitant actions in order to automatically perform repetitive tasks for the inhabitant.

These methods use ad-hoc solutions for managing context information. None of them propose solutions of a high level of abstraction to model context. In addition, these approaches neither provide mechanisms for reasoning about the context information obtained at runtime nor deduce new context information.

3.3 Analysis and Conclusions

In the literature several definitions of the term *context* can be found [34, 36, 35, 12, 37, 38] and several context-aware systems have been designed and implemented. From them, we have established a set of requirements that every context-aware systems must fulfil in order to compare the presented approaches with our approach. These requirements are the followings:

- Context management: indicates the capture and store of context to later retrieval.
- Context reasoning: indicates whether the context model enables reasoning on context data to infer properties or more abstract context information (e.g., deduce user activity combining sensor readings).
- Automatic Learning and service execution: the system, by observing the user behaviour, individual experiences of past interactions with others, or the environment, can derive and anticipate to the next actions that a user want execute.
- Context history: is the history of previous contexts part of (relevant for) the context itself, i.e., the current context state depends on previous ones, or is the context a pure snapshot of the user's current environment.
- Privacy and security: protection of the privacy of users by establishing and enforcing user defined policies, in such a way that a user can only execute the services allowed by its policy.
- Support for heterogeneity: support for hardware components ranging from resource-poor sensors, actuators and mobile client devices to high-performance servers, and a variety of networking interfaces and programming languages.
- Service Discovery: the system is capable of discovering new services that are adding to the system.
- Ease of deployment and configuration: The hardware and software components of the context- aware system are easily deployed and configured to meet user and environmental requirements, potentially by non-experts (for example, in "smart home" environments).
- Methodological guidance: provide a precise and efficient sequence of well-defined steps to deploy a context-aware pervasive system.

Table 1 shows a resume of all the approaches that have been presented in this chapter. We also distinguish in this table the approaches that only give support to context and those that also give support to development of functional Context-aware Pervasive Systems. In addition, we indicate in this table the context model that each approach use, in other words, the class of the conceptual tool used to capture the context (key-value-, mark-up scheme-, logic-, graph-, ontology-based).

First of all, as we can see in Table 1, almost all the current approaches to develop context-aware systems permit context management, however, only in a few cases they provide facilities to derive or interpret knowledge from context and automatic learning to execute tasks without explicit user request.

In addition, only a few of them provides support for the privacy of context and the security of the system neither the services discovery in context-aware system, that are both important requirements in this type of systems.

Furthermore, very few of the presented approaches provide methodological guidance to develop a system by following that approach neither to deploy and put into operation the developed system, and only our approach provides a precise and detailed method for both developing and deploying a functional context-aware pervasive system.

Our proposal tries to cover the whole process of development of a full context-aware pervasive system, from its description to its deployment. To achieve this, we present in this work a hybrid approach to develop full functional context-aware pervasive systems. We propose a set of graphical OO[1] models for capturing the requirements of the system and its context available at modelling time, and a context ontology for managing context at runtime. Thus, on the one hand, we provide to developers with an intuitive model to specify the requirements of the system at a high level of abstraction; on the other hand, we provide one of the best choices to model context, because ontologies guarantee a high degree of expressiveness, formality and semantic richness, as well as facilitate reasoning, interoperability and reusing context [12, 4]. In addition, we provide a transformation engine to automatically obtain an OWL context repository based on this ontology from the models specified by developers. Lastly, we provide an infrastructure that, by using this repository, automatically manages context, derives knowledge from it and anticipates the next user action. This infrastructure is also in charge of ensuring the privacy and security of the system.

[1] The models that we propose are object oriented and also graphical. Thus, we have classified these hybrid models as graphical OO.

	Context Model	Context Management	Context Reasoning	Automatic Service Execution	Context history	Privacy & Security	Support for heterogeneity	Service discovery	methodological development guidance	Easy to deploy a system – guidance	Full functional system (F)/ only context (C)
Context Toolkit	Key-value	√	√	✗	√	✗	√	√	✗	✗	C
FAWIS	Markup	√	✗	√	✗	✗	√	✗	✓	✗	C
CML	Graphic	√	√	√	√	√	√	✗	✓	✗	C
Context UML	Graphic	✓	✓	✓	✗	✗	√	✗	✓	√	C
Ayed	Graphic	√	✓	✓	✓	✗	√	✗	√	✗	C
Hydrogen	OO	√	✗	✗	✗	✗	✓	✗	✗	✓	C
CORTEX	OO	√	√	✓	✗	✗	✓	✗	✗	✓	C
CASS	Logic	√	√	✗	√	✗	✓	√	✗	✓	C
CoBrA	Ontologies	√	√	√	√	√	✓	✗	✗	✓	C
SOCAM	Ontologies	√	√	✗	√	✗	✓	√	✗	✗	C
CoDaMoS	Ontologies	√	√	✗	✗	✗	✓	√	✗	✗	C
COMANTO	Ontologies	√	√	✗	✗	✗	✓	✗	✗	✗	C
U-Learn3	Ontologies	√	✗	✗	✗	✗	✓	✗	✗	✗	C
Context Managing Framework	Ontologies	√	√	✗	✗	✗	✓	√	✗	✗	C
Gaia	Key-value & Logic	√	✗	✓	√	✓	✓	√	✗	✓	F
CSCP	Markup& Ontologies	√	✗	✗	✗	✗	✓	✗	✗	✓	C
Amigo	No model	√	✗	√	√	✗	√	√	✗	✓	F
MavHome	No model	√	√	✓	√	✗	✓	√	✗	✗	F
Our approach	Graphic, OO & Ontologies	√	√	✓	√	√	√	√	√	√	F

Table 1. Summary of Requirements to support context-aware pervasive systems

(Key: √ = comprehensive, ✓ = partial, ✗ = none)

Furthermore, the integration of our approach with the method developed in our research centre provides us with a MDD strategy that allows generating the functional system from the specified models. This strategy provides us with technological independence and automatic service discovery. Finally, we define a precise methodological guidance to carry out the whole process, from the specification of the models to the put into operation of the functional system. This guidance is easy to follow by developers because to develop the system they only have to specify it by using the models and then we automatically transform these models into the corresponding context-aware functional pervasive system.

In conclusion, as we can see in Table 1, none of the presented approaches fulfil all the enumerated requirements; our approach, however, allow us to fulfil all these requirements.

4. A MDD Method for Developing Pervasive Systems

The work developed in this master thesis proposes an approach to develop context-aware pervasive systems. In particular, this work proposes a hybrid modelling approach and an infrastructure to give support to context in pervasive systems. In addition, this work has been integrated with a MDD method [24] for developing pervasive systems that has been proposed in our research centre. The main reasons for this are both our extensive knowledge of this method and the code generation capacities that this method provides. These capacities allow us to automatically obtain an implementation of the pervasive system in Java code. This MDD method also proposes a Pervasive Modelling Language (PervML) for specifying pervasive systems. This language allows us to specify the services and the devices of the system. Thus, to understand our approach, we explain this method in this chapter. To do this, we first introduce PervML and next we explain the code generation strategy to obtain the code of the specified system.

4.1 PervML

PervML is a domain-specific modelling language for specifying pervasive systems using conceptual primitives (Service, Trigger, Interaction, etc.) that are suitable for this domain. This language allows us to specify context-aware pervasive systems in a platform and technology independent way.

As Fig. 6 shows, PervML promotes the separation of roles in Pervasive System Analysts and Pervasive System Architects. On the one hand, System Analysts capture system requirements and describe the pervasive system using the service metaphor as the main conceptual primitive. They specify three models which are shown in Fig. 6: the Services Model, the Structural Model, and the Interaction Model.

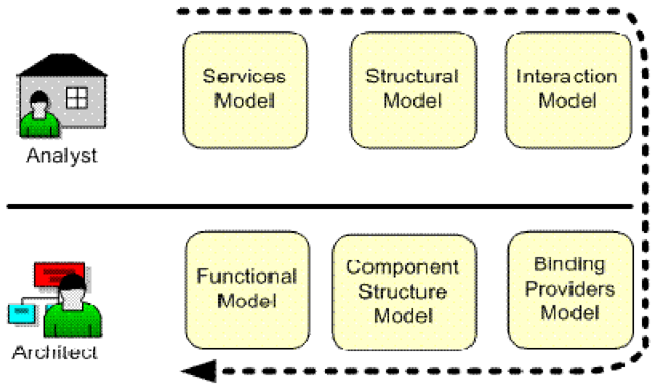


Fig. 6 PervML models

The **Services Model** describes the kinds of services that are provided in the system. The diagram that represents this model in Fig. 7 shows that the system provides services for controlling the lighting, for managing the security, etc. Additionally to the information shown in Fig. 7, the description of a kind of service includes (1) the operation and pre and post conditions (which are expressed using the Object Constraint Language (OCL)) for every operation, (2) a Protocol State Machine (which indicates the operations that can be invoked in a specific moment), and (3) triggers (which allow specifying the proactive behaviour of the services).

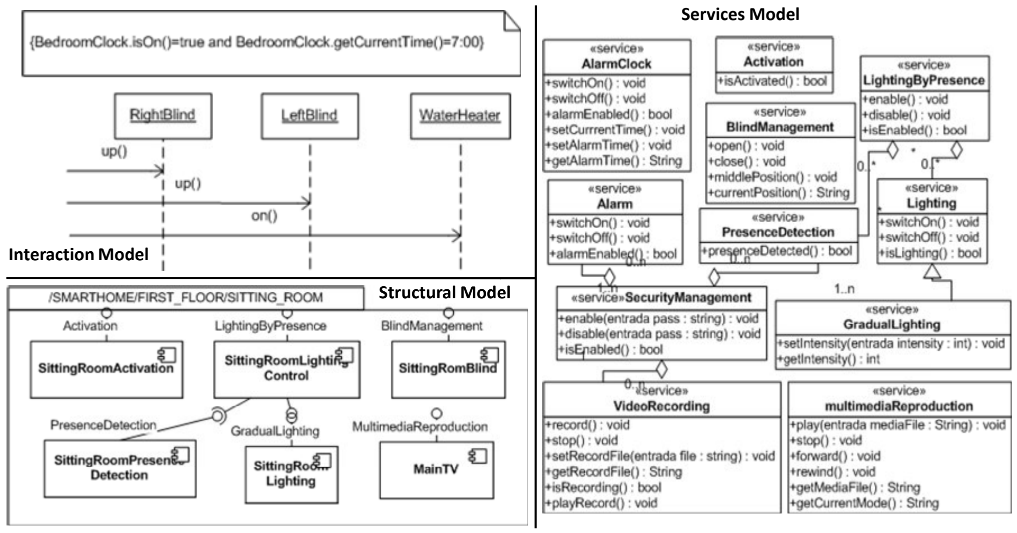
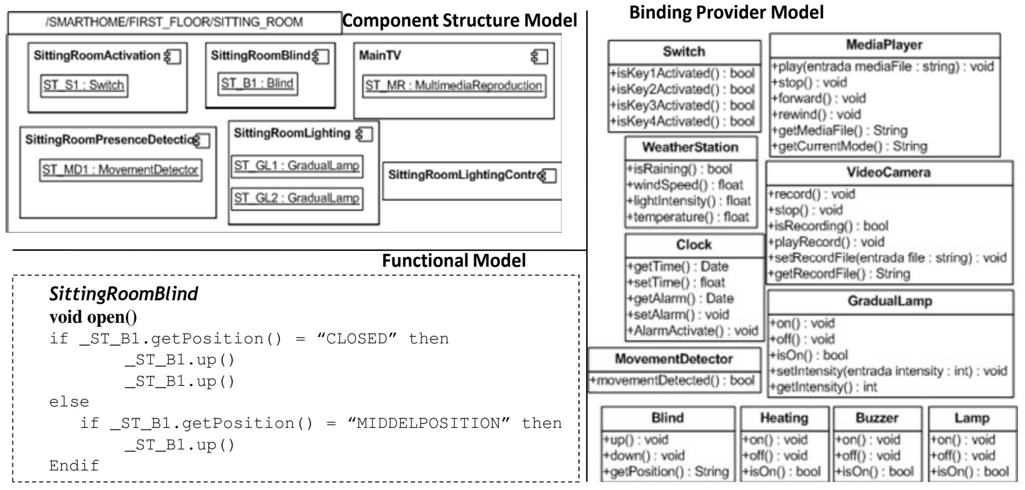


Fig. 7 A partial view of the System Analyst models

The **Structural Model** is used to indicate the instances of every type of service which are provided in each location of the environment by the system. We call to these instances *Components* because they are represented as UML 2.0 components and the type of service that each one provides is depicted as an interface. For example, Fig. 7 shows the *SittingRoomBlind* Component that provides the *BlindManagement* service. Dependency relationships between components can be included to specify that one component uses the functionality provided by another. In order to not overload this figure only the sitting room components are shown. For instance, we have defined several components such as the *SittingRoomLightingControl* which uses the functionality provided by the *SittingRoomPresenceDetection* and the *SittingRoomLighting* components, defined in the model too.

The **Interaction Model** specifies the communication that is produced as a reaction to some system event. Every interaction is described by a set of UML 2.0 sequence diagrams. Thus, analyst identifies the components of the Structural Model that participate in the interaction, defines the messages that the components must interchange and specifies the condition that triggers the interaction by using OCL. The actions described in the diagram will be executed when the condition is satisfied. Fig. 7 shows the interaction that is in charge of opening the blinds and switching on the water heater when the alarm clock goes off at 7.00 a.m.



SittingRoomBlind
void open()
```
if _ST_B1.getPosition() = “CLOSED” then
        _ST_B1.up()
        _ST_B1.up()
else
   if _ST_B1.getPosition() = “MIDDELPOSITION” then
        _ST_B1.up()
Endif
```

Fig. 8 A partial view of the System Architect models

On the other hand, System Architects specify which devices and/or existing software systems support the system services. We refer to these elements (devices and software systems), as binding providers because they bind the pervasive system with its physical or logical environment. System Architects specify three models (the Binding Providers

Model, the Component Structure Model, and the Functional Model) which are shown in Fig. 8.

The **Binding Providers Model** describes the different kind of devices that are used in the system. Fig. 8 shows some of the binding providers for our smart home. For instance, the diagram specifies that the *MovementDetector* sensor provides an operation to know if it detects some movement.

The **Component Structure Model** is used to assign devices and software systems to the system components. For instance, the *SittingRoomLighting* uses the *ST_GL1* and *ST_GL2* devices that are instances of the *GradualLamp* Binding Provider. Besides, the same device can be used for different components.

The **Functional Model** specifies the actions that are executed when an operation of a service is invoked. These actions are specified using the Action Semantics Language (ASL) of UML. Every operation provided for every component must have associated a functional specification. Fig. 8 shows the actions that are executed when the *open* operation of the *SittingRoomBlind* Component is invoked.

4.2 Java Code Generation

In order to automatically generate an implementation of the system specified by using PervML, a model to Java code transformation was defined and implemented.

Following the MDD guidelines, different transformation approaches can be selected to achieve this goal. One key point of these approaches is the number and nature of intermediate models (refined PIMs or PSMs) that must be produced while executing the transformation. In this work, it was selected a straight approach where a PervML model is directly transformed into OSGi bundles in source code format (Java code and Manifest files) that constitute the pervasive system. This approach provides us with some advantages because only one model-to-text transformation is carried out. The advantages are:

1. There are fewer assets (just one) to update and keep synchronized than in other approaches with intermediary models. Therefore, the potential modifications are isolated and completely focused. This characteristic implies that the debugging task is easier than with other approaches.

2. The overall development time of the transformation is considerably reduced, since fewer assets need to be created and maintained.
3. The resulting transformation is faster than the approaches that use PSMs since this approach does not uses intermediate steps or intermediate products.

However, the selected approach has one main drawback, too. The drawback is that the abstraction gap is deal with only one step, so the transformation could be quite complex if the gap is wide. To solve this, the Software Factories guidelines were followed. An implementation framework (which is explained in the next section), that increases the abstraction level of the target technology, was developed. Thereby, the abstraction gap was considerably reduced.

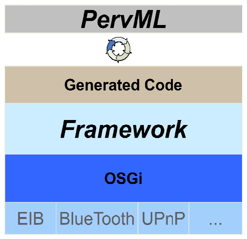


Fig. 9 Our transformation approach

Thus, as Fig. 9 shows, in this approach, first the system must be specified using PervML. Next, this specification is transformed automatically into the java code that conforms the system and that extends the implementation framework code built on top of the OSGi middleware. The OSGi middleware was selected as the implementation technology since it has bridges to many of the technologies used in home automation systems and provides high-level implementation constructs. This middleware help us significantly to fill the abstraction gap between the domain-specific language and the target implementation technology.

Next, we explain how the system java code is obtained from PervML models. However, since the obtained code extends the implementation framework, we present it first.

4.2.1 Implementation Framework for Building Pervasive Systems

The proposed implementation framework raises the abstraction level of the target platform by providing similar constructs to those defined by the primitives proposed in the PervML models. Moreover, the framework encapsulates the common functionality and structure of the elements that are generated by the development method. Therefore, the amount of code that must be generated is significantly reduced.

The framework applies the Layers Architectural Pattern [48], which allows us to organize the system elements in layers with well defined responsibilities. Hence, in order to provide facilities for integrating several technologies (EIB networks, web services, etc.) and for supporting multiple user interfaces, the framework architecture has been designed in three layers (see Fig. 10):

- *The Driver Layer*, which is in charge of managing the access to the devices and external software. In order to achieve the goals of this layer, drivers should be manually developed for dealing with manufacturer-dependent issues. Following this strategy, the drivers adapt the specific mechanisms for using the binding providers (the drivers or APIs supplied by the manufacturers), so a common interface is provided for every kind of binding provider.
- *The Logical Layer*, which is in charge of giving support to the generation of the system logic code. It was subdivided into two sub layers: *the Communications sub-layer*, which gives support to code generation about binding providers; and *the Services sub-layer,* which provides the functionality that is required.
- *The Interface Layer*, which manages the access to the system by any kind of client.

Next we explain in detail the last two layers, since the first one has to be manually implemented to allow the technology independence.

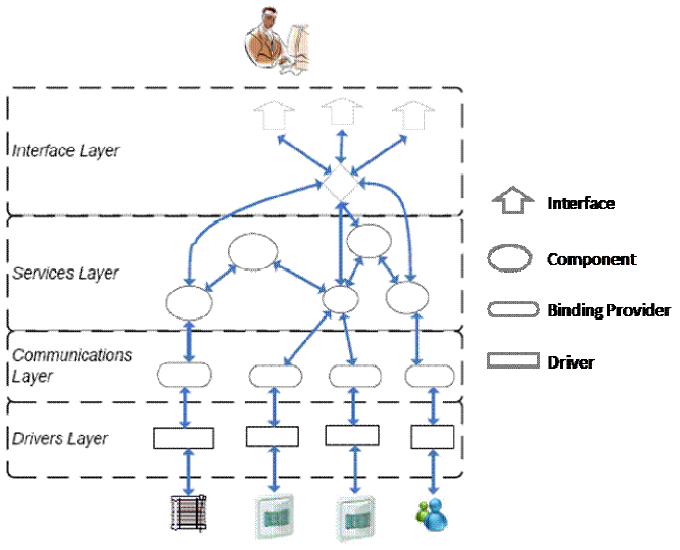


Fig. 10 Architecture of the Implementation Framework

4.2.1.1 Logical Layer Implementation

The logical layer provides the set of classes that facilitates the generation of the logic of the system. To do this, these classes provide us with constructors that are similar to the PervML concepts such as service, binding provider, component, user, etc. These classes are extended in order to generate the final system. These classes define the attributes that must take value when the framework is instantiated and implement the execution strategies of each element using the Template Method pattern [48].

Fig. 11 shows a partial view of the classes of the logical layer as well as the different relationships among them. Classes in this layer can be classified in three functional groups:

- Classes for mapping the PervML conceptual primitives. This functional group is composed of five classes. The *PervMLInteraction, PervMLService,* and *BProvider* classes provide support to the C*ommunication* and *Services sub-layers*. For instance, they provide support to implement the different binding providers that communicate the system with the physical devices.
- Classes for encapsulating OSGi-related functionality. The goal of this group is to isolate some OSGi-related functionality that is inherited by the classes in the previous functional group. Classes in this group provide facilities for logging events (*Logger*), for search services in the OSGi framework (*FrameworkSearcher*) and for participating in the event notification mechanism supplied by OSGi (*WireParticipant*).
- Classes for dealing with the system life-cycle. This functional group is composed of the *InteractionActivator, ServiceActivator* and *BProviderActivator* classes. An activator is an OSGi concept which describes the class that is in charge of registering and unregistering the services in the OSGi framework when a bundle is started or stopped. In our case, the mechanisms for notifying and receiving notification of changes in the OSGi services (Wires in OSGi terminology) are created, too. Most of the functionality supplied by the activators is shared by five elements, so an abstract class (FrameworkActivator) has been implemented.

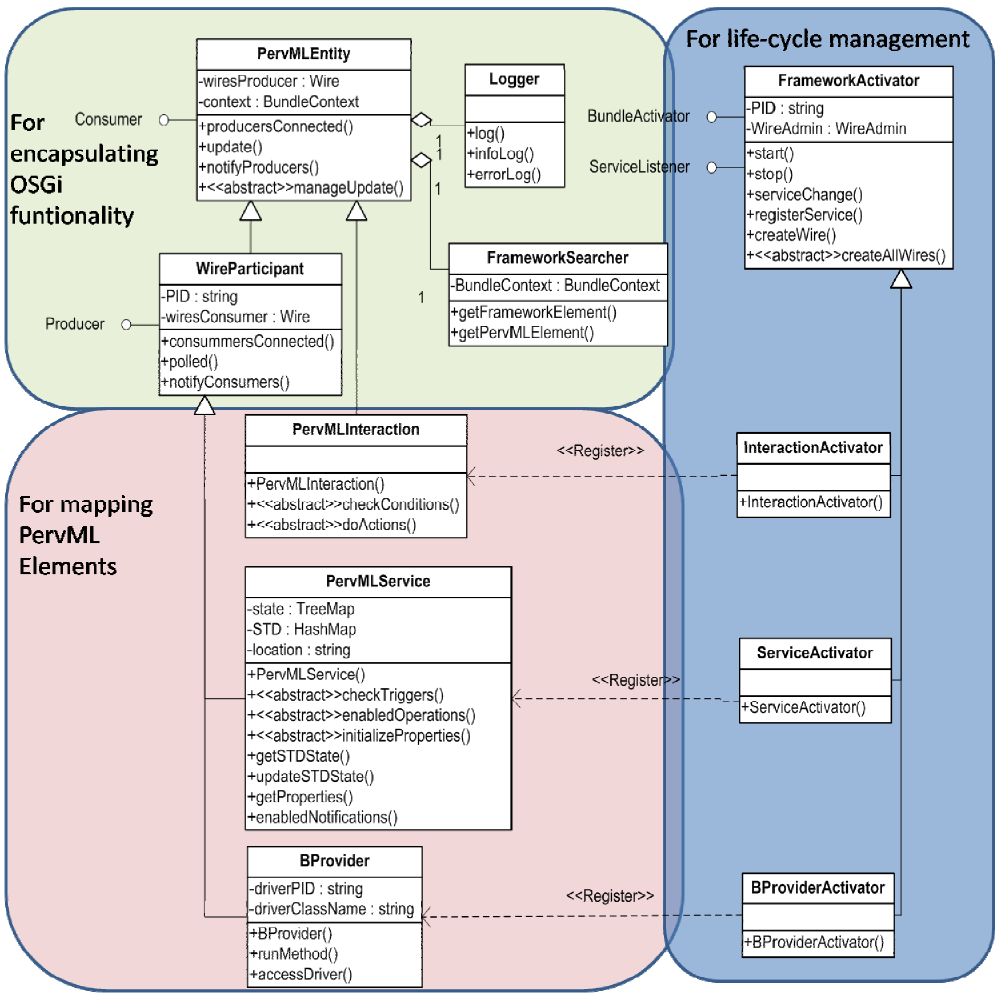


Fig. 11 Design classes for the system logic layer of the framework

4.2.1.2 Interface Layer Implementation

The Interface Layer gives support to the presentation of information and services to users. In order to implement this layer, we have used the Model-View-Controller (MVC) pattern [48], which provides us with support for having different interfaces to interact with the system.

By following this strategy, the components of the *Services Layer* correspond to the *Model*; the *Controller* class, which is reusable for every interface, has to be implemented; and for each supported *Interface*, specific views have to be implemented. Thus, as well as implementing the *Controller* class, we have implemented some views, for example, the corresponding to a Web Interface for access from desktop web browsers. Fig. 12 shows the *Controller* class and the classes that implement this Web interface.

The *Controller* class provides methods that allow users: (1) To select the component that is the specific service with which the user wants to interact. For instance, the method *getServicesFromLocation (String location)* returns all the services that are available for a user in a specific location. (2) To interact (get information and request functionality) with a single component. For instance, *ManageOperation(String ComponentPID, String Operation, Object []parameters)* is in charge of executing a Component operation using Java reflection capacities.

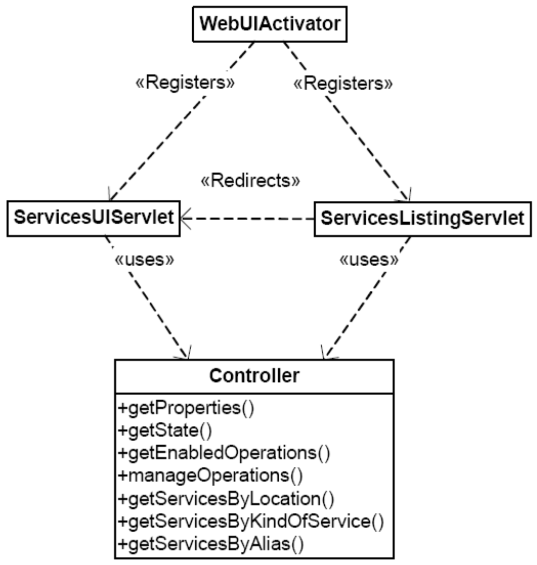


Fig. 12 Web Interface

Fig. 12 also shows the *ServicesListingServlet* and the *ServicesUIServlet* classes that implement Java Servlets. These classes provide the view of the web interface and invoke the Controller operations in order to generate the web pages that will be shown to users. The *ServicesListingServlet* class invokes the first set of methods, whereas the *ServicesUIServlet* class invokes the second set of methods. It is worth to noting that every class in Figure 12 is a concrete class. This means that classes do not need to be extended. In addition, these elements are reusable for all the pervasive systems that are developed using the proposed approach. This feature is feasible since (1) every *Service* implements an interface that is known by the Controller and (2) the Java reflection capabilities have been used to invoke previously unknown methods.

4.2.2 From PervML to Java Code

In this section, the transformation to obtain the system java code is explained. The input of the transformation must be a PervML model, which is composed by the models described in Section 4.1 (each of them describes a view of the system), and the output must be OSGi bundles in source code format (Java code and Manifest files), which extend the framework code and follow its guidelines. Although the input of the transformation is the PervML model (which is composed by both the models from the Analyst view and the models from the Architect view), we only focus on the models from the Analyst view because the context information is described in these models. First we describe the mappings between PervML and the Java/OSGi code that must be generated, in an intuitive way. Then we explain how to automate these mappings, which are done in the same way for each model of PervML.

4.2.2.1 Mappings

The mappings between the PervML models and the code that had to be generated from the models were described in an intuitive way by defining the outputs produced from each PervML model. These outputs are the followings:

- **From the Services Model:** Every type of service produces an abstract class that implements methods that contain the information that is specified for that type; for instance, methods for checking the pre and post conditions of every type of service operation. This abstract class extends the *PervMLService* class of the framework). The state machine is implemented as a set of linked classes for dealing with every state and every transition. Triggers are also implemented in specific classes with methods for checking the triggering condition and executing the actions.
- **From the Structural Model:** Every component from the component diagram produces a concrete class that extends an abstract class that was produced from the Services Model (depending on the type of service that the component implements). This class is in charge of implementing the operations that were defined in the Services Model. It maintains links with the components that it uses, and it provides mechanisms for reacting to changes.
- **From the Interaction Model:** Every interaction produces a concrete class that extends the abstract *PervMLInteraction* class of the framework. This concrete class

implements methods to check the triggering condition and to execute the actions that are specified in the sequence diagram.

- **From the Binding Providers Model:** Every kind of binding provider produces a concrete class in the Communications Layer that is in charge of disseminating to upper layers the drivers notifications and requesting the drivers the execution of operations that are required by the components. During the initialization, the system creates as much instances of these classes as binding providers are in the system of every type. Some configuration is required to set-up the link between the binding provider and its corresponding driver (usually it consists on specifying the driver identifier).
- **From the Component Structural Specification:** The information in this model is used to establish the links between the classes that implement the components in the Services Layer and the classes that implement the binding providers in the Communication Layer. This link is defined, by one side, as a listening subscription from the component to binding provider's modifications.
- **From the Component Functional Specification:** The information in this model is used to fill the methods of the classes that implement the components.

4.2.2.2 Model-to-code Transformation

A model-to-code transformation was carried out in order to automate the presented mappings. To do this, a set of rules was implemented by using the MOFScript tool. Using MOFScript, the mappings described above by means of model-to-code rules that generate Java code from the PervML models were implemented. Figure 13 partially shows an example of these rule. This rule generates the Java-OSGi code of a Component.

```
import "SharedRules.m2t"

texttransformation Component(in pervml:
"http:///org/oomethod/pervml.ecore") {
pervml.Component::generateComponent(){
var idCounter:Integer = 1

<%package org.pervml.application.components.%>
self.alias <%;import org.osgi.framework.BundleContext;%>
self.genetareDependencyImports()

<%public class Component extends org.pervml.application.services.%>
self.serviceProvided.name<%.GenericComponent {
public Component(BundleContext c, String componentPid){
super(c,componentPid);%>

self.ComponentTrigger->forEach(trig:pervml.Trigger) {
     …
}
<%}
public void initializeProperties(){
this.serviceName="%>
self.alias<%";
this.location="%>self.getLocation() <%";
}%> //End initializeProperties()

self.ComponentFunctionalSpecification.Method
->forEach(me:pervml.Method) {
print("\t protected ")
me.ServiceOperation.generateOperation("Implementation_")println("{")
…
}
<%}%> //End class Component
}//End generateComponent()
}
```

Fig. 13 A Mofscript rule

The final code generated by these rules is java files, which are based on the OSGi platform and a set of manifest files.

4.2.3 Tool Support

Model driven methods must be supported by tools in order to be applicable in an effective way. Thus, the PervML Generative Tool (PervGT) was developed for giving support to the method presented in this section. The tool allows pervasive systems developers the creation of graphical diagrams and the automatic translation of these diagrams into the final implementation code using a transformation engine.

PervGT is based on the Eclipse platform .Three plug-ins have been mainly used to develop it: the Eclipse Modelling Framework (EMF) plug-in, the Graphical Modelling Framework (GMF) and the Mofscript tool.

By using these plug-ins, PervGT gives support to the most relevant aspects of a model-driven generative tool:

- Model management: EMF provides us from the PervML metamodel with: 1) A set of Java classes representing each one of the PervML metamodel concepts; these Java classes provide methods to modify PervML models according to the PervML metamodel. 2) A basic tree editor that facilitates the development of PervML models according to the metamodel.

  Thus, PervGT allows us manipulated (create / edit / save / load) the PervML models conforming to the PervML metamodel. Furthermore, models should be stored according to any standard in order to improve the interoperability with others tools.
- Graphical model edition: GMF provides an editor to specify graphical editors in a declarative way, and also provides a runtime where common functionality related to

graphical editors is already implemented, like model printing or automatic layout algorithms.

Thus, PervGT provides a graphical editor for each PervML model. For instance, Fig. 14 shows the graphical editor for the state transition diagram of the Service Model.

- Code generation: By using the rules implemented with the Mofscript tool (see the previous section), PervGT allows us to generate from the PervML models, the Java-OSGi code that constitute the source code of the pervasive system specified in the PervML models.

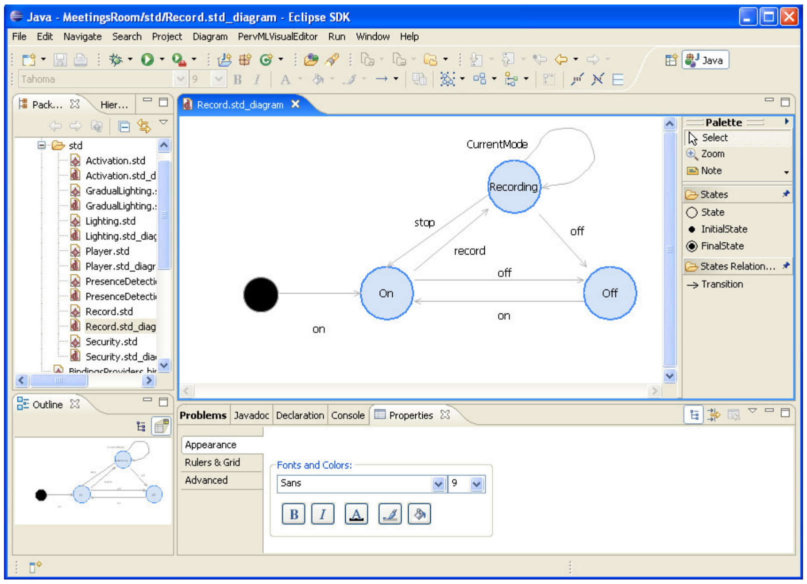


Fig. 14 The graphical editor for the State Transition Diagram

4.3 Conclusions

In this chapter we have explained the MDD method with which the approach propose in this master thesis has been integrated. To understand this integration, we have explained the two big contributions that this method provides us: a modelling language to capture the requirements of a pervasive system and a code generation strategy that allows us to automatically generate a functional pervasive system in Java code. This last contribution is one of the main reasons to carry out this integration. The other important reason is the extensive knowledge that the research group in which this master thesis has been developed has about this method.

5. Supporting Context in Pervasive Systems

In this chapter, we propose an infrastructure to properly capture, manage and understand context in MDD environments. As we have explained in Chapter 4 we have integrated our approach with the MDD method presented in this chapter. MDD is an approach to software design and development that strongly focuses on models. Thus, in order to support context in the MDD method we propose a set of models to properly capture context at modelling time. These models are explained in detail in Section 5.1.

However, it is not enough to properly supporting context because context is continuously changing at runtime. Therefore, we also propose an OWL context repository, which is based on a context ontology, for persistently storing context in a machine interpretable language. This is explained in detail in Section 5.2. Furthermore, we propose a framework that contains the necessary mechanisms to manage context at runtime and interpret it at a semantic level by using this repository. This is explained in detail in Section 5.3. We also integrate all of these contributions with the MDD method presented in the previous chapter; however, they can be easily used in other approaches as it will be explained further.

5.1 Context Modelling

In this section, we present a set of models to capture context at modelling time. The information that must be captured by these models is the following: information about users, privacy and security policies, space information, system services and system devices.

These models have been defined with the purpose of incorporating Context support into the PervML conceptual schema. PervML (see Chapter 4) only captures the system services with the location name where they are provided (in the models of the Pervasive Analyst view) and the system devices (in the models of the Pervasive Architect view). PervML does not allow us either capture information about space information, or system users, or privacy and security policies. For this reason, we have added in the Structural Model a location diagram, which properly specifies the locations where users can be or where services can be deployed; and we have proposed a user model that is composed by a Policy diagram, which specifies the security policies that limit the services to which system users will have access, and a set of User Characterization Templates, which specify the system users. We have also improved the Interaction Model to facilitate its specification. Next, we explain in detail these models.

Thus, Fig. 15 shows the models that the language provides now to specify context-aware pervasive systems.

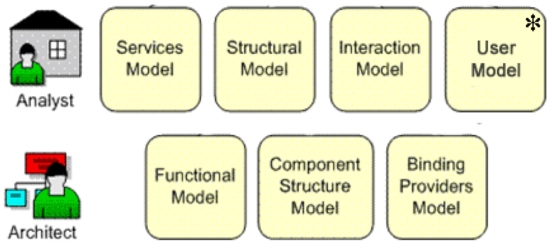


Fig. 15 PervML models (* optional)

5.1.1 The Structural Model

The Structural Model describes the environment where the system will be deployed and the services that this system will provide. To do this, the structural model is defined from two elements:

- **A Location Diagram**, which describes the different areas that constitute the environment where the system will be deployed. These areas represent where the system users can move or where services can be located. This is specified by means of an UML package diagram. Each package represents a certain area, and the hierarchy between packages symbolizes the space hierarchy between the areas represented by those packages. Also, two types of associations can exist between the areas or packages: adjacency and mobility (or accessibility). Adjacency means that the areas are next to each other, whereas mobility is the possibility to go from one area to another. Adjacency is represented by a line between two areas. Since mobility implies adjacency, it is represented by adding arrows to the line between two areas. Thus, this model allows us to infer information such as transitive relations or find out the way to arrive to certain area. For example, if we can go to the corridor from the kitchen and to the bathroom from the corridor, then we can go to the bathroom from the kitchen.

Fig. 16 shows an example of the Location Diagram. It models the locations of the running example. It has seven areas: *Hall, LivingRoom, Kitchen, Corridor, Bathroom, ChildRoom* and *ParentsRoom*. This figure shows that, for instance, the *Hall* and the Bathroom are adjacent, but a user can not go from one to another. In contrast, the *Hall* and the *LivingRoom* are adjacent and a user can also go from one to another.

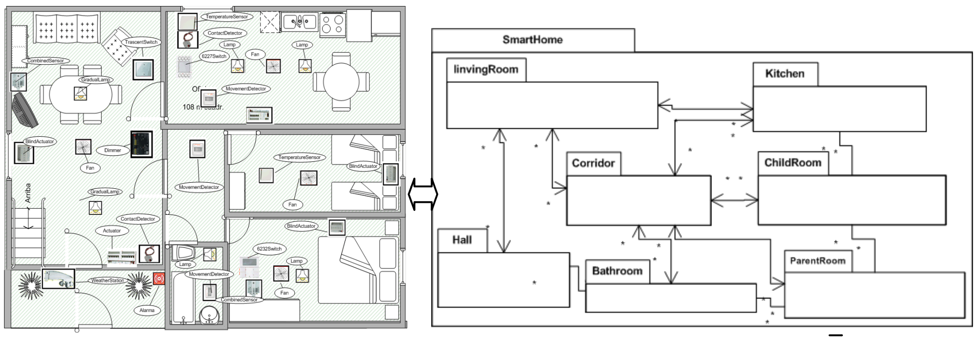


Fig. 16   Location Diagram of the running example

We use a UML package diagram for representing locations because it is very intuitive for the analyst, as Fig. 16 shows, packages can contain other packages like locations can contain other locations. Also, UML components (which represent services as the following diagram shows), are deployed in packages, like the real services, which are provided in locations. This allows us to describe any space easily by simply delimiting areas and relate them intuitively.

- A **UML 2.0 component diagram,** which represents the services that are deployed in each physical location as well as indicating the type of service that is provided by each service. To do this, the services are represented by means of UML 2.0 components and are associated to package that represents the location where they are deployed. The service that provides each component is depicted as a UML interface. Also, relationships between components indicate a relation of use, in other words, the source component uses the functionality of the target component.

Fig. 17 A partial view of the component diagram

Fig. 17 shows a partial view of the Structural Model for the running example. It shows the components associated to the *ParentsRoom*. We have defined several components such as the *ParentsLighting* that provides the *Lighting* service, which is used by the *ParenstLightingControl*; and the *ParentsBlind* that provides the *BlindManagement* service.

5.1.2 The User Model

The User Model is used to specify the security policies of the system and the information related with the system users. It is defined from two elements:

- A **Policy Diagram**, which is used to specify the policies of the system. A policy describes a type of user by defining its allowed operations. Thus, a policy is associated with users (as Fig. 18 shows the *Father* policy is associated with the *Peter* user), which means that these users can perform the operations specified in the policy. Then, in order to associate the allowed operations to each policy, the analyst can: 1) associate a service (then every operation of every component that provide that service will be allowed); 2) associate a component (then every operation of this component will be allowed); 3) associate a service operation (then this operation will be permitted for every component that provides this service); or 4) associate a component operation (then this operation of this component will be permitted).

  Furthermore, inheritance relations can also be established in this model. These relations allow us to define capacities of a policy taking the capacities of a defined policy as a basis. This can be done in two ways: 1) By adding new capacities to capacities of the parent policy. This is used if the child policy can execute more operations than the parent policy. The actions that the child can execute but the parent can not are denoted by "+". 2) By removing capacities of the parent policy. This is used if the child can execute fewer operations than the parent. The actions that the parent can execute but the child can not are denoted by "-".

For instance, in the example of Fig. 18, the parent can execute operations related to the *Lighting*, *GradualLighting* and *BlindManagement* services. The child can execute the same operation except the operations of the *BlindManagement* service.

- A set of **User Characterization Templates**, which specify the users of the system. Pervasive System Analysts must indicate the following information for each user:
    - The policy associated to the user, which has been previously specified in the Policy Diagram.
    - The following personal data: name, surname, gender, date of birth and marital status.
    - The following contact data: email, telephone number, mobile phone and address.
    - Social relations, i.e., information related to people that the user knows.

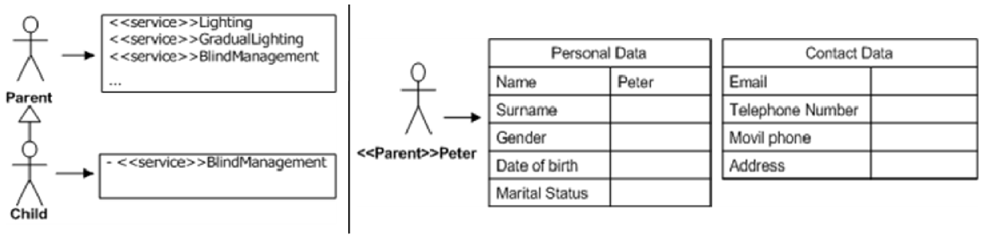


Fig. 18 A Policy diagram on the left and a User Characterization Template on the right

Both Personal and Contact data are those proposed by SOUPA to characterize users. We allow analysts to add new properties if required. Fig. 18 shows an example of User Characterization Templates. This example represents the *Peter* user in the system, whose policy is *Father*.

Finally, note that this model provides support for the privacy, the security and the views of the system, since users will be able to see and execute only system actions that they are authorized to use. On the other side, the construction of this model is optional. The model must be defined only in the case that a personalized system behaviour must be provided. If Pervasive System Analysts do not create this model, three policies are defined (with a default user for each one): (1) the administrator, who can execute all the operations available in the system including configuration operations; (2) the limited user, who is able to execute all the operations available, except configuration operations; and

(3) the guest, who can only consult the state of the system, but not execute any operation that modifies it.

5.1.3 The Interaction Model

This model has been also modified to facilitate its specification and improve its scalability. The Interaction Model is used to describe the communication that is produced as a reaction to a system event. An interaction is a communication between components to provide a specific functionality.

Thus, Systems Analysts may want that the action is ordered to every Component that provides a type of service. In this case, we have allowed that Analysts can indicate that the object that receives the action is a type of service, with the label <<Service>>, instead of having to indicate every component that provided that type. Fig. 19 shows the interaction that is in charge of lowering every blind, winding up every awning and stopping the sprinklers of the garden when it starts to rain. It is also important to note that the actions specified in an interaction are not executed as an explicit user request, but are invoked as a reaction to a situation.

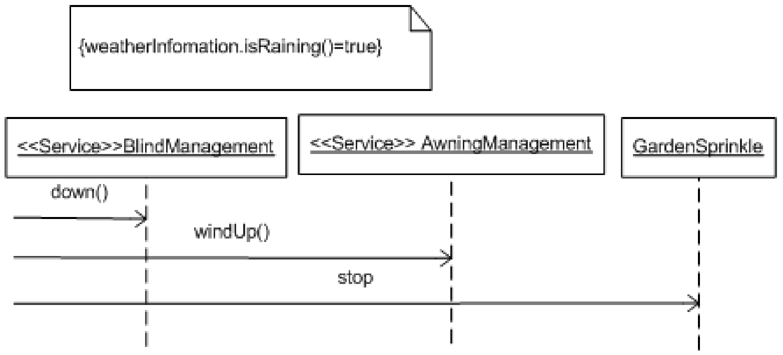

{weatherInfomation.isRaining()=true}
<<Service>>BlindManagement
<<Service>> AwningManagement
GardenSprinkle
down()
windUp()
stop

Fig. 19 An interaction that lowers every blind, winds up every awning and stops the garden sprinklers when it is raining

5.2 The Context Ontology and the OWL Context Repository

Previous chapter explains how Context can be captured at modelling time. However, describing Context information in models is not enough for properly develop a Context-aware system since all this information need to be managed by the system at runtime in order to properly adapt itself accordingly.

In order to store the context information and support its management at runtime we propose an OWL Repository. This repository is based on an ontology that we propose for modelling Context in AmI environments. Whereas the ontology provides concepts and relationships between concepts that allow us to represent the Context at a high level of abstraction, the repository allows us to store, according to this ontology, the specific context information of a pervasive system. In addition, the use of an ontology for representing Context at run time facilitate the understanding of Context information at the semantic level.

5.2.1 The context Ontology

We have defined an ontology [57] to represent the concepts and relationships between concepts that allow us to describe Context at a high level of abstraction. This ontology is based, above all, on the SOUPA ontology [12], where the information considered by different context-aware proposals have been gathered and placed in common. The SOUPA ontology is expressed using the OWL language and includes modular component vocabularies to represent intelligent users with associated beliefs, desires, and intentions, time, space, events, user profiles, actions, and policies for security and privacy.

However, the ontology proposed in our approach differs from others such as SOUPA in the fact that we propose a set of concepts of a higher abstraction level. This aspect facilitates the reasoning about this information at the semantic level and the deduction of new Context data from the existent one. For instance, we propose concepts to define locations such as *Bedroom* or *Kitchen* and propose relationships such as *includes* or *adjacency* to relate this locations (instead of relating locations by means of coordinates such as longitude, latitude and altitude as it is proposed in SOUPA). In addition, we also include other concepts such as Service or Operation in our ontology. These concepts provide the ontology with a greater semantic richness in order to express how the system provides services to users and how users interact with these services (user behaviour). For instance, we can relate services that are available for users with the locations of the AmI environment in which they are provided; we can also relate actions performed by users with the service operations that are activated with these actions. This information helps the system to study the behaviour of users in more detail to properly adapt itself to it.

Next, we explain all these concepts and the relationships among them in detail. Additionally, we show their in Figure 20 using the approach presented by Al-Muhammed et al [49]. We have modified this approach to differentiate between the context information that can be updated at runtime. Thus, two kinds of concepts can be defined: non lexical concepts (enclosed in solid rectangles), that represent the ontology classes; and called lexical concepts (enclosed in solid ellipses), that represent the properties of each class. Figure 20 also shows a set of relationships among concepts, represented by connecting lines, such as *isExecutedIn* from the *Action* class that indicates in which location the action has been executed. The arrow connection represents a one-to-one relationship or many-to-one relationship (the arrow indicates a cardinality of one) and the non-arrow connection represents a many-to-many relationship. On other hand, a small circle near the source or the target of a connection represents an optional relationship.

Thus, Figure 20 shows the principal concepts, properties and relationships of our ontology. The main conceptual primitive of this ontology is the service metaphor. It describes the services of the system for the AmI ecosystem. A service is an entity that provides a coherent set of functionality which is defined in terms of atomic operations. The first concept that we can see is ***Service***, which represents the services that are available in the AmI environment (e.g. Lighting, Video Player, Alarm, etc). A service is characterized by a *name* and a ***ServiceCategory*** *(*e.g. illumination, multimedia, security, etc). Each service category is characterized by a *name* and can belong to another category. This aspect allows us to define hierarchies of categories. Services can be related to each other by means of two relationships: parent and uses. On the one hand, the *parent* relationship allows us to indicate hierarchies of services by defining parent and child services (e.g. we can define the *Gradual Lighting* service as a child of the *Lighting* service). This relationship is use with inheritance purpose in order to indicate that a service presents all the characteristics of another service plus some additional ones. On the other hand, the *use* relationship indicates that a service needs to use another service in order to be properly executed (e.g. the *Alarm* service needs to use a *Presence Detection* service in order to be activated when someone is detected a specific location). The behaviour of each service is characterized by its ***Operations*** which are described with a *name* (e.g. the operations of a service Video Player may be play, stop, forward, review, etc). Finally, each service is located in a specific *Location*.

The Services Operations can be also related by interactions. An ***Interaction*** allows a set of services to communicate among them as reaction to an event. It is describe by means of a condition, which represents the trigger condition of the interaction, and a set of messages that indicate the operations of the services that have to be executed. For instance, an interaction could be the increase in the illumination intensity when it gets dark.

The concept ***Location*** represents the different areas of the AmI environment (e.g. Kitchen, Bedroom, Garden, etc.). It is characterized by a *name*. Locations can be related to each other by means of three types of relationships: made up, adjacency and mobility. The *made up* relationship indicates that a location consists of other locations (e.g. the location First Floor is made up of the locations Kitchen, Hall and Living Room). The *adjacency* relationship indicates that two locations are physically together (e.g. the Parent Bedroom and the Children Bedroom are adjacent). The *mobility* relationship indicates the same as the adjacency relationship plus the fact that there is a way for persons to go from one location to the other (e.g. the Hall and the Living Room are adjacent and the Hall has a door to go to the Living Room).

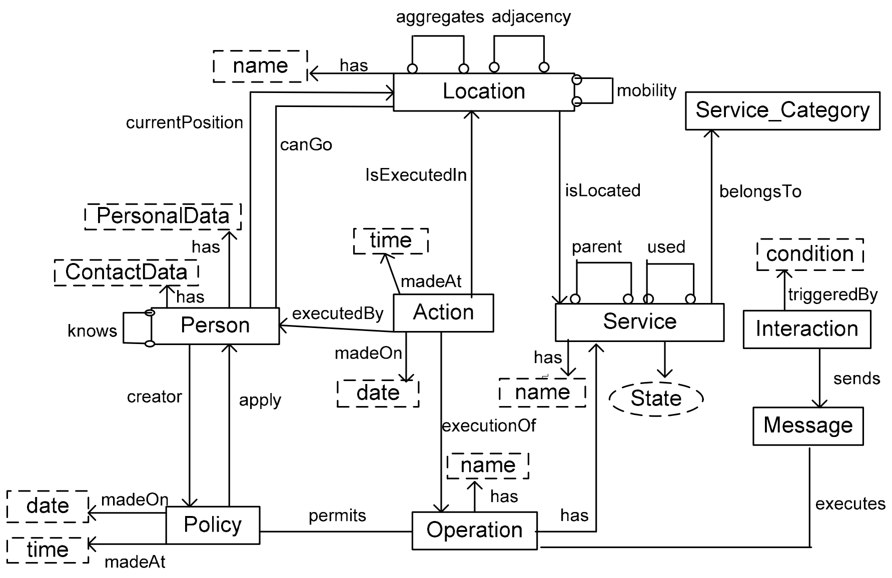


Fig. 20 A partial view of the Context ontology for AmI systems

Another important concept in the ontology is ***Person***, which represents the users of the AmI system. Each person is characterized by some personal data such as the *name* and some contact data such as the *email*. More lexical concepts are associated to Person in order to properly define the personal and the contact data. However, they have been omitted to not overload Figure 20. Each person performs ***Actions*** that are characterized by the *time* and the *date* in which they are performed. The fact that a person performs an action implies that an operation of a service is activated as a consequence of the action. Thus, Actions are related to Operations as well as to the location where the action is performed. Furthermore, persons may be related to each other by means of the relationship *knowns*. This relationship allows us to indicate which the friends of a specific person are. With regard to the location in which persons are, it can be described by means of the relationship *currentPosition*. Another relationship is defined in order to indicate the locations where users can go from its current position (as we will see Section 5.3, this is information derived from the current position of the user and the different mobility relationships defined among locations).

Persons are also associated to ***Policies***. Each Policy restricts or guides the operations that its associated persons can perform (e.g. we can create a policy for children that does not allow them to activate the security service or the heating service). A Policy is characterized by its *creator* (a Person), and its creation *time* and *date*.

To conclude this section, note that the most important contributions of our ontology are both the high level of abstraction that present its concepts and the great semantic richness that the relationships between these concepts have. These aspects facilitate the analysis and reasoning about Context in order to support the adaptation and anticipation aspects. Finally, it is worth noting that this section presents a partial view of the ontology (only the classification scheme). In addition, some concepts such as those that describe characteristics of a Person, or those that describe aspects related to preferences or beliefs have been omitted to not overload the section.

5.2.2 OWL Context Repository

We store all the context information related with a pervasive system in an OWL repository based on our ontology. Thus, we have the system specification in a machine interpretable language what facilitates the automated reasoning about context information. Furthermore, there are several reasoners or inference engine such as

Racer[50] or Pellet[45] that allow us to infer knowledge from OWL. Additionally, the OWL specification provides us with persistence support since OWL allows us to store both information available at modelling time and information available at runtime.

The strategy that we have followed to create an OWL Context repository (see Figure 23) of an AmI system presents two main steps:

1. First, the concepts and the relationships between concepts of the ontology are described by means of OWL classes and properties (we have used the EODM plug-in for this purpose). This description is common to every AmI systems, so it only needs to be created once. After its initial creation, it can be reused in the development of other AmI systems. To create the OWL description of our ontology the following guides are given:
    - Non-lexical concepts of the ontology are defined as OWL classes. For instance, we have created the OWL class Service to represent the non-lexical concept Service.
    - Lexical concepts are defined as properties. The domain of these properties is the class created from the non-lexical concepts to which the lexical concept is associated. The range of these properties is defined from specific DataType. For instance, we have created the OWL property Name to represent the lexical concept name, which appears in the ontology several times. Its domain is defined from all the non-lexical concepts to which it is associated (*Service, ServiceCategory, Person, Location* and *Operation*). Its range is defined from the *string* DataType.
    - Each relationship between Non-Lexical Concepts is defined by means of an object property. The domain and range of this property is defined from the two associated non-lexical concepts. Developers must analyze the relationship semantics to select which concept is the domain, and which is the range. Optionally, if developers consider it opportune, a second object property can be defined as the inverse of the created one (whose domain and range is defined from the same concepts but in an inverse way).

    For instance, the relationship isLocated between the concepts Service and Location is supported by creating the property isLocated whose domain is Service and whose range is Location. This property allows us to indicate the location in which a service is deployed. If we need to indicate also the services that are deployed in a location the inverse property can be created.

2. Second, the concepts of the ontology are instantiated for each specific AmI system. To do this, the corresponding individuals have to be created. For instance, if an AmI system must support a *Lighting* service in the kitchen, the *Lighting* individual (instantiation of the *Service* concept) and the *Kitchen* individual (instantiation of the *Location* concept) have to be created. The relation between these two individuals is defined by creating the *isLocated* property in the *Lighting* individual and making this property refers to the *Kitchen* individual. Note that there two types of individuals that need to be created in the OWL Context repository: (1) those that are available in design time (such as the location of the environment or the services provided by the system) and (2) those that are only available at run time (such as the user actions). The first type of individuals is created by an automatic model-to-model transformation that we explain in next section. How the last type of individuals is created will be explained further.

5.2.2.1 Automatic generation of context data available at design time

In this section, we explain the automatic transformation that we have implemented to generate the individuals that represent the information available at design time in the Context Repository. As we have integrated our approach with the method presented in Chapter 4, the information available at design time is represented in the PervML models. Thus, this transformation takes as input a PervML model (which is composed of the models described in Section 4.1 and Section 5.1) and the OWL repository obtained in the first step (which contains the concepts and the relationships between concepts of the ontology by means of OWL classes and properties); and obtains as output the OWL context repository with all the information contained that was represented in the PervML model of input.

To carry out the transformation, we first describe the mapping between the input and the output in an intuitive way, and then we explain how to automate these mappings.

Mappings

We describe the mappings in an intuitive way defining the outputs produced from each input. On the one hand, the output from the PervML Ontology is one to one, in other words, the output and the input are the same. On the other hand the outputs produced from each PervML element are the followings:

- From every instance of a class C of the PervML metamodel, an individual of the OWL class that represent the PervML metamodel class C is produced. For instance, the individual *Lighting* of the Service OWL class is produced from the service *Lighting* in the PervML Service Model.
- From every instance of a PervML metamodel class attribute A, an instantiation of the corresponding OWL property is obtained. For instance, from the attribute name of the *GradualLighting* service whose value is *GradualLighting*, the datatype property name in the *GradualLighting* individual is instantiated with the value *GradualLighting*.
- From every PervML metamodel relationship between an instance of a class *A* and an instance of a class *B*, the OWL object properties of the individuals that represent these instances are instantiated. This is, the object property defined in the class *A* to represent the relationship with the class *B*, is instantiated with the individual that represents the instance of the class *B*; and the object property defined in the class *B* to represent the relationship with the class *A*, is instantiated with the individual that represents the instance of the class *A*. For instance, from the *child* attribute of the *GradualLighting* service whose value is the *Lighting* service instance*,* the *child* object property of the *GradualLighting* individual is instantiated with the *Lighting* service individual.

Model-to-Model transformation

In order to automate the transformation we have implemented the corresponding rules based on the mappings. To do this, we have used the ATL plug-in and the EODM plug-in (which has been explained in Section 2.5). Figure 21 shows a global vision of the transformation where we can see the input and the output and the metamodels to which they are conformed. Using ATL, we have implemented the mappings described above by means of model-to-model rules that generate the Context Repository with the corresponding OWL individuals (defined according to the OWL metamodel provided by EODM). Examples of these rules are shown in Fig. 22. The first rule is part of the set of rules that copy the input context repository in the output context repository. This rule particularly copies the OWL classes from the input to the output. The second rule is part of the set of rules that transform the information contained in the PervML models into the corresponding individuals in the output context repository. This rule particularly transforms the locations defined in PervML (to be precise in the Location Diagram of the Structural Model) in OWL individuals of the *Location* Class.

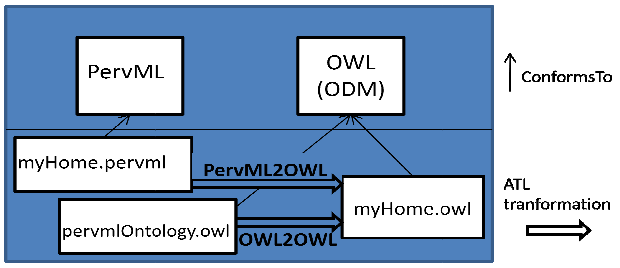


Fig. 21 Transformation to obtain the system OWL specification

Once the PervML models have been specified, we can apply the ATL rules to automatically obtain in OWL the information of the system available at design time. Using the EODM plug-in we can see the result with either the tree editor that provides this plug-in or in OWL source code.

```
1°. From OWL classes to OWL classes (Copy Rule)
    rule OWLClass2OWLClass {
     from  c : OWL!OWLClass
     to    oc : OWL!OWLClass (
               subClassOf <- c.subClassOf,
               uriRef <- c.uriRef,
               label <- c.label,
               namespace <- c.namespace,
               ownedProperty <- c. ownedProperty  )}
2°. From locations defined in PervML to OWL individuals of the Location
class
    rule PervMLLocation2OWLIndividual {
     from  l : PervML!Location
     to    i : OWL!Individual (
               uriRef <- u,
               label <- label,
               type<-OWL!OWLClass.allInstances()->select(c|
                                                c.label.name='Location')),
              label: OWL!PlainLiteral (lexicalForm <- l.name),
              u: OWL!URIReference (fragmentIdentifier <- n, uri <- uri),
              n: OWL!LocalName (name <- l.name),
              uri: OWL!UniformResourceIdentifier (name <- l.name)}
```

Fig. 22 ATL rules

Finally, Figure 23 shows a representative example of the OWL Context repository. This figure shows the OWL description of the non-lexical concepts Service and Location including the lexical concept name and the *isLocated* relationship. Some instantiations of these concepts are also shown.

1. DESCRIPTION OF ONTOLOGY'S CONCEPTS

```
// Non-Lexical Concepts: Location and Service
<owl:Class rdf:about="#Location">
     <rdfs:subClassOf rdf:resource="&owl;Thing"/>
</owl:Class>
<owl:Class rdf:about="#Service">
     <rdfs:subClassOf rdf:resource="&owl;Thing"/>
 </owl:Class>

// Lexical Concept: name
<owl:DatatypeProperty rdf:about="#name">
     <rdfs:domain>
         <owl:Class>
             <owl:unionOf rdf:parseType="Collection">
                 <rdf:Description rdf:about="#Service "/>
                 <rdf:Description rdf:about="#Location"/>
             </owl:unionOf>
         </owl:Class>
     </rdfs:domain>
     <rdfs:range rdf:resource="&xsd;string"/>
 </owl:DatatypeProperty>
// Relationship: isLocated
<owl:ObjectProperty rdf:about="#isLocated">
     <rdfs:domain rdf:resource="#Service"/>
     <rdfs:range rdf:resource="#Location"/>
</owl:ObjectProperty>
```

2. INSTANTIATION OF CONCEPTS

```
<Location rdf:about="#Kitchen">
     <name rdf:datatype="&xsd;string">Kitchen</name>
     <mobility rdf:resource="#Kitchen"/>
     <mobility rdf:resource="#Corridor"/>
     <adjacency rdf:resource="#Corridor"/>
     <adjacency rdf:resource="#ChildrenRoom"/>
</Location>
<Service rdf:ID="Lighting">
    <name rdf:datatype="string">Lighting</name>
    <isLocated rdf:resource="#Kitchen"/>
</Service>
<Person rdf:ID="Peter">
    <name rdf:datatype="string">Peter</name>
    <email rdf:datatype="string">pvalderas@dsic.upv.es</email>
    <currentPosition rdf:resource="#Kitchen"/>
    <canGo rdf:resource="#Corridor "/>
    <canGo df:resource="#Kitchen "/>
</Person>
```

Fig. 23 Example of simple OWL description based on our Ontology

5.3 A Framework for Managing the OWL Context Repository

In this section, we introduce a Java framework to manage the OWL Context repository presented in the previous subsection. This framework allows us to at runtime capture and manage the context information that can be directly obtained from the pervasive system (e.g. through sensors). Moreover, it allows us to derive from this directly captured information all the information at semantic level that we required to carry out the system adaptation.

The structure of this framework is basically defined by three main classes (see Figure 24):

- *CommunicationWithOWL*: This class is in charge of supporting generic access to the OWL Context repository. This class provides two methods, one for add any individual to the OWL Context repository at runtime and other for delete any individual of the OWL Context repository at runtime.
- *Reasoning:* This class is in charge of reasoning about Context information. To do this, it retrieves the Context information and the SWRL rules and uses an OWL reasoner to analyze them and derive additional one. If new information is derived, this class adds this to the Context Repository.
- *PervMLService:* This class has been explained in Section 4.1, but has been modified in order to also capture the context information that is only available at runtime. To do this, when an action is successfully executed, this class creates the corresponding individual of the *Action* class and adds it to the context repository by using the *CommunicationWithOWL* class. In addition, this class updates the derived information (user mobility, services state, etc.) from this action by using the *Reasoning* class.

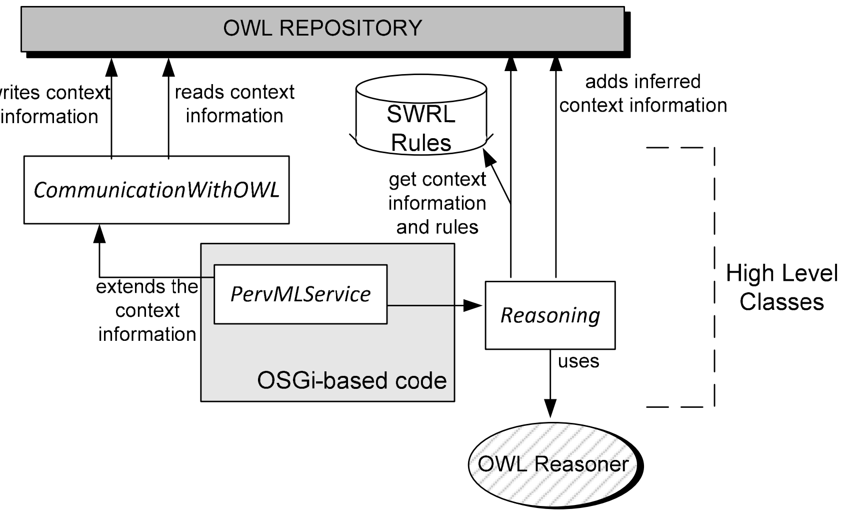

OWL REPOSITORY
writes context information
reads context information
SWRL Rules
adds inferred context information
CommunicationWithOWL
get context information and rules
extends the context information
PervMLService
Reasoning
High Level Classes
OSGi-based code
uses
OWL Reasoner

Fig. 24 General schema of the Java Framework Classes

In order to better understand the classes of the framework we explain how we use them in the following subsections.

5.3.1 Generic management of the OWL Repository

The fact that context continuously changes creates the necessity of updating the Context repository or accessing to it at runtime. It implies to add, modify, delete, or access to individuals in the OWL Context repository. To do this, we have created the *CommunicationWithOWL* class that encapsulate these operations. Therefore, we can change the technology in which the repository is implemented by only modifying this class.

Thus, the *CommunicationWithOWL* class presents four methods that generalize these operations in order to be able to be used for every OWL individual: *addIndividual, modifyIndividual, deleteIndividual* and *getIndividual*. These methods have generic parameters that every individual has. The first two methods have as parameters: the class of the individual, the name that identify the individual, the set of its attributes or properties and the set of its relationships; and the last two methods have parameters: the class of the individual and the name that identify the individual

Implementation Aspects. In order to implement the generic mechanisms that allow the CommunicationWithOWL class to manipulate the OWL Context repository we have used the OWL API 2.1.1 [51]. This API is an open source project that is available as a set of JAR packages. These packages can be imported in any software implemented in Java. Figure 25 shows an example of how this API is used. It shows part of the code that implements the method *addIndividual*. This code does the following:

1. First, it loads the OWL Context repository through the *OWLManager* class and defines an *OWLDataFactory* object for creating OWL elements. These classes belong to the OWL API.
2. Next, it creates the new individual assertion axiom of the corresponding class and adds it to our ontology. As we can see in Figure 25, the identifier of the class and the individual are passed as parameters.
3. Next, it creates the different properties of the newly created individual. To do this it:

   a. gets the properties of the individual passed as parameter (the name of this parameter is *attributes*);
   b. forms the data properties of the individual;
   c. and adds the axioms to our ontology.
4. Next, it creates the relationships of the newly created individual. To do this, it:
   a. gets the relationships of the individual passed as parameter (the name of this parameter is *relations*);
   b. form the object properties of the new individual;
   c. and adds the axioms to our ontology.
5. Finally, both the individual and its properties are saved in the OWL Context repository.

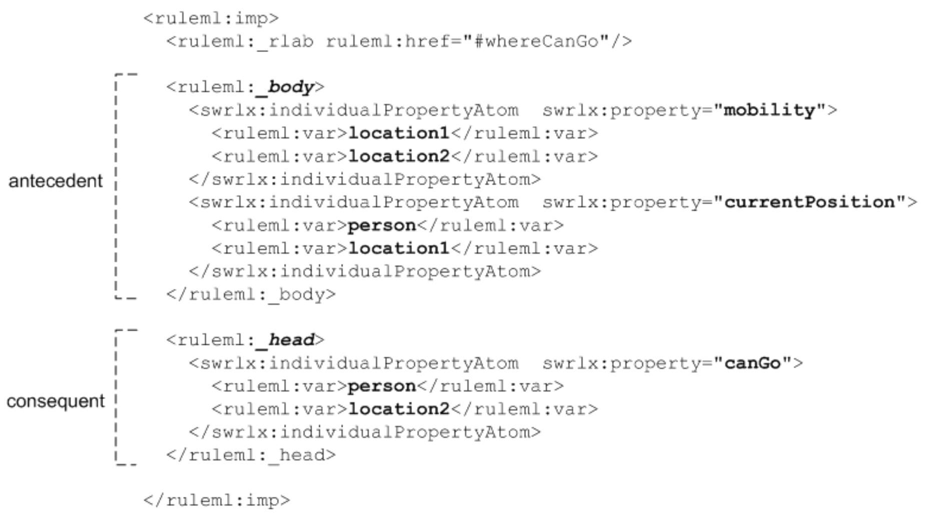
```
    public static void addIndividual(String class_, String ind_, HashMap attributes_, HashMap<String,
                                     List> relations_){
    String uri="http://www.oomethod.com/pervml.owl#";
    String absolutePath="file:/C:/ontologyRepository.owl";

    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
1   OWLDataFactory factory = manager.getOWLDataFactory();
    try{  // read the ontology
          OWLOntology ontology = manager.loadOntology( URI.create(absolutePath) );
          // load the ontology to the reasoner
          Reasoner reasoner = new Reasoner( manager );
          reasoner.setOntology( ontology );

          //We create the new individual assertion axiom, add the axiom to our ontology and save
          OWLIndividual newInd=factory.getOWLIndividual(URI.create(uri+ind_));
2         OWLDescription descripNewInd=factory.getOWLClass(URI.create(uri+class_));
          OWLClassAssertionAxiom  assertion = factory.getOWLClassAssertionAxiom(newInd, descripNewInd);
          AddAxiom addAxiomChange = new AddAxiom(ontology, assertion);
          manager.applyChange(addAxiomChange);
  //We recuperate the attributes, form the data properties of the individual and add the axiom to our ontology
          Set attributes_set=attributes_.keySet();
          Iterator it_at=attributes_set.iterator();
          Tupla tupla;
          String key_data;
          while(it_at.hasNext()){
            key_data=(String)it_at.next();
            tupla=(Tupla)attributes_.get(key_data);
3           if(tupla !=null && tupla.type !=null && tupla.value!=null){
                 OWLDataProperty dataProperty= factory.getOWLDataProperty(URI.create(uri+key_data));
                 OWLDataType ent= factory.getOWLDataType(URI.create(tupla.type));
                 OWLConstant cons=factory.getOWLTypedConstant((String)tupla.value, ent);
                 OWLDataPropertyAssertionAxiom  assertion_data=
                        factory.getOWLDataPropertyAssertionAxiom(newInd, dataProperty, cons);
                 AddAxiom addAxiomData = new AddAxiom(ontology, assertion_data);
                 manager.applyChange(addAxiomData);}
            }
   //We recuperate the relations of the individual, form the object properties of the new individual and add
   //the axioms to our ontology
          Set relaciones=relations_.keySet();
          Iterator it_rel=relaciones.iterator();
          List list_individuals;
          AddAxiom addAxiomObject;
          String clave_object;
          while(it_rel.hasNext()){
             clave_object=(String) it_rel.next();
             list_individuals=relations_.get(clave_object);
             OWLObjectProperty objectProperty= factory.getOWLObjectProperty(URI.create(uri+clave_object));
4            //We have to add each one of the individuals that are related with the new individual
             Iterator it_individuals=list_individuals.iterator();
                 String individual_name;
                 OWLObjectPropertyAssertionAxiom  assertionProperty;
                 while(it_individuals.hasNext()){
                   individual_name= (String)it_individuals.next();
                   OWLIndividual relatedIndividual=
                        factory.getOWLIndividual(URI.create(uri+individual_name));
                   assertionProperty= factory.getOWLObjectPropertyAssertionAxiom(newInd,
                                           objectProperty,relatedIndividual );
                   addAxiomObject = new AddAxiom(ontology, assertionProperty);
                   manager.applyChange(addAxiomObject);
                 }
          }
5         RDFXMLOntologyFormat format=new RDFXMLOntologyFormat();
          manager.saveOntology(ontology,format);
          }
    catch(Exception E){System.out.println("Exception: "+E);}
```

Fig. 25 *AddIndividual* method from the *CommunicationWithOWL* class

5.3.2 Reasoning about Context

Reasoning about Context consists in analyzing the repository of Context in order to derive Context from the existing one at semantic level. For instance, given the current position of a user and the different mobility relationships defined for the AmI environment (those that relate locations indicating that persons can pass from one to another), we can derive all the locations to which the user can pass from its current position.

In order to define these derivations, we use SWRL rules (which have been explained in Section 2.4.2). SWRL extends OWL with Horn-like rules. It enables Horn-like rules to be combined with an OWL knowledge base. These rules are of the form of an implication between an antecedent (body) and a consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold (is "true"), then the conditions specified in the consequent must also hold.

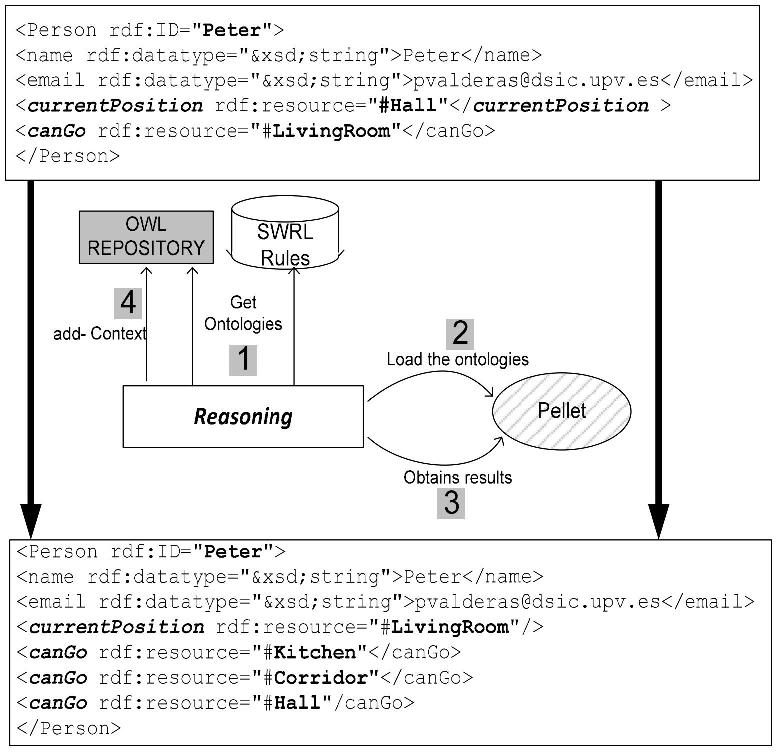


```
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#whereCanGo"/>

  <ruleml:_body>
    <swrlx:individualPropertyAtom  swrlx:property="mobility">
      <ruleml:var>location1</ruleml:var>
      <ruleml:var>location2</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom  swrlx:property="currentPosition">
      <ruleml:var>person</ruleml:var>
      <ruleml:var>location1</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_body>

  <ruleml:_head>
    <swrlx:individualPropertyAtom  swrlx:property="canGo">
      <ruleml:var>person</ruleml:var>
      <ruleml:var>location2</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head>

</ruleml:imp>
```

Fig. 26 Reasoning example by means of SWRL rules

Figure 26 shows an example of SWRL rule. This rule tells the system where a user can go by analyzing its current position and the mobility relationship among locations. To do this, we have defined an antecedent (body) that defines a *mobility* property and a *currentPosition* property. For each property, two variables have been defined. On the one hand, *location1* and *location2* are instantiated with the locations that the mobility property is relating. On the other hand, *person* and *location1* are instantiated with a user and the location where s/he currently is. Note that the current location of the user is instantiated in the same variable that is used to instantiate a location related by the mobility relationship (location1). This means that this antecedent is true only when *location1* is both the location where the user is, and a location that is related to other location. The consequent of this rule indicates that a *canGo* property must exist. This property establishes that the person instantiated by the antecedent in the variable *Person* can go to the location instantiated in the variable *location2*.

Note that this rule is defined by using the high-level concepts defined in our ontology. Thus, we are reasoning about Context in a high level of abstraction, by using notions close to the real environment such as person, location and the possibilities of mobility.



```
<Person rdf:ID="Peter">
<name rdf:datatype="&xsd;string">Peter</name>
<email rdf:datatype="&xsd;string">pvalderas@dsic.upv.es</email>
<currentPosition rdf:resource="#Hall"</currentPosition >
<canGo rdf:resource="#LivingRoom"</canGo>
</Person>
```

```
<Person rdf:ID="Peter">
<name rdf:datatype="&xsd;string">Peter</name>
<email rdf:datatype="&xsd;string">pvalderas@dsic.upv.es</email>
<currentPosition rdf:resource="#LivingRoom"/>
<canGo rdf:resource="#Kitchen"</canGo>
<canGo rdf:resource="#Corridor"</canGo>
<canGo rdf:resource="#Hall"/canGo>
</Person>
```

Fig. 27 Reasoning about Context

In order to include SWRL rules into the AmI system we have created a rule repository (also based on OWL), and we have implemented the *Reasoning* class, which is in charge of applying these rules into the OWL Context repository. To do this, every time that the OWL Context repository is modified, a method from the Reasoning class is invoked. This method receives as parameters the source of the context repository and the rules repository, and performs the following actions (see Figure 27):

1. It create the corresponding ontologies from the source files of the Context repository and the rules repository:
2. It loads in the Reasoner object the ontologies that contains the context information and the SWRL rules.
3. It obtains the inferred information by the reasoner when the Context information is analyzed by considering the SRWL rules.
4. It adds the inferred information to the ontology. For instance, Figure 27 shows an example where a relationship (canGo) is updated by means of the SWRL rule presented above. According to this figure, *Peter* is initially in the *Hall* and then the *canGo* property reference to the location *LivingRoom.* When *Peter* goes to the *LivingRoom* and SWRL rules are applied to the new Context information, the *canGo* property references to the locations *Hall, Kitchen* and *Corridor.*

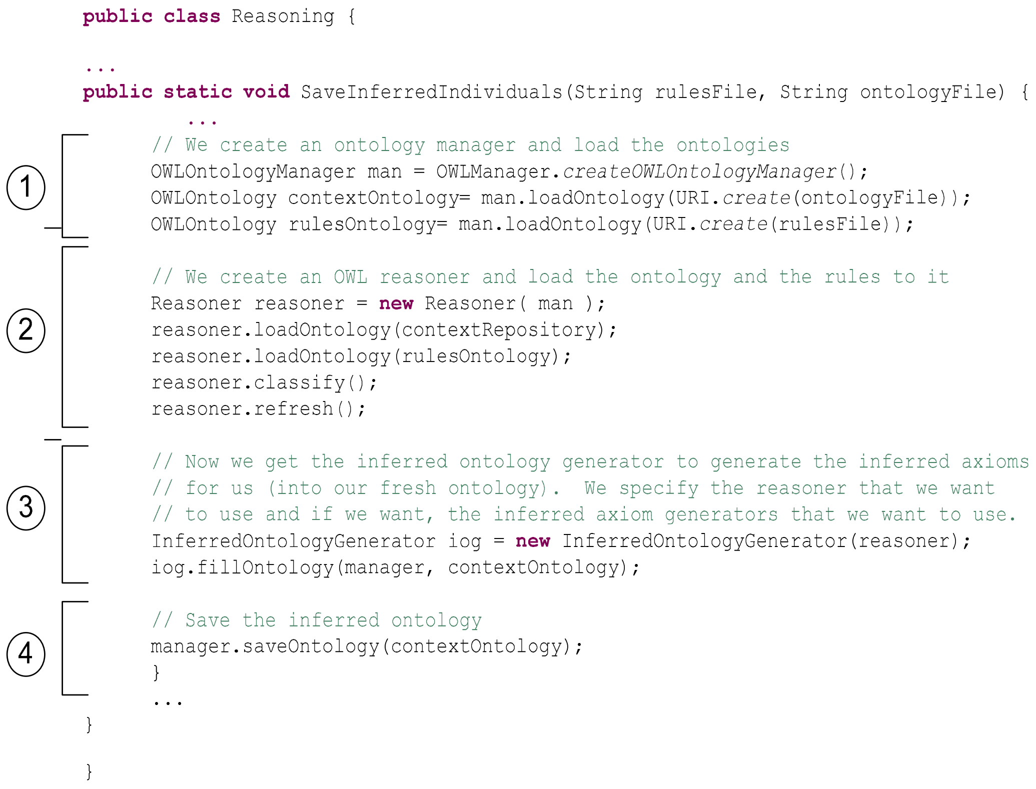
Finally, note that the Reasoning class provides us with:

- Independency from the used reasoner. If we want to use a reasoner different from Pellet, modification efforts are all focused on updating the Reasoning class. The rest of the system does not need to be modified.
- Flexibility for extending the system with new rules, even when the system is deployed and being executed. To do this, we just need to include the new rules into the rule repository. The next time that rules will be recovered from the repository in order to be applied the new rules will be considered.
- Flexibility for doing an exhaustive processing of the inferred information before it is added to the repository. This processing would be done in the Reasoning class before adding the inferred axioms.

Implementation Aspects. As we have introduced above, the reasoning about Context is supported by the reasoner Pellet 1.5.2 and the OWL API 2.1.1. Pellet is an open source OWL reasoner in Java that is provided as a set of JAR packages. The use of this reasoner is controlled by the *Reasoning* class. Thus, this class import the Pellet packages and use the provided OWL API in order to communicate with Pellet. Figure 28 shows part of the code that infers the corresponding context information:

1. We create an *OWLManager* object in order to create the OWL Context repository and the SWRL rules repository ontologies from their source files.
2. We create an object of the Pellet *OWLReasoner* and load the OWL Context repository ontology together with the SWRL rules ontology into it.

3. We get the inferred information about individuals after the application of the SWRL rules by using the reasoner.
4. We update the OWL Context repository with the values obtained from the reasoner.



```
public class Reasoning {

...
public static void SaveInferredIndividuals(String rulesFile, String ontologyFile) {
         ...
      // We create an ontology manager and load the ontologies
      OWLOntologyManager man = OWLManager.createOWLOntologyManager();
      OWLOntology contextOntology= man.loadOntology(URI.create(ontologyFile));
      OWLOntology rulesOntology= man.loadOntology(URI.create(rulesFile));

      // We create an OWL reasoner and load the ontology and the rules to it
      Reasoner reasoner = new Reasoner( man );
      reasoner.loadOntology(contextRepository);
      reasoner.loadOntology(rulesOntology);
      reasoner.classify();
      reasoner.refresh();

      // Now we get the inferred ontology generator to generate the inferred axioms
      // for us (into our fresh ontology).  We specify the reasoner that we want
      // to use and if we want, the inferred axiom generators that we want to use.
      InferredOntologyGenerator iog = new InferredOntologyGenerator(reasoner);
      iog.fillOntology(manager, contextOntology);

      // Save the inferred ontology
      manager.saveOntology(contextOntology);
      }
      ...
}

}
```

Fig. 28 Example of use of the Pellet Reasoner

5.3.3 Capturing Context at Runtime

To deal with capturing all the changes in the context information produced at runtime, our approach registers the information about every interaction in the OWL specification, since every change in the context information is produced by an interaction. This makes the approach is flexible and powerful enough to facilitate the management and the processing of the context information. So, an interaction with the system can be due to:

1. A change in the environment. This interaction is detected by the sensors of the system. In this case, it is not the user who executes the operation, but the environment. For instance, when a user goes into the parents' room, a state change of the presence detection sensor of the parents' room is produced. As a

consequence, the driver that controls this sensor informs to *PresenceDetection* service that uses it and then, this service checks its state by executing the corresponding operation (in this case, the presenceDetected operation).

2. An explicit request by a user. This interaction is produced when a user executes a specific operation of a service (for example, the *open* operation of the *Blind* service of the parents' room) through the user interfaces that the system provides.

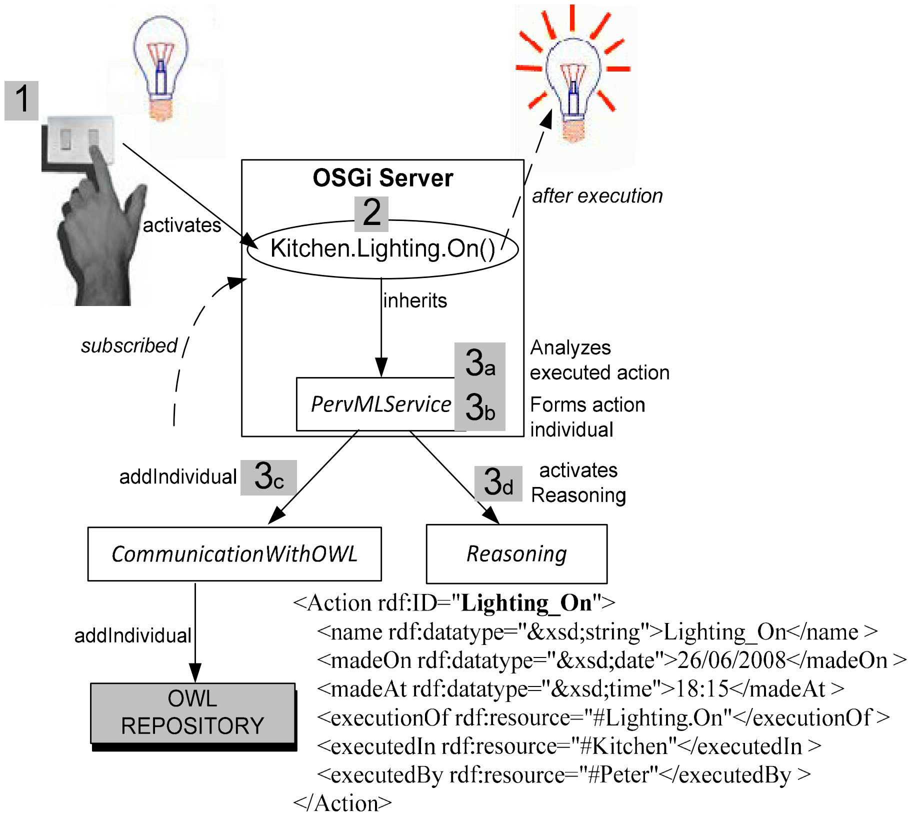
An interaction in terms of the ontology is an action; thus, when an interaction is produced, the corresponding instance of the Action Class is created and is added to the Context Repository. To do this, we have used the *addIndividual* method of the *CommunicationWithOWL* class.

Finally, when an action is added, we use the *Reasoning* class in order to derive information from the executed actions, such as user mobility (from the presence detector services and the identification services) or service state (from state machine of the system services and executed actions). To allow this derivation of information we have added to the SWRL rules repository the corresponding SWRL rules. Thus, we can deduce every information that we need at semantic level, since every context information that we can directly obtain from the pervasive system is stored in the repository; we only need to add the appropriate rules to deduce the corresponding information.

As a representative example, Figure 29 illustrates how user actions are stored in the OWL Context repository. In particular, this figure shows how the service Lighting located in the Kitchen is activated, and how the corresponding user action is registered in the OWL Context repository. According to this figure:

1. First, a user activates the switch of the kitchen in order to turn lights on.
2. This fact activates the *On* operation of the *Lighting* service, which is implemented in an OSGi server. This operation executes code that switches the lamps of the kitchen on.
3. Next, for inheritance from the *PervMLService* class*,* the *Lighting* service checks if the operation has been successfully executed and if so the service:
   a. analyzes who and when the action has been executed;
   b. creates the corresponding individual;
   c. adds this action to the context repository by using the *CommunicationWithOWL* class;

d. finally, it an object of the Reasoning class in order to derive relevant context information from the executed actions.



```
<Action rdf:ID="Lighting_On">
   <name rdf:datatype="&xsd;string">Lighting_On</name >
   <madeOn rdf:datatype="&xsd;date">26/06/2008</madeOn >
   <madeAt rdf:datatype="&xsd;time">18:15</madeAt >
   <executionOf rdf:resource="#Lighting.On"</executionOf >
   <executedIn rdf:resource="#Kitchen"</executedIn >
   <executedBy rdf:resource="#Peter"</executedBy >
</Action>
```

Fig. 29 Extending the OWL Context Repository at runtime

In order to integrate our approach with the method presented in Chapter 4, the step 3 is carried out in the *PervMLService* class (see Section 4.2). If we wanted to integrate our approach with other method, we would only have to carried out this step in the corresponding class.

Implementation Aspects. In order to capture every operation that is executed in the system, we have used the *PervMLService* class of the implementation framework explained in Section 4.2. This class is an abstract class from which every service inherits. If a change is occurred in any service, the *manageUpdate* method of the *PervMLService* is executed. This method checks if the service state has change, and if so, it realizes a set of actions. The last action of this set of actions is the invocation of the method *captureContextInformation* that we have implemented in order to add the corresponding action to the OWL Context Repository. This method performs the following actions (see Figure 30):

1. First, it gets the system user that has executed the action and the current date and time.
2. Next, it forms the parameters that the *addIndividual* method of the *CommunicationWithOWL* class needs to add the corresponding individual to the context repository. To do this, it creates the different properties of the newly created individual. In this example, we only show the creation of the property that indicates the user that has performed the action. The rest of properties are analogously created.
3. Next, the individual with all its properties are saved in the OWL Context repository by using the *addIndividual* method of the *CommunicationWithOWL* class.
4. Finally, the method invokes the *saveInferredIndividuals* of the *Reasoning* class in order to infer the corresponding new context information from the new added action.

```
public void captureContextInformation{

        String user=getUser();
 1      String time=getTime();
        String date=getDate();

        //Forming the parameters for adding the individual
        String class="Action";
        String ind=service_PID+executedOperation;

        //Properties
        HashMap <String,Tupla> attributes = new HashMap<String,Tupla>();
        List <Tupla> type_value = new ArrayList();

        Tupla name=new Tupla("http://www.w3.org/2001/XMLSchema#string", ind);
        type_value.add(name);
        attributes.put("name", name);
 2
        Tupla madeOn=new Tupla("http://www.w3.org/2001/XMLSchema#date", date));
        type_value.add(madeOn);
        attributes.put("madeOn", madeOn);
        ...
        //Relations
        HashMap<String, List> relations= new HashMap<String, List>();
        List<String> users= new ArrayList<String>();
        users.add(user);
        relations.put("executedBy", users);
        ...
        //We add the individual to the Context Repository
 3      CommunicationWithOWL.addIndividual(class, ind, attributes, relations);

 4      //We add the inferred individuals to the Context Repository
        Reasoning.saveInferredIndividuals(contextRepositoryFile, rulesRepositoryFile);

}
```

Fig. 30 Code for extending the owl context repository

5.4 Conclusions

We have described in this Chapter a set of models to properly capture context at modelling time. We have also defined a context ontology and an OWL context repository, which is based on this ontology, for persistently storing context in a machine interpretable language. Furthermore, we have proposed a framework that contains the necessary mechanisms to manage context at runtime and interpret it at a semantic level by using this repository. Finally, we have integrated all of these contributions with the MDD method presented in the previous Chapter and explained how they could be easily integrated with other similar approaches.

6. User Support: Privacy and Adaptation

We have presented in the previous chapter an infrastructure to properly capture, manage and understand context in order to systems possess sufficient intelligence and knowledge-awareness to react appropriately according to context. However, a context-aware pervasive system not only has to know its context, but also it must react and act according to it in order to increase usability and effectiveness of the services that the system provides. To achieve this we focus on two requirements that every context-aware system must fulfil [12, 52, 7]: ensuring the privacy and security of the system and adapting the system to user behaviour by automatically performing user actions when needed without explicit user intervention.

6.1 Privacy and Security of the system

Context information, by its nature, can contain very private and personal data, what introduces privacy and security concerns that must be faced by every context-aware system by means of establishing and enforcing user defined policies.

Both contact and personal user information, and security policies of the system are specified at design time by using the User Model (see Chapter 5). However, this information has to be usually modified after the system is put into operation by the system end-users themselves. Thus, we must also provide tools to update this information but ensuring at the same time the privacy of this information.

There are numerous the changes that this information can have: the number of users can increase or decrease; the information of a user can change or a user can want complete or modified his/her information; if some services are added to the system, users can want create or modified some policies; etc. In addition, it would be very inefficient and annoying having to inform the analyst and the architects of the system in order to specify and generate again the code of the system, stop the current system, and install and put into operation the new system, every time that users or policies change. Therefore, it would be the most desirable that this information could be updated at runtime and by end users, in an intuitive and easy way. For this reason, we develop an end-user tool, which is explained in detail in Section 6.1.1. This tool allows system users update its personal and contact information when required; and also allow users with the corresponding permissions to modify the security policies. This tool provides a web interface, an interface very intuitive for the users of AmI systems.

Furthermore, user interfaces must also adapt to user information in order to only the users that are registered in the system can access to it and a user only can use the services that its policy allows. To do this, we have extended the interface layer provided by the method presented in Chapter 4. Thus we get ensuring the security and privacy of the system. We explain this extension in detail in Section 6.1.2.

6.1.1 An End-User Tool for managing User Information at Runtime

User information must be able to be managed by the system end-users as any service of the system, in such a way that these services are available for the users whose policy has permission to manage this information (e.g. only users with the Administrator policy will be able to change the policies of system users and each user will be able to change only its own private information). To allow this, we have developed an end-user tool for managing this information. This tool has been developed by following the philosophy of the proposed method.

Thus, we have specified, by using the PervML models, and generated, by using the generation tool provided by the PervML method (Section 4.2.7), two new services: the *UserManagement* service and the *PolicyManagement* service. These services will be available for every system, developed or generated by following our method or not, by using the interfaces that our method provides to control the system. In addition, we plan in further work developing a tool to give support to the whole development of context-aware pervasive systems; this tool will have specified these services by default in the corresponding models of PervML.

As we have said, to develop these services, our approach allows us to automatically generate their code from their specification by using the proposed models. However, the drivers that use these services have to be manually implemented. These drivers will be in charge of managing the information in the OWL repository. Thus, our approach provides us with technology independence, in such a way that if we changed the repository for using another technology instead of OWL, we would only have to change the implemented drivers, because the rest of the code would be the same.

Therefore, in order to create these services by following our approach, we have followed the following steps:

1. Specify the type of service by using the Services Model. To do this, we specify two new services in this model by indicating: the operations that it will have and the pre and post conditions of each operation; and the state machine of the service. Fig. 31 shows the specification of these type of services by using the tool presented in Section 4.2.3. As we can see in this figure, we have specified the *UserManagement* and the *PolicyManagement* as UML classes. Each one of these classes contains the operations that each service provides. In addition, Fig. 31 shows at the bottom the state machine of each service. Both of them are very similar, with an only state, the *Created* state, and with four operations whose execution returns the service to this state.

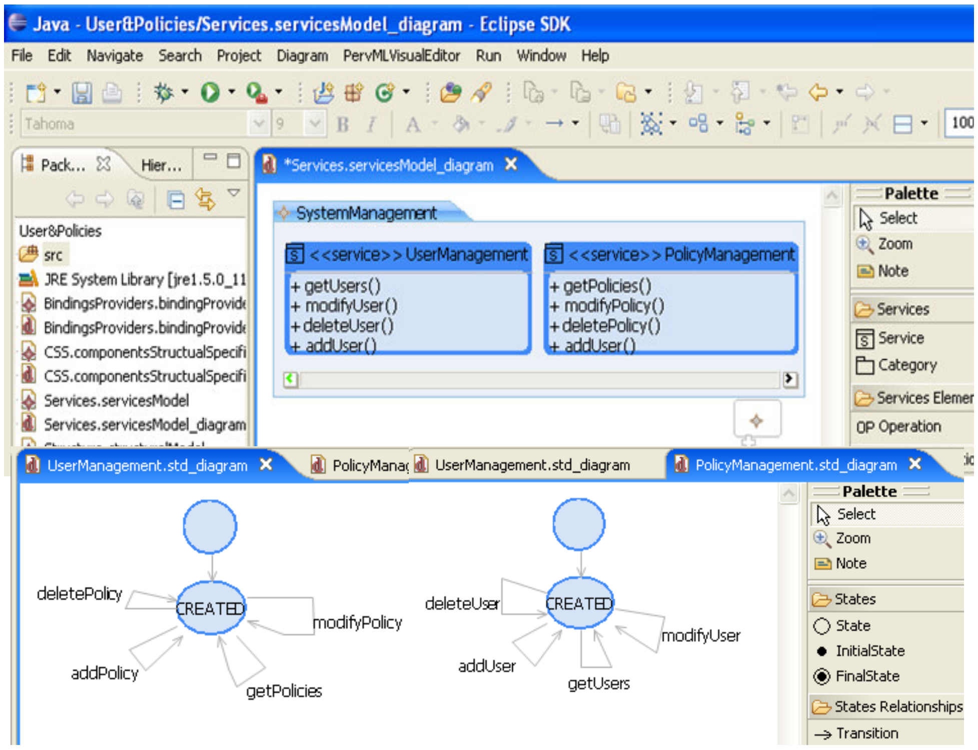

Java - User&Policies/Services.servicesModel_diagram - Eclipse SDK
File Edit Navigate Search Project Diagram PervMLVisualEditor Run Window Help
Pack...
Hier...
User&Policies
src
JRE System Library [jre1.5.0_11
BindingsProviders.bindingProvide
BindingsProviders.bindingProvide
CSS.componentsStructualSpecifi
CSS.componentsStructualSpecifi
Services.servicesModel
Services.servicesModel_diagram
*Services.servicesModel_diagram
SystemManagement
<<service>> UserManagement
+ getUsers()
+ modifyUser()
+ deleteUser()
+ addUser()
<<service>> PolicyManagement
+ getPolicies()
+ modifyPolicy()
+ deletePolicy()
+ addUser()
Palette
Select
Zoom
Note
Services
Service
Category
Services Elemen
OP Operation
UserManagement.std_diagram
PolicyMana
UserManagement.std_diagram
PolicyManagement.std_diagram
deletePolicy
CREATED
modifyPolicy
addPolicy
getPolicies
deleteUser
CREATED
modifyUser
addUser
getUsers
Palette
Select
Zoom
Note
States
State
InitialState
FinalState
States Relationships
Transition

Fig. 31 A partial view of the Services Model for the user and policy management

2. Specify the service by using the UML Component diagram of the Structural Model. To do this, we specify two new components by indicating the type of service that they provide and the location where it will be deployed. Fig. 32 shows the specification of these services by using the tool presented in Section 4.2.3. As we can see in this figure we have specified two components, the *UserManagementComponent* and the *PolicyManagementComponent*. These components respectively provide the *UserManagement* and *PolicyManagement* services, and are located in the smart home.

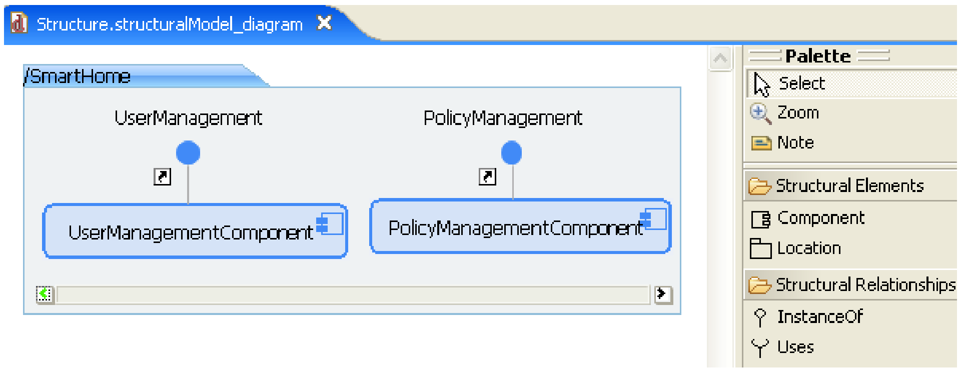


Fig. 32 The UML Component Diagram for the user and policy management

3. Specify the driver interfaces by using the Binding Provider Model. To do this, we only have to specify the methods that each driver must have. Fig. 33 shows the specification of binding providers by using the tool presented in Section 4.5.2. As we can see in this figure, we have specified the *UserManagementBP* and the *PolicyManagementBP* as UML classes. Each one of these classes contains the operations that each interface has.

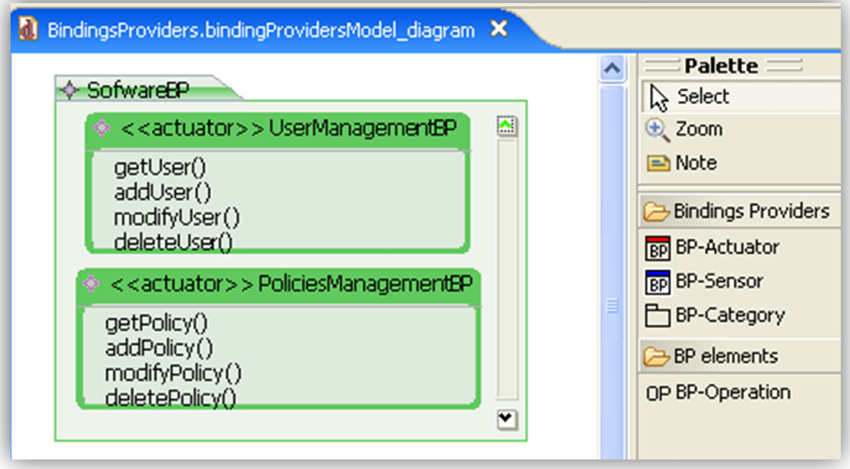
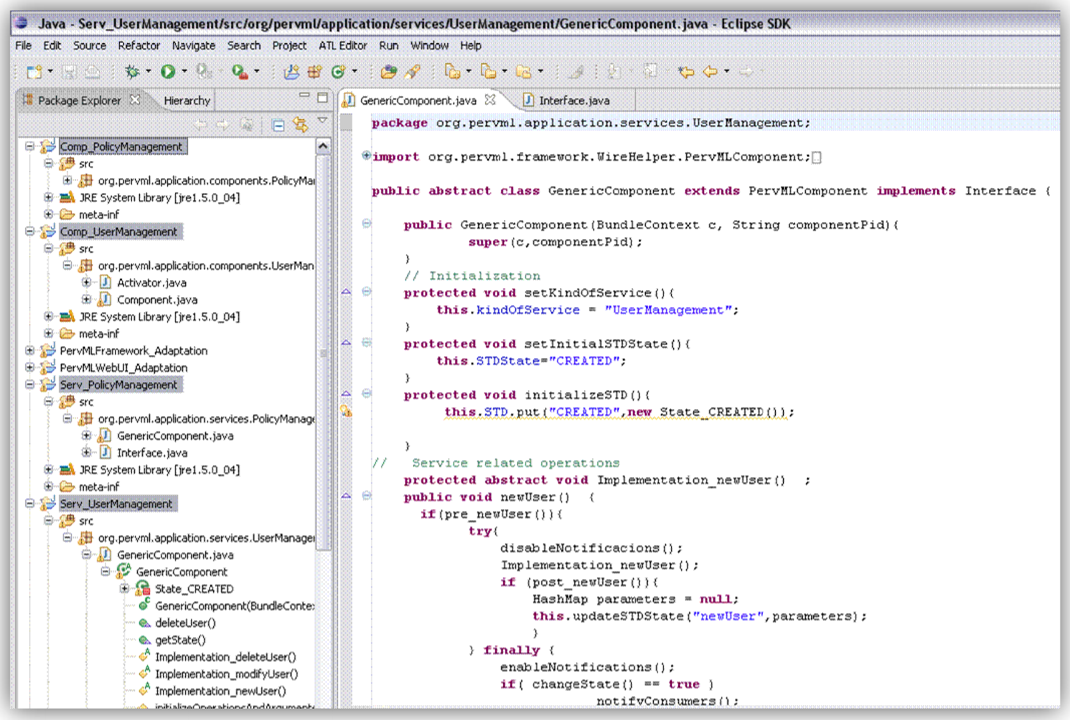


Fig. 33 The Binding Provider Model for the user and policy management

4. Apply the automatic code transformations that translate the above specified models into Java-OSGi code. To do this, we have also used the tool presented in section 4.2.3. Fig. 34 shows brought out some of the projects generated from the models and also shows a partial view of the code of the *UserManagement* service.



```
Comp_PolicyManagement
    src
        org.pervml.application.components.PolicyMa
    JRE System Library [jre1.5.0_04]
    meta-inf
Comp_UserManagement
    src
        org.pervml.application.components.UserMan
            Activator.java
            Component.java
    JRE System Library [jre1.5.0_04]
    meta-inf
PervMLFramework_Adaptation
PervMLWebUI_Adaptation
Serv_PolicyManagement
    src
        org.pervml.application.services.PolicyManage
            GenericComponent.java
            Interface.java
    JRE System Library [jre1.5.0_04]
    meta-inf
Serv_UserManagement
    src
        org.pervml.application.services.UserManage
            GenericComponent.java
                GenericComponent
                    State_CREATED
                    GenericComponent(BundleConte
                    deleteUser()
                    getState()
                    Implementation_deleteUser()
                    Implementation_modifyUser()
                    Implementation_newUser()
```

```
package org.pervml.application.services.UserManagement;

import org.pervml.framework.WireHelper.PervMLComponent;

public abstract class GenericComponent extends PervMLComponent implements Interface {

    public GenericComponent(BundleContext c, String componentPid){
            super(c,componentPid);
    }
    // Initialization
    protected void setKindOfService(){
        this.kindOfService = "UserManagement";
    }
    protected void setInitialSTDState(){
        this.STDState="CREATED";
    }
    protected void initializeSTD(){
         this.STD.put("CREATED",new State_CREATED());

    }
//   Service related operations
    protected abstract void Implementation_newUser()   ;
    public void newUser()   {
      if(pre_newUser()){
            try{
                disableNotificacions();
                Implementation_newUser();
                if (post_newUser()){
                    HashMap parameters = null;
                    this.updateSTDState("newUser",parameters);
                    }
            } finally {
                enableNotifications();
                if( changeState() == true )
                            notifyConsumers();
```

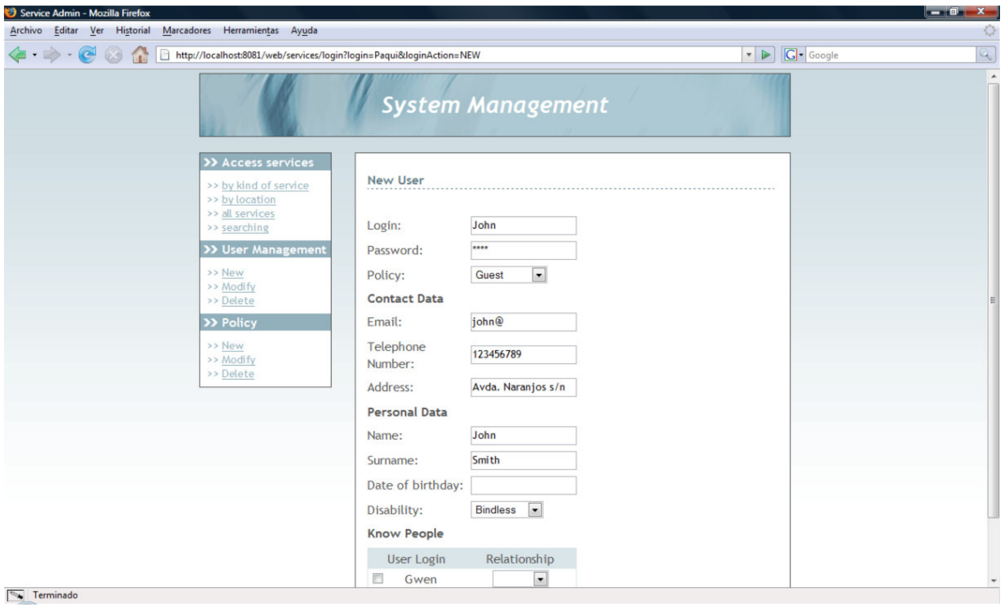
Fig. 34 A screen shot that shows part of the generated code from the specification of the User and Policy Management

5. Implement the drivers that respectively manage the users and policies of the system. To do this, we have implemented a driver for each service. These drivers implement the methods defined in their interfaces, which provide the functionality required by the *UserManagementComponent* and the *PolicyManagementComponent* services. These methods use the CommunicationWithOWL class to add, modify, delete and get the individuals of users or policies. For instance, Fig. 35 shows the method that add a new user. As we can see in this figure, it forms the attributes that the *addIndividual* method of the *CommunicationWithOWL* class needs to add the corresponding individual to the context repository. To do this, 1) it indicates the name of the class (*Person*) and the name of the individual; 2) it creates the different properties of the newly user individual; 3) it creates the different relationships with other individuals; and 4) it calls the addIndividual method of the *CommunicationWithOWL* class to add the new user. We only show in Figure 35 the creation of a few properties and relationships; the rest of properties and relationships are analogously created.

```
    public void addUser(User user){
    String clase="Person";
①   String ind=user.login;

    //Preparation of the attributes (Personal data, Contact data, etc.)
    HashMap <String,Tupla> attributes = new HashMap<String,Tupla>();
                 List <Tupla> type_value = new ArrayList();

②   Tupla email_tupla=new Tupla("http://www.w3.org/2001/XMLSchema#string",
    user.contactData.get("email"));
    type_value.add(email_tupla);
    attributes.put("email", email_tupla);

    Tupla disability_tupla=new
          Tupla("http://www.w3.org/2001/XMLSchema#string",
          user.personalData.get("disability"));
    type_value.add(disability_tupla);
    attributes.put("disability", disability_tupla);
    …
    // Preparation of the relations (Relationships with other users and policy)
    HashMap<String, List> relations= new HashMap<String, List>();
    Set knownPeople_set=user.knownPeople.keySet();
    Iterator knownPeople_it=knownPeople_set.iterator();
    List<String> individuals = new ArrayList<String>();//null;
    String related_person;
    String relation_with;
③   while(knownPeople_it.hasNext()){
    related_person=(String)knownPeople_it.next();
    relation_with=(String)user.knownPeople.get(related_person);
    individuals.add(related_person+"_"+ relation_with);
    }
    relations.put("isRelatedWith", individuals);

    List<String> policy = new ArrayList<String>();
    policy.add(user.namePolicy);
    relations.put("apply", policy);

④   CommunicationWithOWL.insertarIndividual(clase, ind, attributes, relations);
    }
```

Fig. 35 Method from the *UserManagement* Driver to add a new user to the OWL Context Repository

6. Finally, we export as jars the generated code (step 4) and the drivers (step 5) and install them in an OSGi server in order to provide the services to users. As we have modelled the management of users and policies like PervML services, the web interface that was provided by the method to develop pervasive systems (Chapter 4) provides us with a web user interface by default for these services. For instance, Fig. 36 shows the user interface to add a user to the system.



New User

Login:	John
Password:	****
Policy:	Guest
Contact Data	
Email:	john@
Telephone Number:	123456789
Address:	Avda. Naranjos s/n
Personal Data	
Name:	John
Surname:	Smith
Date of birthday:	
Disability:	Bindless

Know People

User Login	Relationship
Gwen	

Fig. 36 Screen shot of the web interface to add a new user to the system

6.1.2 Interface Layer implementation

So that system interfaces adapt according to user information, we have extended the Interface Layer of the framework presented in Section 4.2.1. This layer is in charge of giving support to the presentation of information and services to users. It also provides facilities for supporting multiple user interfaces by using the Model-View-Controller (MVC) pattern [48].

The controller class of the MVD pattern is in charge of processing the incoming events from the User Interface and, by consulting the information of the Model, the invoking the appropriate View which displays the required output. The models are the PervML models, which information is stored in our context repository. And the views of the web interface are provided by a set of java servlets that invoke the Controller operations to generate the web pages that are shown to users. Thus, to ensure the security and privacy of the system, we have extended the *Controller* class with a set of methods for consulting the information related with system users and their policies; and we have implemented the *LoginUserServlet* class, which implements a new Java Servlet, to check the user autentification.

On the one hand, the *Controller* class provided us with methods that allow users: (1) To select the component that is the specific service with which the user wants to interact;

(2) To interact (get information and request functionality) with a single component. Therefore, we have added a set of methods to ensure the security and privacy in the System. The goal of this set of methods is to provide functionality for checking the user identification and to only show to user the services and the operations for which the user has permission (according to its policy). In order to support this objective, the following methods are provided:

- List list_usersLogin(): This method returns a list that contains all the logins of the system users that are currently registered in the pervasive system.
- Boolean exist_the_login(String login): This method checks if the login passed as parameter belongs to any user currently registered in the pervasive system. If this login exists, the method returns true, else returns false.
- Boolean login_password (String login, String password): This method checks if the login passed as parameter belongs to any user currently registered in the pervasive system by calling to the *exist_the_login* method, and checks if the password passed as parameter is the password of this login.
- List list_policiesName():This method returns a list that contains all the policies that are currently created in the pervasive system.
- String getPolicyOfAUser(String user): This method returns identifier of the policy of the *user.*
- Boolean exist_the_policy(String policy): This method checks if the policy passed as parameter is currently created in the pervasive system. If this policy exists, the method returns true, else returns false.
- List getServicesOfAPolicy(String policy): This method checks if the policy passed as parameter is currently created in the pervasive system by calling to the *exist_the_policy* method, and if so, this method returns a list that contains all the services that the policy has allowed.

Fig. 37 Extended Layer Interface and Web Interface

On the other hand, the *LoginUserServlet* class provides the view of the web interface for making the logging of a user (as further work we incorporate new technologies for making the logging, as identification chips or fingerprint recognition), which is shown in Figure 37, and invokes the Controller operations in order to generate the web pages ensuring that each user only can see and execute the operations for which the user has permission. For instance, when a user select to show the services of the system:

1. First, the *LoginUserServlet* gets the user login of the web page, if it is not found; the class shows the logging web page.
2. Next, this servlet gets the list that contains the services related with the policy of the user that has the corresponding login. To do this, this servlet uses the *getPolicyOfAUser* and *getServicesOfAPolicy* methods of the Controller class.
3. Finally, according to the search type (by location, by kind of service, etc.), the servlet shows the corresponding services that are also in the list obtained in the second step.

Figure 37 shows, as well as the *Controller* class, all the implemented classes to generate the web user interface and the logging web page which is the first web page that this interface shows. It is worth to noting that every class in this figure can be reusable for all the pervasive systems that are developed using the proposed approach. This feature is feasible since (1) every PervML service implements an interface that is known by the

Controller and (2) the Java reflection capabilities have been used to invoke previously unknown methods.

6.2 Anticipating the Next User Action

In this section, we explain how system can adapt itself according to user behaviour. To give support this adaptation we have implemented the *Adaptation* class in the framework presented in Chapter 4. This class is in charge of asking the adaptations when they are needed if the opportune conditions are satisfied. This class has been implemented to support different types of adaptations such as the adaptation to specific weather conditions, the consideration of user's habits, or the anticipation of the next user action. Up to now, we have developed the last one and are working to make the other adaptations. Thus, in this section, we present how we carry out the anticipation of the next user action.

Anticipating the next user action implies to execute a specific service operation (e.g. turns lights on) when the user need it and before the user explicitly requests its execution (e.g. before users activates the switch). To do this, we propose a strategy based on the use of machine learning algorithms. There are several proposals of machine learning algorithms that can be used to predict user actions [22, 53, 54]. Basically, these algorithms take the last performed user action as source, and try to identify patterns of actions that include this action along a given sequence of actions. In these patterns of actions, the source action must appear before the last position. Once a significant number of these patterns are identified, these algorithms select the pattern that is most frequently performed in the current state of the system. Then, they return the action of the pattern that appears after the executed action as the predicted action. The probability of the predicted action performing by the user is also returned. This probability is calculated from both the frequency in which the pattern of actions appears along the whole sequence of actions and the length of the pattern.

Our objective is to provide an implementation infrastructure that makes the use of any existing machine learning algorithms independent from the rest of the system. This infrastructure is based on the *Adaptation* class, which is in charge of controlling the whole process of prediction. However, it is complemented with three additional elements (see Figure 38):

- A class which implements the selected prediction algorithm.
- The interface *NextActionPrediction*, which describe the methods that an implementation of a prediction algorithm must have.
- The class *ActionExecution*, which gives support to execute the predicted action.

Note, that this way of implementing and using machine learning algorithms provides us with a great flexibility to change them, even when the system is already deployed. To use these algorithms they must be implemented according to an interface. Thus, any of these implementations present the same behaviour (i.e. the same methods) and then, the Adaptation class must always do the same method invocations (independently from the implemented algorithm). Therefore, the Adaptation class does not need to be modified although the machine learning algorithm changes.

In order to predict the next user action, an Adaptation object is created when the system is started up. Then, every time that an action is added to the OWL Context repository, this object performs the following actions (see Figure 38):

1. It gets the whole list of user actions through the CommunicationWithOWL object by using the *GetIndividuals* method. Applying the corresponding filters, we first get the last action performed in order to know the user that executed it, and we next get all the actions performed by this user.
2. The Adaptation object creates an object of the class that implements the machine learning algorithm and passes to it the list of user actions and the last action performed by the user.
3. The prediction algorithm is applied and the Adaptation object obtains the predicted user action and the probability of being performed.
4. If this probability is greater than a given threshold, the Adaptation object creates an *ActionExecution* object and passes the predicted action to it in order to be executed. This execution is performed by interacting with the framework presented in Section 4.2.1.

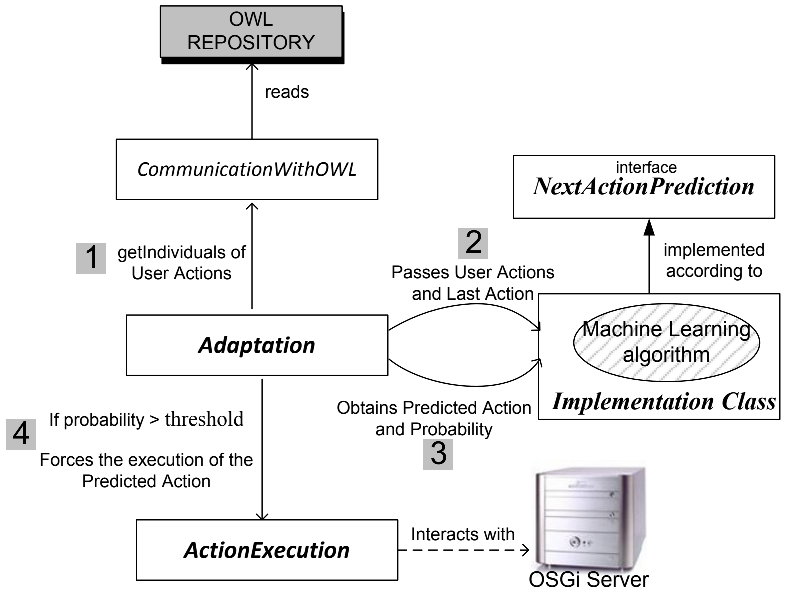
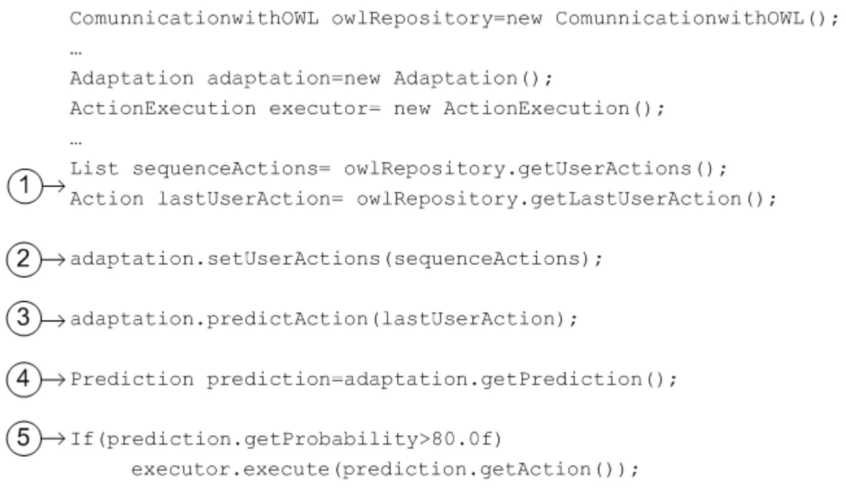

OWL REPOSITORY
reads
CommunicationWithOWL
interface
NextActionPrediction
1
getIndividuals of User Actions
2
Passes User Actions and Last Action
implemented according to
Adaptation
Machine Learning algorithm
Implementation Class
If probability > threshold
4
Forces the execution of the Predicted Action
Obtains Predicted Action and Probability
3
ActionExecution
Interacts with
OSGi Server

Fig. 38 Predicting the Next User Action

Implementation Aspects. We have implemented a prediction algorithm based on the SHIP algorithm presented in [22]. This implementation has been done according to the *NextActionPrediction* interface. This interface basically indicates that three methods must be implemented: (1) *setUserActions*, which establishes the sequence of user actions to be analyzed, (2) *predictAction*, which receive as parameter the last user action and apply the machine learning algorithm, and (3) *getPrediction*, which returns the predicted action and its associated probability. Figure 39 shows an example of code where the scenario illustrated in Figure 38 is implemented. First, (1) the set of user actions and the last user action is obtained through the *CommunicationWithOWL* object. Next, (2) we pass this sequence to the machine learning algorithm and (3) we execute this algorithm for the last user action. Finally, (4) we obtain the prediction and (5) if the associated probability is greater than the 80% we pass the action to an object *ActionExecution*, which execute it. To perform this execution, we have integrated our framework with the method presented in Chapter 4 by using its OSGi-Java based framework. Thus, the *ActionExecution* object uses the *FrameworkSearcher* class that this framework provides us in order to search and gets the corresponding system service and then uses the Java reflection capabilities in order to invoke the corresponding method of this service. As further work, we will evaluate and consider the effect of an automatically executed action that the user did not want to execute.



```
   ComunnicationwithOWL owlRepository=new ComunnicationwithOWL();
   …
   Adaptation adaptation=new Adaptation();
   ActionExecution executor= new ActionExecution();
   …
   List sequenceActions= owlRepository.getUserActions();
①→ Action lastUserAction= owlRepository.getLastUserAction();

②→ adaptation.setUserActions(sequenceActions);

③→ adaptation.predictAction(lastUserAction);

④→ Prediction prediction=adaptation.getPrediction();

⑤→ If(prediction.getProbability>80.0f)
        executor.execute(prediction.getAction());
```

Fig. 39 Example of Java code for predicting the next user action

6.3 Conclusions

We have explained in this chapter the mechanisms that we have developed in order to both ensure the privacy and security of the system, and adapting the system to user behaviour by automatically performing user actions when needed without explicit user intervention. As for the first requirement we have developed an end-user tool to properly manage context information and adaptive interfaces that are in charge of establishing and enforcing user defined policies. As for the second requirement, we have extend the framework presented in the previous chapter to allow the anticipation of the next user action.

7. Execution Strategy

Once our approach has been introduced, a brief description of the global execution strategy is needed in order to understand how the developed pervasive system works. The execution strategy is the rules that define the sequence of actions which are carried out when the system is running.

According the proposed architectural style, as we have said, two kind of events can start a sequence of actions in the pervasive system:

1. A driver notifies a change on the (physical or logical) environment.
2. A user requests the execution of an operation by means of a user interface.

Both situations could derive into the invocation of a component operation, where a very important part of the PervML execution strategy is embedded. Therefore, next sections introduce the sequences of action that are carried out in these three situations (1) a driver notifies of a change in the environment, (2) a user requests a functionality by means of a user interface and (3) a component operation is invoked.

7.1 Change in the Environment

When a *Driver* notifies a *BindingProvider* that a change in the environment has been detected, the *BindingProvider* informs about this event to the *Components* that make use of it. These *Components* must evaluate their *Triggers*, since now some condition may hold. Moreover, they also must notify to their related *Components* and *Interactions* because their triggering condition may also hold with the new situation.

In detail, the following steps are carried out:

1. A Driver notifies to its related BindingProvider that a change in the environment has been detected.
2. The BindingProvider notifies to all the Components that make use of it about the change. Of course, it does not inform about what has changed.

3. The Component checks is its state has changed (e.g. the lighting was off and after execution of the operation lighting was on) by means of the execution of its operations for consulting its state, in other words, those operations which return a value and do not receive parameters; for instance: “int getIntensity()” or “boolean isLighting()).
    a. If the returning values do not changed from the last check:
        i. The sequence of action ends.
    b. If the returning value has changed from the last check:
        i. Store the new values.
        ii. Continue with the next steps.
4. The Component creates and stores an individual of the executed action in the OWL Context Repository. It also activates the reasoning in order to derive the corresponding context information from the new added individual.
5. The Component checks the conditions of its Triggers. When evaluating these condition expressions, Components invoke operations from other Components or from its own interface, which could call to functionality provided by the BindingProviders, including the BindingProvider that detected the environment change.
6. The Component informs that a change in any of its operations has happened to the elements that are listening to its change notifications.
    a. The Adaptation class is subscribed to all components, thus, it always receive the notification. When it happens:
        i. The *Adaptation* class uses the implementation of the machine learning algorithm to predict the next action.
        ii. If the probability of the prediction is higher than 80%, th*e* *Adaptation* Class use*s* the *ActionExecution* class to execute the predicted action.
    b. Also, other Components or Interactions can be listened to its change. When they receive the notifications, they evaluate their triggering conditions, which imply invoking some operation from the Component that notified the change.
7. Occasionally, as a result of the steps 3 or 6, an operation of a Component is invoked or an Interaction is initiated.

7.2 User Request

When a user wishes to request a functionality that is provided by a service of the pervasive system, he or she uses a user interface (he presses a button, passes the point by a screen region, says a voice command, etc.). That interface interacts with the controller element, which redirects the invocation to the suitable Component.

In detail, the following steps are carried out:

1. The User interacts with a user interface (a View) for requesting a functionality that is provided by service of the pervasive system. The View is the responsible of gathering the arguments of the operation using the mechanisms more adequate according it its characteristics.
2. The View sends to the Controller element a request for invoking an operation over a Component with a list of arguments.
3. The Controller searches the Component in the OSGi environment and, if the request operation is enabled in that moment, it invokes the operation using the Java reflection mechanisms.

7.3 Operation Execution

The execution strategy of the Components operations embeds a critical part of the overall execution strategy, since it uses most of the system elements. In short, it must ensure that all the constraints that were specified (pre, post-conditions and state machine) are satisfied and suitably updated. The algorithm presented in Fig. 40 in pseudo code summarizes the rationale that has been applied.

```
IF precondition hold and
   the operation is the trigger of an outgoing transition from the
   current active state
DO
       disable change notifications
       execute operation actions
       IF postcondition hold DO
              update state machine
       ELSE
              log error
              stop
       ENDIF
       enable change notifications
       IF component state has changed DO
              notify listeners
       ENDIF
ELSE
       log error
ENDIF
```

Fig. 40 Algorithm followed to execute an operation

In detail, the following steps are carried out:

1. First of all two checks are done:
    a. The precondition of the operation is evaluated to check if it holds. This could imply invoke other Component operations or operations of related Components. If the precondition expression is not satisfied, the failure is logged and notified to the user, and the execution of the operation is stopped.
    b. The state machine is checked to see if the operation is the trigger of an outgoing transition from the current active state. If there is not any outgoing transition, the operation cannot be invoked and, therefore, the corresponding errors are logged and notified to the user, and the execution of the operation is stopped.
2. The mechanism of changes in the system environment is paused, since interruptions during the following steps could cause system inconsistencies. Note that the following step is executing the operation actions. Imagine that an action could switch on the lights in a room and a lighting sensor may detect this event. If this event is received and managed before the lighting Component updates its state to switch on, some system element may infer that is the daylight who is lighting the room.
3. The actions that implement the operation functionality are executed. In order to carry out this step, the following strategy is applied:
    a. The methods that implement the operation functionality are invoked and the required computations are executed. If any execution occurs, the error is logged and the execution is stopped. If the operation returns a value, it is stored in a variable called *returnValue*.
    b. If the operation returns a value, the *returnValue* variable is returned.
4. The postcondition of the operation is evaluated to check if it holds. This could imply invoke other Component operations or operations of related Components. If the postcondition expression is not satisfied, the failure is logged and notified to the user. If it holds the state machine must be updated.
5. The mechanism of changes in the system environment is re-activated.

6. Provided that the executed operation is not an operation of the component to check its state, the Component calculates its state by using the operations which return a value and do not receive parameters; for instance: “int getIntensity()”, “boolean isLighting()). If any value has changed from that last check, the following steps are done:
   a. The new values are stored.
   b. The Component creates and stores an individual of the executed action in the OWL Context Repository. It also activates the reasoning in order to derive the corresponding context information from the new added individual.
   c. Finally, the subscribed elements are notified about a change.
      i. The Adaptation class is subscribed to all components, thus, it always receive the notification. When it happens:
         1. The *Adaptation* class uses the implementation of the machine learning algorithm to predict the next action.
         2. If the probability of the prediction is higher than 80%, th*e Adaptation* Class use*s* the *ActionExecution* class to execute the predicted action.
      ii. Also, other Components or Interactions can be listened to its change. When they receive the notifications, they evaluate their triggering conditions, which imply invoking some operation from the Component that notified the change.
   d. Since the Adaptation class is subscribed to all components, :
      i. It uses the implementation of the machine learning algorithm to predict the next action.
      ii. If the probability of the prediction is higher than 80%, th*e Adaptation* Class use*s ActionExecution* to execute the predicted action.

7.4 Conclusions

We have specified in this chapter the global execution strategy that the context-aware pervasive systems developed by using our approach follow when are running. The execution strategy is defined for the two kind of events that can start a sequence of actions in these systems.

8. Developing a Context-aware Pervasive System

Along this work, we have introduce both an MDD approach that allow us to capture Context at modelling time and an ontology-based repository with a high level implementation framework that allows to capture and managing Context information at runtime. In addition, we have also explained that these contributions have been integrated into the PervML method in order to provide a complete MDD method to develop context-aware pervasive systems. In this section, we explain the steps that must be followed in order to develop a context-aware pervasive system by using this method.

The proposed method divides the development of a context-aware pervasive system into two main phases: the development phase and the deployment phase. Next, we explain each of these phases in detail.

8.1 The Development Phase

The development phase obtains the code to put the system into operation. This phase, which is shown in Fig. 41 consists in the following three steps:

1. **Conceptual modelling**. On the one hand, pervasive system analysts specify the system requirements by using the *service* conceptual primitive. To do this, pervasive system analysts specify in this order: The Services Model (which has been explained in Section 4.1) and the Structural Model, the Interaction Model, and the User Model that we propose (which have been explain in Section 5.1). By means of these models, analysts respectively describe (1) the kind of services available on the system, (2) the locations of the environment and the number of services which are available in every location, (3) how they interact when some condition holds and (4) privacy policies as well as information of each system user.

On the other hand pervasive system architects select the kind and number of devices or software systems that are more suitable in order to provide the services specified by the analyst. The selection could have into account economical reasons or constraints in the system physical environment. System Architects use other three models that have to be specified in this order: The Binding Provider Model, the Component Structure Model and the Functional Model. By means of these models, architects respectively describe (1) the kind of devices or software systems that are used for providing the system services, (2) the specific elements that are going to implement every service and (3) the actions that the device or software systems must carry out for providing every service operation.

2. **Code Generation**. From the specified PervML models, two model transformations are executed in order to translate them into both Java code and the OWL Context Repository.

   The first transformation engine automatically transforms the specified models into Java code. This Java code consists of Java files and Manifest files that implement the functionality that supports the system services. This code is based on the implementation framework explained in Section 4.1.1 that provides a common architecture for all the systems that are developed by using the method. This transformation has been explained in detail in Section 4.2.2.

   The second transformation automatically transforms the models into the OWL Context Repository, which describes the context aware pervasive system that is available at design time by using concepts of our ontology. OWL Context Repository will be continually updated at runtime by the framework for the adaptation according to the changes produced in the system.
3. **Driver implementation**. Finally, drivers for managing the selected devices or software systems have to be implemented by an OSGi developer. Drivers must be developed by hand, since they deal with technology-dependent issues. However, if any device or external software system has been used in a previous system, the same driver can be reused; thus, we can have a driver repository and only implement the drivers for the devices or software systems that have not been used before. Moreover, in a previous work we implemented a European Installation Bus (EIB) driver generator [55] to generate the EIB driver code from a simple description.

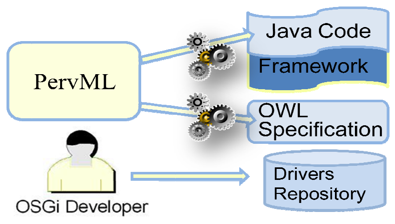


Fig. 41 The development phase

8.2 The Deployment Phase

The development phase provides us with all the context-aware pervasive system code. In order to start up the system it is only necessary the following steps, which are shown in Fig. 42:

1. **Configuration**: The Java files that use the manually implemented drivers are associated with the selected drivers. To do this, we just need to set up the driver identifiers.
2. **Bundle installation**: The files generated from the PervML to Java code transformation are compiled, packaged into bundles (JAR files) and deployed in an OSGi server along with the framework and the drivers. Also the implementation framework, which has been explained in Section 5.3 to support the system adaptation is deployed in the OSGi server. The generated OWL Context Repository must be copied in the carpet where the OSGi server is located.
3. **Starting Bundles:** Finally, the installed bundles are started in the OSGi server. Thus, the services are already available to users who can execute them through the interfaces [24] that are provided by the implementation framework (as explained in Section 6.1).

Fig. 42 The deployment phase

8.3 Conclusions

We have described in this Chapter a methodological guidance that can be used to develop context-aware pervasive systems by following the approach presented in this master thesis. This guidance has been specified by precisely enumerating a sequence of the well-defined steps that guide developers since the description of the system until its put into operation.

9. A Case Study

In this chapter, we present a case study that we have implemented in order to validate our work.

9.1 A Pervasive System for managing a Smart Home

In order to validate our work, we have implemented a context-aware pervasive system for a smart home. This system has been used as running example along the whole paper. The main goal of this system is to improve everyday life by addressing all vital user aspects such as home care and safety, comfort, entertainment, etc. In particular, the services that we have been implemented for this case study are the following:

- **Multimedia Management**: this service provides inhabitants with support to store, manage and reproduce multimedia archives.
- **Intelligent lighting**: controls the lighting according to both user presence and light intensity. For instance, the intensity of the lighting is increased as the outside light is decreased.
- **Air conditioning**: is in charge of achieving the optimum temperature.
- **Security**: this service controls the security of the home. It can be activated or deactivated. If the presence of someone is detected when it is activated, an alarm starts to ring; the system starts to record, informs users and, if necessary, sends a warning to the alarm central.
- **Blind Management**: allows inhabitants to control the blinds of the home.
- **Cooking**: this service allows inhabitants to control and program kitchen's electrical appliance in order to cook meals. It also controls the amount of feeds that there must be always in the home to satisfy the inhabitants' preferences.
- **Alarm Clock**: this service manages the way in which inhabitants wants to wake up.

As we have explain in Chapter 8 , to obtain the functionality associated to each one of these services, they have to be specified by means of the proposed models and next we the Java-OSGi code and the OWL Context Repository are automatically generated from these models. However, as we have said, in order to allow users to take advantage of these services they need to be supported by the proper hardware devices which must be deployed in the proper locations of the home.

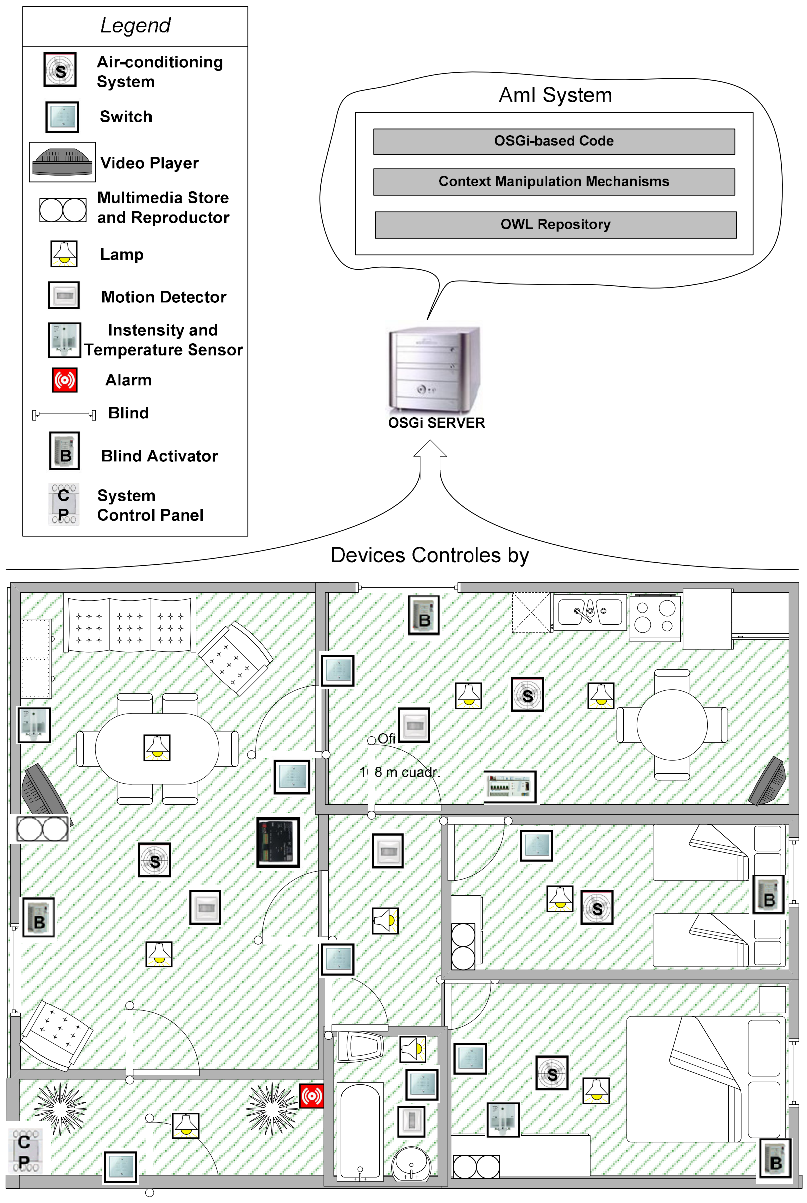


Fig. 43 Smart Home for the Case Study

The home for which we have implemented the context-aware pervasive system is made up of the following locations with the following devices (see Figure 43):

- **A hall**: in this location, an Intelligent Lighting service is supported by a *Lamp* and a *Movement Detector*. This *detector* is also used by a Security Service (which is defined for the whole home) in order to detect presence. There is also a *Switch* that allows users to interact with this service in order to manually activate or deactivate the lamp. With regard to the Security service, an *Alarm* is also situated in this location to be activated when intruders are detected. The *System Control Panel* is also situated in the Hall.
- **A living room**: In this location, there are two *Gradual Lamps* that are controlled by an Intelligent Lighting service. To allow users to control these Lamps, three *Switches* are situated around the Living room. As happens in the Hall, there is a *Movement Detector* that is used by the Intelligent Lighting service of this location and the Security service. Also the Intelligent Lighting service uses an illumination sensor to adapt to the illumination intensity level. A *Video Player and* a *Multimedia Store* that give support to the Multimedia Management service (which is created for the whole home). Finally, there is an Air-conditioning service which is supported by combining a *Temperature Sensor*, an *Air-conditioning device* and a *Blind Management service* which controls the two *Blinds* of the room. Users can also interact with this service through a *Blind Activator*.
- **A kitchen:** In this location, there are two *Lamps* that are controlled by an Intelligent Lighting service and a *Switch* that is used by inhabitants to control this service. There is a *Movement Detector* that is used by the Lighting service of this location and the Security service. There is also a *Video Player* and a *Multimedia Store* that support the Multimedia Management service. Finally, there is an Air-conditioning service which controls the temperature of the kitchen by using an *Air-conditioning device* and a *Blind Management service* which controls a *Blind*, which can be manually control by using a *Blind Activator*.
- **A corridor and a bathroom**: In these locations, there are a *Lamp* and a *Switch* that give support to the Intelligent Lighting service of each location. There is also a *Movement Detector* that is used by each Intelligent Lighting service and the Security service.
- **Two bedrooms**: In these locations, there are a *Lamp* and a *Switch* that give support to the Intelligent Lighting service of each bedroom. There is also a *Movement Detector* that is used by the Intelligent Lighting service and the Security service. Also, in each location there is an Alarm clock service which is supported by the *Multimedia*

Management service and the system clock. A *Blind* and a *Blind Activator* gives support to the Blind Management service of each of these locations. Finally, an Air-conditioning service maintains the optimum temperature in the bedrooms by combining a *Temperature Sensor*, an *Air-conditioning device* and a *Blind Management* service.

Furthermore, note that a *System Control Panel* is also situated in the Hall. This control panel is used to configure the AmI system. For instance, it is used to query the system state, to create users, to set the alarm clock up, to introduce user's preferences, to activate or deactivate the security service, etc.

9.2 A Validation Infrastructure

It is clear that the best way of validating the proposed case study is to deploy it in a real Home such as the one shown in Figure 43. However, the empirical validation is not easy to be performed within the academic environment in which this research work has been done. Thus, in order to execute the proposed case study and validate our work, we have developed an execution environment that allows us to emulate these real scenarios. This execution environment is shown in Figure 44 and it has been defined as follows:

- On the one hand, we have created a hardware infrastructure from a set of EIB devices such as switches, lamps, alarms, movement detectors, etc. EIB (European Installation Bus) is a network communications protocol for intelligent buildings, and currently, we can find a great variety of EIB devices that support the developing of AmI systems. The hardware infrastructure [55] that we have built is shown at the left side of Figure 44. The devices included in this infrastructure allow us to emulate the Intelligent Lighting services of the Hall and the Living Room, and the Security service. This infrastructure includes:
    - A Lamp and a dimmer to emulate the gradual lamps of the Living Room and a Switch to emulate the switches of this location. It also includes a movement detector that emulates the Living Room's Movement Detector. It is used by the Intelligent Lighting service of this location as well as the Security service.
    - Another Lamp to emulate the lamp of the Hall and a Switch to emulate the switch of this location. It also includes a movement detector that emulates the Hall's Movement Detector.

- A Red lamp and an Alarm Device to be used by the Security service.
- A programmer with RS232 interface is also deployed in this infrastructure in order to connect these devices with the AmI system.
- Finally, there is also a weather station which allows us to detect specific whether conditions. This device is not used for the purpose of this work. It will be used to validate further work related to the adaptation to weather conditions.

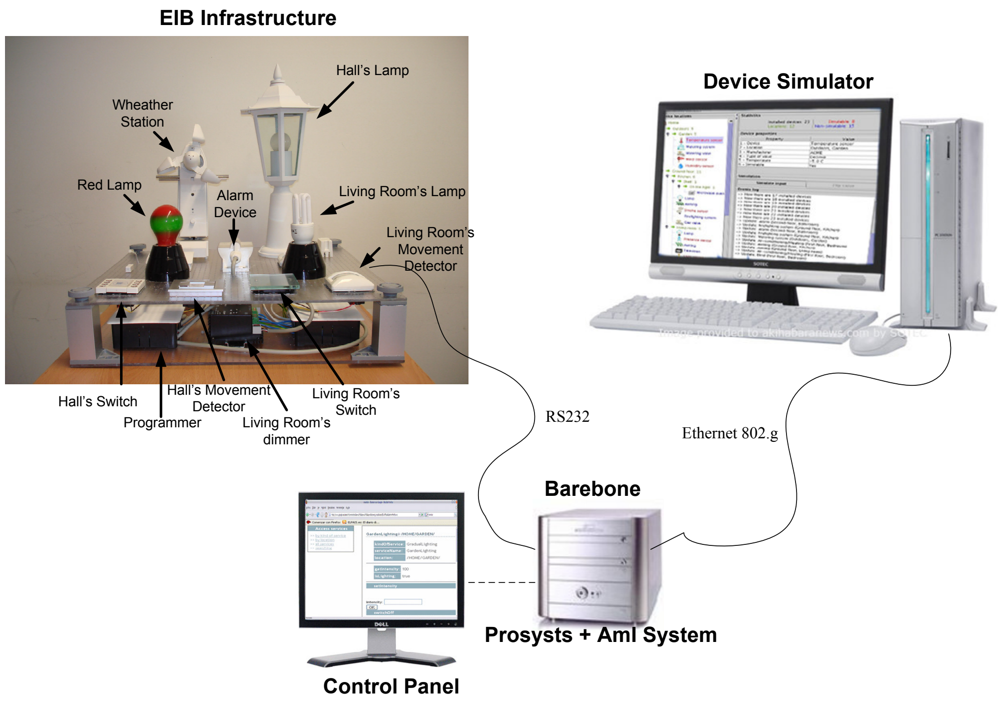


Fig. 44 Execution Environment

- On the other hand, we have used a Device Simulator developed in our research group that allows us to simulate the behaviour of the rest of devices by software. This simulator has been presented in [56]. It allows us to define virtual devices and to manage them by means of an intuitive user interface. It also provides mechanisms that allow OSGi-based code to control the virtual devices trough real EIB device drivers.

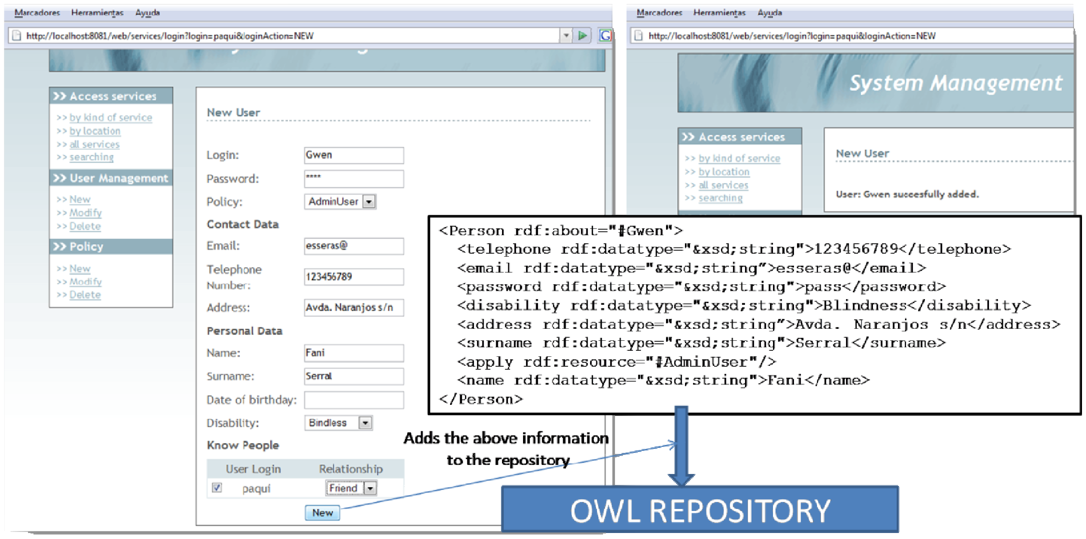
Finally, to control all these devices (both real and virtual), we have installed the AmI system (i.e. the OSGi-based code together with the OWL context repository and our framework for the system adaptation), with the Device Simulator, into a barebone Pentium IV with 1Gb RAM and with a Windows XP Professional Edition. This barebone has connectivity by Ethernet 802.g (which is used to connect it to the device simulator) and two RS232 ports (which are used to connect it to the EIB infrastructure). We have also included a Prosyst Embedded Server 5.2 in the barebone. Prosyst is a commercial OSGi server implementation. It is in charge of executing OSGi-based code in order to manage the devices. It also provides mechanisms to communicate with external networks and define Java APIs and several standard services like Logging, HTTP Server, Device Management, etc. Finally, we have also included in the barebone the Web interface that we have extended (Section 6.1) that plays the role of System Control Panel. By means of this interface we can execute operations provided by the system services, manage the information about the users and the policies of the system, etc.

9.3 Validation of Context Management at Runtime

In order to validate our work we have developed the case study presented in Section 9.1 and we have deployed it in the infrastructure introduced in Section 9.2.

Once the developed AmI system has been deployed in the validation infrastructure we have performed the following validations:

1. **Managing User information:** We have checked that user information is properly stored in the OWL Context Repository when it is managed at runtime. This implies to store properly the new created users or polices, as well as their association (each user is related with a policy); to correctly modify in the repository these information when it be required; and delete a user or a policy when it be required. To do this, we have used the Control Panel (the Web application executed in the barebone) to create several users such as, for instance, *Fani* and *Peter*, and the policies *Parents* and *AdminUser*, which are associated to the users. As result, the corresponding individuals of the concepts Person and Policy have been successfully created in the repository. We can see in Figure 45 how we have added the *Fani* user (whose login is *Gwen*) and the information that this web page adds to the OWL repository. As we can see, the web also shows a confirmation when the user is successfully added. In addition we have modified each property and relationship of the *Fani* user and the *AdminUser* policy and we have created and delete to *John* and *Parents* individuals.



New User

Login:	Gwen
Password:	****
Policy:	AdminUser
Contact Data	
Email:	esseras@
Telephone Number:	123456789
Address:	Avda. Naranjos s/n
Personal Data	
Name:	Fani
Surname:	Serral
Date of birthday:	
Disability:	Bindless
Know People	

	User Login	Relationship
✓	paqui	Friend

New User

User: Gwen succesfully added.

```
<Person rdf:about="#Gwen">
  <telephone rdf:datatype="&xsd;string">123456789</telephone>
  <email rdf:datatype="&xsd;string”>esseras@</email>
  <password rdf:datatype="&xsd;string">pass</password>
  <disability rdf:datatype="&xsd;string">Blindness</disability>
  <address rdf:datatype="&xsd;string”>Avda. Naranjos s/n</address>
  <surname rdf:datatype="&xsd;string">Serral</surname>
  <apply rdf:resource="#AdminUser"/>
  <name rdf:datatype="&xsd;string">Fani</name>
</Person>
```

Fig. 45 Web interface to add a new user

2. **Extending context information:** We have checked that user actions are properly added to the repository. In order to validate this, we have considering every type of action that can be executed: 1) those produced by a change in the environment: we can produce a change in the environment by interacting with the EIB infrastructure or with de Device Simulator; and 2) those directly required by a user: a user interact with the system by using the web interface. Thus, we have first activated and deactivated the switches of the EIB infrastructure. Next, we have opened/closed blinds and used the media player in order to play and stop a media file by using the Device Simulator. Finally, we have controlled the lighting intensity by interacting with the Intelligent Lighting service by using the web interface. In addition we have activated the Security service by using the web, we have moved to activate the living room’s movement detector, what has produced that the red lamp has blinked.

   In all these interactions the corresponding individual of the concept Action has been correctly created in the repository.
3. **Derived Context Information:** We have checked that SWRL rules are correctly applied at run time. To do this, we have interacted with the evaluation infrastructure in order to make each rule to be applied and check that the proper Context information is derived.

For instance, we have check that when the current position of a user changes, the location where he or she can go are updated (rule presented in Section 5.3.2). In order to update the *currentPosition* of a user we also have a SWRL rule. It is due to our limited infrastructure. In order to properly manage the location of each user we would need that each user had an identification device (e.g. a Radio Frequency Identification (RFID) bracelet) to be identified by the proper sensors. However, our validation infrastructure only includes movement detectors and they cannot identify users (just the presence of someone anonymous). Thus, we consider an only one user (e.g. Peter) in such a way that if any time a movement detector is activated, Peter is in its associated location. Therefore, we have in the rules repository a SWRL rule for updating Peter's user location by considering which movement detector is activated.

Thus, in order to validate that the current location of Peter and the locations where he can go are correctly updated, we have activated the movement sensor of different locations by following the mobility relationships defined among the locations (graphically described by the doors specified in the blue print of Figure 43). To activate the movement sensors we have used the Device Simulator and one of the movement detectors installed in the EIB infrastructure. For instance, we have activated first the Living Room's movement sensor (from the EIB infrastructure), next the corridor's movement sensor (from the Device Simulator), and next the Kitchen's movement sensor (from the Device Simulator). Each time that a movement sensor has been activated, the object properties *currentLocation* and c*anGo* of the individual Peter have been correctly updated to reference the proper locations.

4. **Prediction of the Next User Action:** We have checked that the next user action is correctly predicted. In this validation, we have established the prediction threshold in 80%. To perform this validation we needed to have a significant history of user actions in order to allow the algorithm to predict the next action with the specified probability. In addition, this history had to include sets of actions performed repetitively. In a real scenario, this history would be progressively created by the user behaviour. However, in this validation scenario we have needed to create it manually.

   To do this, we have first defined possible sets of actions that a common user could execute (e.g. in the morning, the user turn the alarm clock of the bedroom off and then s/he activate the coffee maker to prepare a coffee). Second, we have implemented a program that creates a significant history of actions where these sets of actions appear among other random set of actions. After the execution of this program we have manually performed the first action of the predefined sets of actions (e.g. we have turned the alarm clock off by means of the Device Simulator) and we have checked that the system has automatically executed the second action of the set. In all the cases checked, the action has been correctly anticipated when the prediction probability has been greater than the established one.

9.4 Conclusions

This chapter describes a full case study for a smart home in order to validate the approach presented in this master thesis to develop context-aware pervasive systems. In addition, this chapter describes the infrastructure and the tasks that have been carried out by using the developed case study in order to validate the contributions provided in this work.

10. Conclusions

In this last chapter, we introduce the conclusions of the work presented in this master thesis. First, we list the main contributions of this master thesis in the area of Context-Aware Pervasive Systems. Next, we explain the work that is currently being performed as well as future work. Finally, we enumerate the publications that have been obtained from the work of this master thesis.

10.1 Main Contributions

In this work, we have presented a hybrid approach to develop full functional context-aware pervasive systems. We have proposed a set of graphical OO models for capturing the requirements of the system and its context available at modelling time. We have also proposed a context ontology for managing context at runtime. In addition, we have provided a transformation engine to automatically obtain an OWL context repository based on this ontology from the models specified by developers. Lastly, we have provided an infrastructure that, by using this repository, automatically manages context, derives knowledge from it and anticipates the next user action. This infrastructure is also in charge of ensuring the privacy and security of the system.

Furthermore, we have integrated our approach with a method developed in our research centre that allows generating functional system from the conceptual models. This strategy also provides us with technological independence and automatic service discovery.

To sum up, the proposed approach provides us with:

1. A set of models that provides high-level abstractions to manage and handle context information of pervasive systems at modelling time.
2. A context ontology and an OWL context repository based in this ontology for being capable of store in a machine processable language both current context information and historical context information.

3. A transformation engine to automatically translate the proposed context models into the OWL Context repository.
4. A framework for managing the OWL context repository, interpreting this information at runtime and adapting the system to user behaviour. To do this, this framework automatically updates the OWL context repository according to the changes produced in context information at runtime. Moreover, it allows us reason about this information at semantic level in order to interpret it and derive knowledge from this information. Lastly, this framework provides mechanisms to anticipate the next user action when needed.
5. An end-user tool to manage the private user information and preliminary mechanisms to ensure the privacy and the security of the system.
6. The integration of our approach with a MDD method previously implemented by our research centre what allows us to obtain functional context-aware pervasive systems.
7. A methodological guidance that guides developers from the description of the context-aware pervasive system by means of the models that we propose to its deployment.

10.2 Current and Further Work

This master thesis is not a closed work and many research efforts can still be done in this research area. The main research activities that are currently underway and that we plan to face are the following:

1. Studying and selecting the suitable learning machine algorithms to extract behaviour patterns in order to be able to automatically execute them. By using the *Adaptation* class of the proposed framework we aim for automating not only the next user action, but also sets of actions that always occur at a regular time interval. Once identified the patterns, we will update the system to add an interaction that will be triggered at the right time. In addition, we will evaluate and consider the effect of automatically executing actions that the user did not want to execute in order to undo the corresponding actions if needed.
2. Generating the SWRL rules from models. Nowadays the reasoner rules have to be manually specified in SWRL in the rules repository. In further work we want to

propose a model to specify these rules by using high-level abstractions and transform this model into SWRL code by a model-to-code transformation.

3. Studying user preferences and the approaches propose to model them in order to adapt system behaviour according to these preferences. We also want to extend our interfaces in order to show the corresponding information taking into account user preferences.
4. Extending the conceptual models with mechanisms that allow us to specify services whose adaptation can be configured by end-users.
5. Developing more case studies for context-aware pervasive systems in order to properly validate our approach. We are currently developing two case studies, one for a car and another one for a researcher department.
6. Developing a tool to give support to the whole process of developing a context-aware pervasive system by following the presented approach.
7. Improving the security system mechanisms that have been proposed in this work by using more modern techniques (such as identification chips, Bluetooth, etc), in order to automatically carry out the authentication.

10.3 Publications

The work that has been carried out during the master thesis development has been published in the following scientific workshops, conferences and books:

- **Estefanía Serral**, Pedro Valderas, Vicente Pelechano. *Towards the Model Driven Development of Context-Aware Pervasive.* Special Issue of Pervasive and Mobile Computing Journal on Context Modelling, Reasoning and Management. Submitted

- **Estefanía Serral**, Pedro Valderas, Vicente Pelechano. *A Model Driven Development Method for developing Context-Aware Pervasive Systems.* In Springer Berlin / Heidelberg editor. Ubiquitous Intelligence and Computing (UIC-08). Volume 5061/2008 of **Lecture Notes in Computer Science**. pp. 662-676. ISBN 978-3-540-69292-8

- **Estefanía Serral**, Pedro Valderas, Javier Muñoz, and Vicente Pelechano. *Towards a Model Driven Development of Context-aware Systems for AmI Environments*. In

Springer Paris editor, Proceedings of the International Conference on Ambient Intelligence Developments (AmI.d'07)/, pages 114-124, 2007. ISBN/ISSN: 978-2-287-78543-6

- **Estefanía Serral**, Carlos Cetina, Javier Muñoz, and Vicente Pelechano. *PervGT: Herramienta CASE para la Generación Automática de Sistemas Pervasivos*. Tool demostration in XII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2007), Zaragoza (Spain), Sept 2007. ISBN/ISSN: 978-84-9732-595-0

- Carlos Cetina, **Estefanía Serral**, Javier Muñoz, and Vicente Pelechano. Tool Support for Model Driven Development of Pervasive Systems. In 4th International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007), pages 33-41, Los Alamitos, CA, USA, March 2007. ISBN: 0-7695-2769-8. **IEEE Computer Society**.

- Javier Muñoz, Vicente Pelechano, **Estefanía Serral**. *Aplicación del Desarrollo Dirigido por Modelos a los Sistemas Pervasivos: Un Caso de Estudio*. II Congreso IberoAmericano sobre Computación Ubicua (CICU 2006), Alcalá de Henares (Spain), 7-9 June 2006, pp. 171-178, ISBN: 84-8138-703-7

- Javier Muñoz, **Estefanía Serral**, Carlos Cetina and Vicente Pelechano. *Applying a Model-Driven Method to the Development of a Pervasive Meeting Room*. **ERCIM News**, April 2006, vol. 65, pp. 44-45, ISSN: 0926-4981

- Javier Muñoz, Carlos Cetina, **Estefanía Serral**, Vicente Pelechado. *Un Framework basado en OSGi para el Desarrollo de Sistemas Pervasivos*. 9 Workshop Iberoamericano de Ingenieria de Requisitos y Ambientes Software (IDEAS2006), La Plata (Argentina), 24 - 28, Apr 2006 pp. 257 – 270. ISBN-10: 950-34-0360-X

- Javier Muñoz, Vicente Pelechano, **Estefanía Serral**. *Providing platforms for developing pervasive systems with MDA. An OSGi metamodel*. X Jornadas de Ingeniería de Software y Base de Datos (JISBD), Granada (Spain). September 2005, pp. 19 - 26, ISBN: 84-9732-434-X

11. References

[1] Weiser, M. (1991). The Computer for the 21st Century. Scientific American, 265(3):94–104.

[2] Dey, Anind K. (2001). "Understanding and Using Context". *Personal Ubiquitous Computing*

[3] Object Management Group. Model Driven Architecture Guide, 2003.

[4] Strang, T. and Linnhoff-Popien, C. (2004). *A context modeling survey*. In First International Workshop on Advanced Context Modelling, Reasoning And Management, UbiComp 2004.

[5] Biegel, G. and Cahill, V. (2004). "A framework for developing mobile, context-aware applications". In Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communication, pp.361–365

[6] R. De Virgilio, R. Torlone, and G.-J. Houben. *A rule-based approach to content delivery adaptation in web information systems*. In Proc. 7th IEEE/ACM Int. Conf. on Mobile Data Management, page 21, 2006

[7] McFadden, T., Henricksen, K., Indulska, J. (2004). *Automating context-aware application development*. In Jadwiga Indulska & David De Roure (eds) 1st International Workshop on Advanced Context Modelling, Reasoning and Management, in conjunction with the Sixth International Conference on Ubiquitous Computing, Tokyo, Japan, 2004. (pp. 90-95)

[8] Sheng, Q.Z. and Benatallah, B. *ContextUML: a UML-based modelling language for model-driven development of context-aware web services*. Proceedings of the International Conference on Mobile Business (ICMB'05),pp.206–212.

[9] Dhouha Ayed, Didier Delanote, and Yolande Berbers. Mdd approach for the development of context-aware applications. In Boicho Kokinov, Daniel C. Richardson, Thomas R. Roth-Berghofer, and Laure Vieu, editors, Modeling and Using Context - 6th International and Interdisciplinary Conference, CONTEXT'07, Roskilde, Denmark, Lecture Notes in Computer Science 4635, pages 15-28. Springer-Verlag Berlin Heidelberg, 2007.

[10] Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., and Altmann, J. (2002). Context-awareness on mobile devices – the hydrogen approach. In Proceedings of the 36th Annual Hawaii International Conference on System Sciences, pages 292–302.

[11] Fahy, P. and Clarke, S. (2004). "CASS – a middleware for mobile context-aware applications". In Workshop on Context-awareness, MobiSys 2004.

[12] Chen, H., Finin, T., and Joshi, A. (2004). "An ontology for context-aware pervasive computing environments". Special Issue on Ontologies for Distributed Systems, Knowledge Engineering Review, 18(3):197–207.

[13] H. Chen, T. Finin, and A. Joshi. A context broker for building smart meeting rooms. In *Proceedings of the Knowledge Representation and Ontology for Autonomous Systems Symposium, 2004 AAAI Spring Symposium.*

[14] T. Gu, H. K. Pung, and D. Q. Zhang. A service-oriented middleware for building context-aware services. Journal of Network and Computer Applications, 28(1):1–18, 2005

[15] D. Preuveneers, J. V. den Bergh, D. Wagelaar, A. Georges, P. Rigole, T. Clerckx, Y. Berbers, K. Coninx, V. Jonckers, and K. D. Bosschere. Towards an extensible context ontology for ambient intelligence. In Proc. 2nd European Symp. Ambient Intelligence, LNCS 3295, pages 148–159, 2004

[16] S. J. H. Yang, A. Huang, R. Chen, S.-S. Tseng, and Y.-S. Shen. Context model and context acquisition for ubiquitous content access in ulearning environments. In IEEE Int. Conf. Sensor Networks, Ubiquitous, and Trustworthy Computing, volume 2, pages 78–83, 2006

[17] Korpipää, P., Mäntyjärvi, J., Kela, J., Keränen, H., and Malm, E.-J. (2003). "Managing context information in mobile devices". IEEE Pervasive Computing.

[18] Roman, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R. H., and Nahrstedt, K. (2002). *A middleware infrastructure for active spaces*. IEEE Pervasive Computing. ->Gaia

[19] S. Buchholz, T. Hamann, and G. Hübsch. *Comprehensive structured context profiles (CSCP): Design and experiences*. In Proc. 2nd IEEE Conf. on Pervasive Computing and Communications Workshops, pages 43–47, 2004

[20] M. Strimpakou, I. Roussaki, and M. E. Anagnostou. *A context ontology for pervasive service provision*. In 20th Int. Conf. on Advanced Information Networking and Applications, pages 775–779, 2006

[21] Vallée M, Ramparany F, Vercouter L. *A multi-agent system for dynamic service composition in ambient intelligence environments*. Pervasive 2005

[22] Diane J. Cook, G. Michael Youngblood, Edwin O. Heierman III, Karthik Gopalratnam, Sira Rao, Andrey, Litvin, Farhan Khawaja. *MavHome: An Agent-Based Smart Home*. PerCom 2003

[23] M. Baldauf, S. Dustdar, F. Rosenberg. *A survey on context aware systems*. International Journal of Ad Hoc and Ubiquitous Computing, forthcoming.

[24] Javier Muñoz and Vicente Pelechano. *Building a Software Factory for Pervasive Systems Development*. In CAiSE 2005, Porto, Portugal, June 13-17, volume 3520 of LNCS, pages 329–343, May 2005.

[25] March S and Smith G. *Design and Natural Science. Research on Information Technology*. Decision Support Systems 15, 251 - 266. DOI: 10.1016/0167-9236(94)00041-2. 1995.

[26] Vaishnavi, V. and Kuechler, W. 2004. *Design Research in Information Systems.*

[27] Friedemann Mattern. *Ubiquitous Computing: Scenarios from an informatised world*, pages 145_163. Springer-Verlag, 2005

[28] Jochen Burkhardt, Thomas Schaeck, Horst Henn, Stefan Hepper, and Klaus Rindtor. *Pervasive Computing: Technology and Architecture of Mobile Internet Applications*. Addison-Wesley, April 2002

[29] Uwe Hansmann, Lothar Merk, Martin S. Nicklous, and Thomas Stober. Pervasive *Computing Handbook*. Springer-Verlag, 2001

[30] David Wright, Elena Vildjiounaite, Ioannis Maghiros, Michael Friedewald, Michiel Verlinden, Petteri Alahuhta, Sabine Delaitre, Serge Gutwirth, Wim Schreurs, and Yves Punie. Safeguards in a world of ambient intelligence (swami) deliverable d1. *The brave new world of ambient intelligence: A state-of-the-art review*, June 2005. A report of the SWAMI consortium to the European Commission under contract 006507

[31] Lieberman, H. & Selker, T. (2000). *Out of Context: Computer Systems That Adapt To, and Learn From, Context*. IBM Systems Journal, **39**(3&4), 617-631.

[32] Lenat, D. (1998) *The Dimensions of Context Space.* Technical report, CYCorp, October 1998. Invited talk at the conference Context 99. Retrieved 18 July, 2008 from http://www.cyc.com/doc/context-space.pdf.

[33] Stalling, W. (2000) *Operating Systems: Internals and Design Principles*. Prentice Hall.

[34] Schilit, W. N., Adams, N. I. & Want, R. (1994). *Context-aware Computing Applications*. In Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, Santa Cruz, CA, USA, 1994. (pp. 85-90). Washington, DC: IEEE Computer Society.

[35] Ryan, N.S., Pascoe, J., Morse, D. R. (1998). *Enhanced Reality Fieldwork: the Context-aware Archaeological Assisstant*. In V. Gaffney, M. van Leusen & S. Exxon (eds.) Computer Applications in Archaeology, British Archaeological Reports, Oxford, October 1998. Tempus Reparatum. Retrieved 18 July, 2008 from http://www.cs.kent.ac.uk/pubs/1998/616/content.html

[36] Mitchell, K. (2002). *Supporting the Development of Mobile Context-Aware Computing*. Ph.D. Thesis, Department of Computing, Lancaster University, Lancaster, UK.

[37] Crowley, J. L., Coutaz, J., Rey, G., Reignier, P. (2002). *Perceptual Components for Context Aware Computing*. In Proceedings of the International Conference on Ubiquitous Computing, Goteborg, Sweden, September 2002. (pp. 117-134). London: Springer Verlag.

[38] Bardram, J. E. (2005). *The Java context awareness framework (JCAF) - a service infrastructure and programming framework for context-aware applications*. In Proceedings of the Third International Conference on Pervasive Computing, Munich, Germany, 2005. (pp 98–115). London: Springer Verlag.

[39] OMG UML 2.0 Specifications, http://www.uml.org/

[40] Jack Greenfield, Keith Short, Steve Cook, and Stuart Kent. *Software Factories*. Wiley Publishing Inc., 2004.

[41] http://msdn2.microsoft.com/en-us/vstudio/aa718368.aspx

[42] Gruber, T. R. (1993). *A Translation Approach to Portable Ontology Specifications*. Knowledge Acquisition.

[43] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P.F. Patel-Schneider, L. A. Stein. *Web Ontology Language (OWL) W3C Reference version 1.0*, 18 August 2003. At http://www.w3.org/TR/2002/WD-owl-ref-20021112.

[44] Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet S., Grosof, B. & Dean, M. (2004). *SWRL: A Semantic Web Rule Language Combining OWL and RuleML.* Retrieved 18 July, 2008 from http://www.w3.org/Submission/SWRL/

[45] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur and Yarden Katz. *Pellet: A practical OWL-DL reasoner*, Journal of Web Semantics, 5(2), 2007.

[46] www.eclipse.org

[47] http://www.osgi.org/

[48] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1994.

[49] AL-Muhammed, M., Embley, D.W., and Liddle, S. *Conceptual Model Based Semantic Web Services*. In ER 2005, volume 3716 of LNCS. Springer.

[50] http://www.sts.tu-harburg.de/~r.f.moeller/racer/

[51] http://sourceforge.net/projects/owlapi

[52] Maria Ebling, Guerney Hunt, and Hui Lei. *Issues for Context Services for Pervasive Computing*. In Proceedings of Middleware'01, Advanced Workshop on Middleware for Mobile Computing, Heidelberg, Germany, November 2001

[53] Byun, H., & Cheverst, K. (2004). *Utilizing Context History to Provide Dynamic Adaptations*. Journal on Applied Artificial Intelligence **18**(6), 533-548.

[54] Verpoorten, K. & Karin-Coninx, K.L. (2007). *Mixed Initiative Ambient Environments: A Self-Learning System to Support User Tasks in Interactive Environments*. In Proceedings of Third Workshop on Context Awareness for Proactive Systems, Guildford, United Kingdom, June 2007. Retrieved 18 July, 2008 from http://owlapi.sourceforge.net/publications.html

[55] http://oomethod.dsic.upv.es/labs/

[56] Javier Muñoz, Idoia Ruiz, Vicente Pelechano, and Carlos Cetina. *Un framework para la simulación de sistemas pervasivos*. In Simposio sobre Computación Ubicua e Inteligencia Ambiental (UCAmI'05), pages 181-190, Granada, Spain, September 2005.

[57] Estefanía Serral, Pedro Valderas, Javier Muñoz, and Vicente Pelechano. *Towards a Model Driven Development of Context-aware Systems for AmI Environments*. In Springer Paris editor, Proceedings of the International Conference on Ambient Intelligence Developments (AmI.d'07)/, pages 114-124, 2007. ISBN/ISSN: 978-2-287-78543-6