Loughborough University Institutional Repository

Semantic interoperability in ad-hoc computing environments

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: https://dspace.lboro.ac.uk/2134/3072

Please cite the published version.

This item was submitted to Loughborough's Institutional Repository by the author and is made available under the following Creative Commons Licence conditions.

COMMONS DEED

Attribution-NonCommercial-NoDerivs 2.5

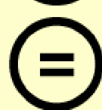
You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.



No Derivative Works. You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Legal Code (the full license).

Disclaimer

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/

Semantic Interoperability in ad-hoc Computing Environments

by

José Ignacio Rendo Fernández

A Doctoral Thesis

Submitted in partial fulfilment
of the requirements for the award of

Doctor of Philosophy

of

Loughborough University

June 2007

© José Ignacio Rendo Fernández, 2007

This thesis is dedicated to
María Paz and José, my parents.

Acknowledgements

First, I would like to thank Loughborough University, and specially the Computer Science department for placing their trust in me.

To Dr. I.W. Phillips, my most sincere gratitude for his supervision, wise advise, support, friendship and help provided during this there years in Loughborough.

I would like to thank my parents, José and María Paz, and my brother Ángel, who always are with me, even they live 2000 miles from Loughborough.

Infinite gratitude to the owner of my heart, my beloved Isabel. Her love, support and advice give me the will and the strength to keep going. Also, my gratitude to her parents, Carlos and Felisa, who make me feel like their third son.

Once again, I always be grateful to all Computer Science department staff and students for all the assistance given during my PhD. Specially, I am very grateful to Jose A. Hernandez who wisely advised me to start this PhD and John Whitley, for the ubiquitous help given during these years.

Finally, thanks to all friends who have share with me these three years.

Abstract

This thesis introduces a novel approach in which multiple heterogeneous devices collaborate to provide useful applications in an ad-hoc network.

This thesis proposes a smart home as a particular ubiquitous computing scenario considering all the requirements given by the literature for succeed in this kind of systems. To that end, we envision a horizontally integrated smart home built up from independent components that provide services. These components are described with enough syntactic, semantic and pragmatic knowledge to accomplish spontaneous collaboration. The objective of these collaboration is domestic use, that is, the provision of valuable services for home residents capable of supporting users in their daily activities. Moreover, for the system to be attractive for potential customers, it should offer high levels of trust and reliability, all of them not at an excessive price.

To achieve this goal, this thesis proposes to study the synergies available when an ontological description of home device functionality is paired with a formal method. We propose an ad-hoc home network in which components are home devices modelled as processes represented as semantic services by means of the Web Service Ontology (OWL-S). In addition, such services are specified, verified and implemented by means of the Communicating Sequential Processes (CSP), a process algebra for describing concurrent systems.

The utilisation of an ontology brings the desired levels of knowledge for a system to compose services in a ad-hoc environment. Services are composed by a goal based system in order to satisfy user needs. Such system is capable of understaning, both service representations and user context information. Furthermore, the inclusion of a formal method contributes with additional semantics to check that such compositions will be correctly implemented and executed, achieving the levels of reliability and costs reduction (costs derived form the design, development

and implementation of the system) needed for a smart home to succeed.

Keywords: Ubiquitous computing, ad-hoc, smart home, OWL-S, CSP, ontology, formal method

Contents

Acknowledgements **ii**

Abstract **iii**

1 Introduction **1**

1.1 Motivation 1

1.1.1 Scope, Aims and Method 1

1.2 Ubiquitous Computing 2

1.3 Smart Home 3

1.4 Smart Home Requirements 5

1.5 Proposal 7

1.6 Thesis Structure 9

2 Methodology **10**

2.1 Literature Review 10

2.2 Ontology and Formal Method Criteria 11

2.2.1 Ontology 11

2.2.2 Formal Method 13

2.3 Alternatives 15

2.4 Election 15

3 State of the Art Analysis **17**

3.1 Comparative Framework 19

3.1.1 Horizontal Integration 19

3.1.2 Spontaneous Collaboration 22

3.1.3 Design for Domestic Use 24

3.1.4 Reliability 25

3.1.5 Cost 25

4 Smart Home Proposal 27

4.1 Ontologies and Formal Methods 27

4.2 Smart Home Proposal 29

5 Semantic Home Device Services 31

5.1 Web Service Ontology (OWL-S) 31

5.2 CONON: The Context Ontology 33

5.3 Service Representation 33

5.3.1 Static Service Representation 34

5.3.2 Dynamic Service Representation 36

6 Formal Specification of Home Appliance Services 42

6.1 Communicating Sequential Processes (CSP) 42

6.2 CSP Translation 44

6.2.1 Static Representation 44

6.2.2 Dynamic Representation 48

6.2.3 Service Refinement 50

7 Implementations of Home Appliance Services 54

7.1 CSP implementations 54

7.1.1 occam 54

7.1.2 Java CSP (JCSP) 55

7.1.3 Communicating Threads for Java (CTJ) 55

7.1.4 C++ CSP 56

7.1.5 Handle C 56

7.2 CSP Service Implementation 56

7.2.1 JCSP Service Implementation 57

7.2.2 Channel Implementation 60

7.3 Service Grounding 62

7.3.1 UDP Service Grounding 62

8 Examples of Home Appliance Services 64

8.1 Simulator 64

8.2 Orchestration Node 67

8.2.1 Service Composition 68

8.3 Examples of Home Device Services 71

8.3.1 HVAC System 71

8.3.2 AV System 80

8.3.3 Communication System 84
8.3.4 More Examples. Ac-Hoc Composition 85

9 Conclusions, Contributions, and Future Work 89
9.1 Conclusions 89
9.2 Contributions 91
9.3 Future Work 92

Bibliography 97

List of Figures

1.1 Smart Home Proposal 8

4.1 Ontologies and Formal Method Relation in an Smart Home Project 28
4.2 Smart Home Proposal 29

5.1 OWL-S Class Diagram 32
5.2 CONON Upper Ontology 33
5.3 LufService Class Diagram 34
5.4 LufProcess Class Diagram 35
5.5 Data Communication 35
5.6 Washing Machine Service 36
5.7 Activity Class Diagram 37
5.8 Activity Effects Class Diagram 38
5.9 Types of Activities 38
5.10 Home Automation Activity 39
5.11 Activity Diagram 40
5.12 *WashingLaundry* Activity Diagram 40

6.1 Static Representation of a LufService 45
6.2 EventWashingStatus Diagram 47
6.3 CSP Activity Process Diagram 48

7.1 Relation between Services and Devices 57
7.2 UDP Channel Implementation 60
7.3 LufGrounding Class 63

8.1 Simulator 65
8.2 Washing Machine Interface 66

8.3 Simulator and Central Node Connection 66
8.4 Central Node Process Diagram 67
8.5 StartWashing Effect 69
8.6 Activity associated with measuring the livingroom temperature . . 73
8.7 Home Temperature Evolution 80
8.8 Activities Related with the DVD Service 82
8.9 Simulation of the AV System 83
8.10 Simulation of the Communication System 86

List of Tables

1.1 Research challenges in ubiquitous computing 3
1.2 Smart Home Drawbacks 4
1.3 User Concerns Toward a Smart Home 5
1.4 Smart Home Requirements 6
1.5 Relations 7

2.1 Ontologies and Requirements 12
2.2 Formal Methods and Requirements 14

3.1 Ubiquitous Computing Technologies 20
3.2 Ubiquitous Computing Technologies based on the Semantic Web . . 21
3.3 Smart Home Technologies 22

6.1 CSP_M Operators 43

8.1 Heat Room Activity Triggers 72
8.2 Heat Pump Evolution 80
8.3 State Diagram for Activity WatchingVideo 81

Listings

7.1 Washing Machine service declaration in JCSP 58
7.2 Washing Machine service run method 59
7.3 Washing Machime instantation 61

Chapter 1

Introduction

1.1 Motivation

1.1.1 Scope, Aims and Method

In Latin, ad hoc literally means ”for this,” further meaning ”for this purpose only, ” and thus usually temporary. In computing science, an ad-hoc (or ”spontaneous”) network is a network in which some of the nodes are part of the network only for the duration of a communications session or, in the case of mobile or portable devices, while in some close proximity to the rest of the network.

The aim of this thesis is to enhance communication between components in an ad-hoc computing environment in order to achieve interoperability between them. To that end, this thesis present a rich semantic description of components in order to understand what function each component performs.

In particular, the focus is on achieving automatic collaboration between home devices for helping inhabitants in their daily activities. This kind of home is known as a “smart home”. For example, imagine an householder watching one of their favorite DVD movies in a darkened lounge. After a while, he decides to go bed. Just after the user has left the lounge, some kind of intelligent system acting on the user's behalf pauses the DVD and switch off the lounge lights. Once the user is in his room, he changes his mind and decides to continue watching the movie in his laptop screen. To that purpose, he turns on his laptop. The system notices that a device capable of rendering the DVD movie, that is the laptop, has joined the home network. Automatically the system offers to the user the service of continue watching the movie on the laptop screen, so the user can continue watching the movie in his bedroom, only by switching on his laptop.

The proposed semantic description for home devices is based on a set of re-

quirements for a smart home to succeed. Considering a smart home as a subcase of an ubiquitous computing environment, the requirements for the semantic description are derived from the research challenges for ubiquitous computing, users' concerns toward a smart home and the requirements for smart home found in the literature.

The methods used include systematically surveying literature and available information, designing and implementing prototypes to prove the feasibility of the proposed ideas, creating models and concepts that generalise the prototypes, and evaluation of the proposed solutions.

Next sections on this chapter discuss about the ubiquitous computing and the smart home concepts. A brief introduction of the semantic description of devices proposed in this thesis is presented afterwards. Finally, last section describes the structure of this document.

1.2 Ubiquitous Computing

One of the hottest current topics in Computer Science is ubiquitous or pervasive computing. One justification of this assertion stems from the number of terms related to this topic, such as "invisible computing", "always connected" or "natural interfaces". There is no lack of authors pointing ubiquitous computing as a revolution for industries and a new source of income through the generation of new value added services [73, 38, 65]. It is not strange that the governments of the most powerful economies of the world have included in their agendas the development of the ubiquitous computing paradigm. For example, in 2003 the Japanese government has encouraged the U-Japan as the IT strategy for revitalising the Japanese economy through the use of ubiquitous computing technologies [78].

The ubiquitous term comes from the Latin and means "everywhere". In the IT world, the term started to be popular in the United States in 1988. The concept is also known as "pervasive computing", "ambient intelligence" or "ubiquitous networking". Ubiquitous computing central is invisibility, meaning embedding computation into the environment and everyday objects would enable people to interact with information-processing devices more naturally and casually than they currently do [104]. To achieve the concept of ubiquitous computing, researchers on this topic have found the challengues listed on table 1.1 [47, 64, 36].

Challenge	Description
Seamless component interaction	It is widely accepted that for an ubiquitous system to succeed it needs to be build up of independent components capable of interacting between each other
Adaption to changes in the environment	Typically, ubiquitous computing applications involve multiple mobile and heterogeneous devices. Ideally, an ubiquitous computing application should not be affected by alterations in its context, such for example, a change or failure in the resources needed for the application
Task analysis and component interaction	Components should be combined in order to help users in their daily activities. Therefore, an automatic recognition of such activities it is needed to bring valuable applications to the user
Generation of effective business models	To ensure revenue when deploying and operating ubiquitous computing applications
Provision of trust and confidentiality to end users	Ubiquitous computing applications must be robust and reliable. Besides, there needs to be a balance between security and privacy

Table 1.1: Research challenges in ubiquitous computing

This thesis is motivated by the creation of ubiquitous computing-based applications for helping householders in their daily activities. This scenario, known as "smart home", has risen the interest of not only electrical and consumer electronics industries, but also service providers.

Before presenting in more detail the smart home concept, we can introduce some of the solutions proposed by industry to pursue this objective. On one side, there exist technologies directly designed for home automation, such as X-10, CeBus, HAVi or HES. On the other side, services architectures protocols such as web services, UPnP or Jini have been proposed as an uniform way of accessing home appliance functionality. All of these standards are designed to provide a common access for home appliances in order to facilitate home automation. In addition, the combination of these standards with the arriving of Internet access and wireless networks will enhance the ad-hoc nature of home networks. However, none of these technologies on their own are enough to offer a complete ubiquitous computing scenario for home residents.

1.3 Smart Home

A "smart home" can be defined as: "A residence equipped with computing and information technology, which anticipates and responds to the needs of the occu-

pants, working to promote their comfort, convenience, security and entertainment through the management of technology within the home and connections to the world beyond" [33].

Although a great number of companies and academic institutions have tried to implement the smart home concept, none of them have completely succeed. Table 1.2 summarises some reasons for such failure [33].

Drawback	Description
Lack of common protocol	Typically, smart homes are implemented with proprietary technology, which raises interoperability issues
High initial investment form the consumer	At present, home automation suppliers do not offer enough value for potential customers because current solutions do not satisfy user needs
Technology push by suppliers	It seems that smart home providers do not consider user aims. In fact, customers are not attracted by the products marketed by suppliers
Little usability evaluation by suppliers	It seems that suppliers do not put much attention to the evaluation of the usability of smart home products

Table 1.2: Smart Home Drawbacks

Some of these drawbacks start to be overcame. Advances in microelectronics and communications have reduced the initial set-up and purchase costs. However, the last two obstacles require more effort from suppliers, considering not only technological aspects, but also social. The nature of a home life is complex, involving people with different ages, concerns, tastes and skills. The study of such a complex environment will help smart home designers to understand what user needs are [82]. Accordingly, Loughborough University has performed a study of what the requirements and expectations toward smart homes are [54]. Generalising the results of this study, table 1.3 plots the main concerns of people about smart homes.

User Concern	Description
Cost	This issue includes worries about the excessive expenses of initial investment and maintenance of the system
Reliability	In most of the cases, users relate smart homes with IT, and hence with previous bad experiences with computers
Security, privacy and safety	In this area, personal physical security and data protection are the most important users' worries
Flexibility	Smart homes should be adaptable enough to adjust to different user preferences. For example, some people do not accept for the smart home to anticipate their intentions while others do
Convenience	In relation with the previous topic, it seems desirable a system capable of helping home residents when needed
Maintaining independence and keeping active	It is widely accepted that a smart home should not completely remove home residents from the household control
Future proof technology	People consider that a smart home should be a lasting product

Table 1.3: User Concerns Toward a Smart Home

From the previous discussion, it is possible to propose that the reasons for smart home poor market acceptance and the expectations towards and smart home can be tackled from the research challenges for ubiquitous computing listed on table 1.1. Next section goes deeper in this assertion.

1.4 Smart Home Requirements

The world of smart homes is populated with different alternatives, which share a poor market acceptance. This failure has been justified because of the lack of attention of smart home suppliers to user requirements. Under this situation, the objective of this thesis is to propose a semantic description of home devices based on the discussion performed in the two previous sections. Accordingly, table 1.4 plots several requirements to achieve the aim of this thesis [48]:

Requirement	Description
Horizontal integration	The proposed system should be built up of independent components. These components should be capable of interacting between them
Spontaneous collaboration	Component collaboration should be accomplished with little planning in advance. As a consequence, it is needed an agreement in the syntax and semantics of the functionality offered by each component
Design for domestic use	One of the aims of the proposed smart home is to support home residents in their daily activities. To understand these activities, the proposed smart home should be capable of acquiring and managing the information related to the context of the house occupants such as location, time or the set of home appliances used. In connection to the previous objective, not only a syntactical and semantic agreement between components is needed, but also pragmatic, in order to distinguish in which situations a component might be useful or not
Reliability	One of the common characteristics of home appliances and devices is their reliability. It is not surprising that a smart home would raise the same expectation in potential customers. For this reason, one objective is to investigate how these devices are designed to apply the same procedures to a smart home
Low cost	This objective focus in reducing the cost in all the stages involved from the development to the final deployment of the smart home. Particular interest will be taken to component design and implementation. This objective should not compromise the previous ones

Table 1.4: Smart Home Requirements

The requirements for a smart home described in table 1.4 are closely related with research challenges in ubiquitous computing, the reasons for smart home failures and the expectations of user's towards a smart home described in the previous sections. Accordingly, table 1.5 plots this relation.

Requirement	Challenge	Drawback	Concern
Horizontal integration	* Seamless component interaction	* Lack of common protocol	* Easy of use * Convenience * Flexibility
Spontaneous collaboration	* Adaption to changes in the environment	* Lack of common protocol	* Easy of use * Convenience * Flexibility
Design for domestic use	* Task analysis and component association	* Little usability evaluation by suppliers * Technology push by suppliers	*Easy of use * Convenience * Flexibility
Reliability	* Provision of trust and confidentiality to the user	* Little usability evaluation by suppliers	* Reliability * Security, privacy and safety * Future proof of technology
Cost	* Generation of effective business models	* High initial investment from the consumer * Technology push by suppliers	* Cost

Table 1.5: Relations

In order to achieve the semantic description of devices following the requirements of table 1.5, a deep study of the current technologies for ubiquitous computing and smart homes should be carried out. This analysis will serve as a way to acquire the needed knowledge to understand how a smart home topic should be tackled. With this information, the next objective is the development a framework for a smart home. Once the framework is defined, the proposal is validated with the implementation of a prototype. Finally, the last objective of this thesis is to analyse the work and extract some conclusions.

1.5 Proposal

This work proposes a novel model in which home devices functionality is offered as services modelled by a set of embedded processes, which are composed in a dataflow network. These processes are described by an ontology which is subsequently verified and implemented by means of a formal method. The principal aim and contribution of this approach is to develop a description of home devices which addresses the set of requirements listed on table 1.4.

The semantics are given by the Semantic Markup for Web Services (OWL-S) [17], an upper ontology for describing web services and their composition. In

OWL-S, services are modeled as atomic or composite processes. Both of these specify a set of inputs with preconditions, and a set of outputs with effects. The functionality of a service is encapsulated in a profile (class ServiceProfile) while its operation, that is, how it works, is described by the process model (class ServiceModel). The implementation details are wrapped in a service grounding declaration (class ServiceDeclaration).

The chosen formal method is (Communicating Sequential Processes) CSP [62, 94]. It is a notation for describing concurrent systems whose component processes interact with each other by communication. Our aim is to find a procedure that, given a service description in the OWL-S service model, finds the most suitable implementation for the service.

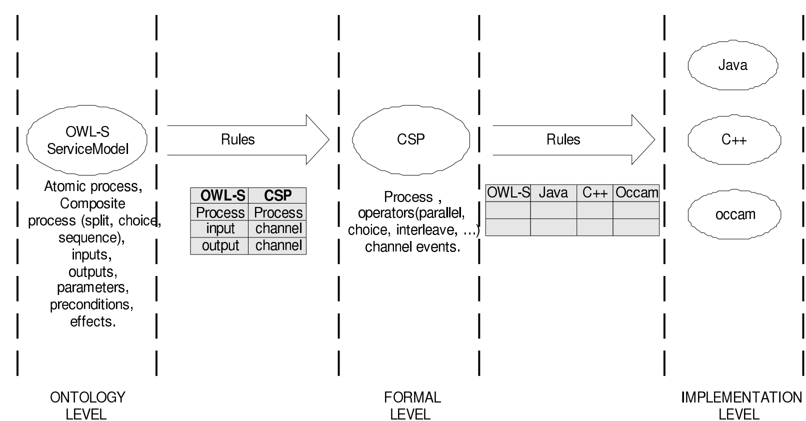


Figure 1.1: Smart Home Proposal

As shown in figure 4.2 , firstly, the service model description is translated to CSP, which gives a formal view of the service. At this level, it is possible to check the correctness of the service in terms of deadlocks and other topics related with concurrent systems. Once the service is proved safe, with the use of predefined procedures it is possible to obtain the desired implementation.

The objective of this proposal is to achieve a semantic description of devices meeting the requirements for a smart home plotted on table 1.4. Firstly, the use of a service oriented architecture bring the desired modularity in order to achieve the horizontal integration. Secondly, the ontological nature of OWL-S gives the desired semantics for achieving spontaneous collaboration and opens the door for a design for domestic use. Finally, the adoption of a formal method gives high levels of reliability and the opportunity of achieve cheap hardware implementations.

1.6 Thesis Structure

The present document has been divided in nine chapters. To clarify the reading of them, this section introduces some comments.

The present chapter, **Chapter 1**, is intended to show the motivation and the objectives of this thesis.

In **Chapter 2**, the research methods used in this thesis are introduced. The focus of this chapter is on the criteria for performing the literature review and the reasons for choosin OWL-S and CSP.

Chapter 2 involves the starting point for this thesis. It introduces the current technologies for ubiquitous computing and smart homes. The chapter ranges from the early trials to achieve home automation to the latest technologies for ubiquitous computing. The analysis is based on how the technologies tackle the requirements presented in Chapter 1. For example, it is analysed how the horizontal integration is achieved determining the network protocols involved.

A proposal for a smart home is introduced in **Chapter 4** based on the information gathered in Chapter 3.

In **Chapter 5**, the proposal introduced in Chapter 4 is extended. Chapter 5 focus principally on how to achieve the objectives 1 and 2 and opens the door of how to accomplish objectives 2 and 5 by means of the utilisation of OWL-S.

Chapter 6 and **Chapter 7** continue extending the model introduced in Chapter 4. The aim of these two chapters is the presentation of how to succeed in objectives 2 and 5 with the utilisation of CSP.

In **Chapter 8**, a simulator of a smart home is presented as a proof of concept. Several subsystems inside a home such a HVAC control and AV system are modelled with the proposal introduced in chapters 4, 5,6 and 7.

Finally, **Chapter 9** resumes all the conclusions extracted in this thesis and defines the future lines of research to improve this work.

Chapter 2

Methodology

This chapter shows the basis of the decisions adopted in this thesis. The steps taken are:

- Literature review
- As the proposal includes de combination of an ontology with a formal method:

  Inclusion/exclusion criteria for choosing the ontology

  Inclusion/exclusion criteria for choosing the formal method

2.1 Literature Review

The objective of the literature review is to acquire a broad knowledge about the topics of this thesis, those are, ubiquitous computing and smart home. The main focus is on studying how the most popular projects for smart home and ubiquitous computing solve the requirements pointed in chapter 1.

In order to carry out the literature review, firstly, classical home automation standards such as X10 and CEbus are studied. Typically, these approaches focus on achieving horizontal integration through the definition of well established interfaces between home devices, opening the door to the achievement of the second requirement, that is, the accomplishment of seamless component interaction. However, it is possible to see how these standards have a poor market acceptance, maybe, because of the lack of attention of suppliers to user needs.

After studying home automation standards, the second step in the literature review focus on projects from academia related to achieve a smart home. The main difference with home automation standards is that in this case, there exists some kind of intelligence acting on users' behalf in order to help householders in

their daily activities. These approaches mainly focus on meeting the requirement relative to the design for domestic use.

The next step in the literature review points to the study of ubiquitous computing projects, regardless to their main deployment environment (office, home ...). This area of the literature review is especially important because it introduces the use of ontologies and formal methods. Especially interesting are those projects which propose the use of the semantic web in ubiquitous computing projects. Next sections in this chapter introduces the basis followed in order to combine a formal method and an ontology for achieving the aim of this thesis.

2.2 Ontology and Formal Method Criteria

In order to accomplish the aim of this thesis, the achievement of a rich semantic description of components based on the requirements for a smart home introduced in chapter 1, we propose the combination of an ontology and a formal method. To facilitate this integration, we propose a framework in which several ontologies and formal methods are studied independently, regarding of how they resolve the requirements of chapter 1.

The best solution for our approach will be the combination of an ontology with a formal method which best meets the requirements for a smart home with less difficulties. For example, OWL-S solves the requirement of horizontal integration with the representation of devices functionality as services which can be modeled as processes. CSP is based on the composition of processes. OWL-S and CPS has the common approach of modeling components as processes in order to meet the requirement of horizontal integration, which facilitates their integration.

In order to find the best combination of a formal method and an ontology, next sections introduces some ontologies and formal methods.

2.2.1 Ontology

To achieve semantic interoperability in ad-hoc environments, as the title of this thesis says, some kind of agreement between components is needed. Accordingly, research in ubiquitous computing points to the use of ontologies as the way of achieving semantic interoperability [60, 36].

An ontology can be considered as an agreed terminology by providing concepts, and relationships between the concepts. In this work, an especial interest is taken on ontologies which address automatic web service composition because of the similarity of this challenge with the aim of this thesis. As it is shown in table 2.1, the following ontologies are studied:

	ONTOLOGY		
Requirement	**OWL-S**	**WSMO**	**GAS**
Horizontal integration	Services, processes	Services	Gadgets
Spontaneous collaboration	Profiles(inputs, outputs, effects, preconditions), composite processes	Goals, mediators, orcherstation, choreography	Synapses, plugs, eGadgetsWorld
Design for domestic use	Enhanced with other ontologies	Use of upper ontologies	Use of upper ontologies
Reliability	N/A	N/A	N/A
Cost	Use of standards for the semantic web (OWL)	Use of standards for the semantic web (OWL)	N/A

Table 2.1: Ontologies and Requirements

- **OWL-S (Ontology for Web Services)** [17]: Is an ontology based on OWL to describe web services. OWL-S leverages on OWL to support capability based discovery of Web services, automatic composition of Web Services and Support automatic invocation of Web services

- **WSMO (Web Service Modeling Ontology)** [30]: Is an ontology and conceptual framework to describe Web services and related aspects. It is based on the Web Service Modeling Framework (WSMF).

- **GAS (Gatgetware Architectural Style) Ontology** [43]: It describes the hierarchy of basic concepts and the local capabilities and experience of GAS components (eGadgets). It is encoded in XML and contains a set of basic terms that are understandable by all eGatgets and a mechanism to translate local definitions using the basic terms.

The first two requirements for a smart home (**horizontal integration** and **spontaneous collaboration**) are related with the capabilities of the ontologies plotted in table 2.1 to achieve automatic composition.

Horizontal integration is addressed through the decomposition of systems in basic building blocks: services in OWL-S and WSMO and eGadgets in GAS. These building blocks represent an abstraction of the functionality offered by a device.

The semantics enclosed in each component (services or eGadgets) is the key factor to achieve spontaneous collaboration. In OWL-S, the functional description of the service is expressed in terms of the transformation produced by the service. It specifies the inputs required by the service and the outputs generated; furthermore, since a service may require external conditions to be satisfied, and it has the effect of changing such conditions, the profile describes the preconditions

required by the service and the expected effects that result from the execution of the service. With this information, services are composed to achieve more complex ones connecting processes with compatible input/outputs or effect/results. This combination of services use primitives imported from formal methods, such as two processes running in parallel or in sequence.

In WSMO, service functionality is expressed in terms of the goals expected from users' towards services and the capabilities of each service. With this information, services are composed as a set of if - then -else structures (choreography), or using a goal based approach (orchestration).

In GAS, service functionality is exposed through plugs, a kind of input/output so composition is achieved through the combination on compatible plugs.

In all the cases, the **design for domestic** use is addressed with ontologies which represent the deployment environment. The solution for this requirement is immediate in WSMO and GAS, which offers upper ontologies in their respective frameworks to that purpose. However, OWL-S on its own does not satisfy this requirement, making necessary the inclusion on external ontologies.

Finally, **cost** reduction is addressed with the utilization of standards, like in OWL-S and WSMO.

2.2.2 Formal Method

The following section analyzes several formal methods focusing on how the meet they requirements for a smart home. Because there are a great amount of formal methods, it would be very difficult to analyse all of them. For this reason, in this section only the following four methods are considered:

- **CSP (Communicating Sequential Processes)** [62], because is a process algebra for the specification of concurrent systems capable of being implemented in hardware or software. In fact, process algebras have been recommended as one of the best approaches for web service composition, a challenge very similar to the aim of this thesis.
- **Ambient Calculi** [41], because is a process algebra specially designed for the specification of mobile applications.
- **Z** [34], because is a very popular formal method.
- **Petri Nets** [84], because is a very popular family of formal methods that have used before in the specification of web service composition.

Table 2.2 introduces how each one of the introduced formal methods meets each one of the requirements for a smart home.

	FORMAL METHOD			
Requirement	**CSP**	**A. Calculi**	**Z**	**Petri Nets**
Horizontal integration	Processes	Processes	Schemas	Places, transitions, arcs
Spontaneous collaboration	Channels, primitives	channels, primitives	Operations, constrains	Transitions
Design for domestic use	N/A	N/A	N/A	N/A
Reliability	Model checker	Model checker	Type and constrain checker	Model checker
Cost	Hardware/ Software implementation	Agents	No code generation	Hardware/ Software implementation

Table 2.2: Formal Methods and Requirements

Each one of the introduced formal methods specifies systems through the combination of building blocks. In our analysis, these building blocks determine how each formal method meets the requirement of **Horizontal Integration**. For example, CSP and Ambient calculi use processes as the essential building blocks, while Z uses schemas, a kind of data structure with operations.

How the building blocks communicate or interact between them is how the **Spontaneous Collaboration** requirement is met. Accordingly, while in CSP processes interact between them by means of primitives such as "Parallel or Sequential" and special sinchronisation structures called channels, in Z schemas interact through operations, similarly as in the Object Oriented paradigm.

The formality inherent to each one of the methods ensures some level of **Reliability**. In this thesis, it is considered the way in which it is possible to ensure that a specification satisfies some criteria. For example, in CSP and Ambient Calculi by means of model checker it is possible to proof in the development stage if a system implementation is going to behave in the same way as an specification. For example, imagine a home equiped with two systems: one responsible for ensuring the environmental securty keeping all windows and other system capable of controlling the home indoor temperature managing windows and blinds. In a summer night, the temperature control system will open the windows, while the security system will close them. This will end up in an infinite loop in which the windows are constantly opening and closing. The power of CSP and Ambient Calculi is that behaviour can be detected in advance, and hence corrected in the design stage.

2.3 Alternatives

This section introduces some possible alternatives for the combination of an ontology and a formal method. As it was said before, the best combination will be that one in which the formal method and the ontology coincide in the way of meeting all the requirements for a smart home. However, this solution will be very difficult to achieve.

In our study, the combination of OWL-S with CSP or Ambient Calculi offers the best solution because they coincide in two requirements, more than the rest of combinations. Both, OWL-S and CSP or Ambient Calculi decompose systems in processes, and combine them by means of primitives such as Sequence, or Parallel. In fact, the channel concept of CSP and Ambient Calculi has the equivalence in the concept of inputs and outputs of OWL-S processes. Only the concept of preconditions and effects has no direct equivalence in CSP and Ambient Calculi.

Both, CSP and Ambient Calculi belong to a family of formal methods called formal algebras. The main difference between them is that Ambient Calculi is designed to address mobile applications. On the one hand, most of the implementations of Ambient Calculi are based on agent technology. In general, a distributed implementation of agent technology needs capacity of computation in devices, in order to run the agents. On the other hand, CSP specifications can be implemented directly in hardware. Comparing both approaches, agents and hardware implementation, it seems that CSP meets better the cost requirement than Ambient Calculi.

Another possible alternative will be the utilization of Petri Nets as the formal method. This approach is not new, and has been taken before in the specification and analysis of web service compositions in OWL-S [80, 76]. However, literature points to process algebras as a better solution for web service formalisation, as they offer better composability features than Petri Nets [95].

Finally, an interesting alternative is the combination of WSMO with Petri Nets. WSMO service composition (orcherstation or choreography) is based on abstract state machines, which can be easily modeled with Petri Nets [96]. However, it is convenient to remember that CSP has better composability features than Petri Nets.

2.4 Election

In conclusion, in this thesis, the best approach is the combination of OWL-S and CSP. The first reason of this choice stems from the coincidences in modeling components as processes with interfaces (channels in CSP and input/outputs

in OWL-S) capable of being combined to offer more complex services (that are modeled with processes, as well). Secondly, CSP specifications are easily implemented as it is supported by several popular programming languages, such as Java [12, 6] and C++ [39] or specific CSP based languages such as occam [26]. Thirdly, the inherent composability characteristics of CSP, points to the use of CSP for succeeding in the aim of this thesis.

From the previous analysis, if CSP is chosen as the formal method, the best ontology for meeting the requirements of the smart home is OWL-S. Another reason for choosing OWL-S against WSMO is because at the time of developing this thesis, OWL-S was under the process of standarisation by W3C and it was used successfully in several ubiquitous computing projects. We think that these characteristics of OWL-S will mature the technology, and hence, help in meeting the requirements of Reliability and Cost.

The chosen combination of OWL-S and CSP properly meets four of the five requirements, those are: horizontal integration, spontaneous collaboration, reliability and cost. Although the design for domestic use is not addressed directly, it is possible to meet it with the inclusion of an ontology for modeling context information.

This chapter has presented the process followed to conclude the main proposal of this thesis, that is, the combination of an ontology and a formal method in order to acomplish a smart home framework. The next chapter includes the first step in this process, that is, the literature review.

Chapter 3

State of the Art Analysis

This chapter is a review of the past, present and future technologies involved in the implementation of smart home and ubiquitous computing applications. The connecting thread of this literature review is the five requirements introduced in chapter 1 for a smart home: horizontal integration, spontaneous collaboration, and design for domestic use, reliability and cost.

Basically it has been identified the following approaches:

- **Research in smart home**: This chapter introduces some of the numerous smart home projects [66], which fulfill or at least aim to, the requirements presented on Chapter 1.
- **Research in ubiquitous computing**: A smart home can be though of as a particular ubiquitous computing environment. As a consequence, it is worth to study general purpose ubiquitous computing projects suitable for being adapted to a smart home.
- **Research in ubiquitous computing based on the semantic web**, where device operability is achieved by means of the technologies of the semantic web.

The earliest approaches to achieve a smart home were the technologies for home automation. The main objective of these technologies was to develop a common framework in order for home appliances to communicate between them. Accordingly, the common approach was to develop communication protocols over the power line. At the beginning, the communication protocols used were very simple, involving only a naming agreement for home appliances and the communication of simple commands (on/off), like in X-10 [32]. It is estimated that X-10 compatible products can be found in over 10 million American homes. This is

because it has so many advantages over other types of remote control products and systems: inexpensive, no new wiring is required and is very simple to install. However, its principal disadvantage is that X-10 is only suitable for sending very simple commands, making necessary to complete it to succeed in the achievement of a true smart home.

Later, the complexity was increased with the utilization of protocols involving almost all the layers of the OSI tower and the utilization of complex data structures to describe home appliances, like in the Consumer Electronic Bus (CeBUS). In this technology, the Common Application Language (CAL) is used for specifying appliance features and communication. The standard covers communication via the 110V AC powerline (PLC), twisted pair (TP) cable, coax cable, RF and Infrared. The disadvantages of CEBus are the relatively few products currently available and the very high cost of those products.

One of the main promising efforts in home automation is the Home Electronic System (HES) [11]. In HES, home appliances are organised in a hierarchy of devices and subdevices. A subdevice which has no more subdevices is considered a set of objects. Each object may have several members and methods for monitoring, controlling or invoking features of the subdevice. Key device functionality will be encoded in XML (eXtended Markup Language), enabling web communication. This metadata will be stored in a registry in order to make the standard expandable to new products. It seems that HES has only advantages. However, the time to complete the standard (it is incomplete at the moment) is the principal disadvantage of HES.

Other home automation standard is OSGi [16], which uses the concept of service as the basic building block. OSGi can be though of as an environment in which device funtionality is managed by means of Java interfaces. However, devices do not implement directly OSGi, making necessary the completion of OSGi with other technologies, like X-10 or CeBUS. Other home automation standard is HAVi, a standard optimised for Audio Video devices interoperability, which is its principal advantage.

This introduction has presented some of the most popular home automation standards. However, there are still more approaches for achieving home automation. For the interested reader, an exhaustive directory of home automation standards is provided by the Continental Automated Buildings Association (CABA) [4].

In summary, home automation standards only provide the necessary middleware for home devices communication and remote access. These standards do not provide a truly smart home on their own. To overcome this drawback, next sections introduces some projects related with the smart home and ubiquitous

computing.

3.1 Comparative Framework

In order to understand how the different solutions for smart home and ubiquitous computing solve the challenges presented as objectives for this thesis, a systematic study should be done. This study is carried out decomposing each challenge in different aspects, which are exposed in the following lines:

- **Horizontal Integration**: The basic objective of this challenge is to build up a ubiquitous computing scenario by the integration of components. Among other factors, this topic involves the study of the mechanism used for components to cooperate or middleware, the network protocols required and how device functionality is modeled and represented.

- **Spontaneous Collaboration**: In relation with the previous objective, this topic requires the study of the level of expressiveness given to each component. This knowledge is essential to achieve the needed syntactic and semantic agreement. In addition, this objective comprises the study of the instruments related to the representation of context information, and the set of algorithms used to interpret such information.

- **Domestic Use**: This objective includes the declaration of the principal objective for technology studied.

- **Reliability**: Aspects such as the use of standard protocols, tools and the methodology adopted for software development are studied under this challenge.

- **Cost**: Multiple factors influence the final price of a smart home product. In this thesis, reliability and the use of standard protocols and tools are considered as factors to reduce costs.

Tables 3.1, 3.2 and 3.3 summarise the information gathered in this literature review. Related with the utilisation of ontologies, we have taken special interest to that projects that try to achieve pervasiveness through the use of the semantic web [23]. Next sections clarify all the information presented in these tables.

3.1.1 Horizontal Integration

The basic objective of this requirement is the achievement of complex applications by the integration of simple components. The study of this requirement includes

Challenge	Aura http://www.cs.cmu.edu/~aura/	Oxygen http://www.oxygen.lcs.mit.edu/	Portolano http://portolano.cs.washington.edu/	Endeavour http://endeavour.cs.berkeley.edu/
Horizontal Integration				
Network Protocol	TCP/IP	Network technologies which provide essential services such as security, discovery, location and adaptation to environmental changes.	TCP or UDP	HTTP, WAP, ...
Middleware	CORBA, RPC, COM depending on the connector nature	Grid protocol, Resilient Overlay Network for optimal routing and adaptation to changes. Chord and the Intentional Naming System for service discovery.	one.world	Ninja
Device Functionality	Services	Pebbles	Components	Operators
Device Functionality representation	XML	XML, APIs	tuples	XML
Spontaneous Collaboration				
Syntactic agreement	Common tags and attributes	Common tags and attributes	Common tags and attributes	Common tags and attributes
Semantic agreement	Representation of user activities as coalitions of services.	GOALS and Pebbles. Representation of an user goal	Importing and exporting event handlers	Operator transformations and proxies
Domestic Use				
Type of approach	Minimise user distraction. Balance between an attentive and learning home	Help potential users in their daily tasks through automation and collaboration technologies and natural interfaces. Attentive home.	Universal access to information and understanding of user activities. Attentive home.	Making more convenient for people to interact with information, devices, and other people.
Reliability				
Standard protocols	Yes	Yes	Yes	Yes
Helping tools	No	IOA-Tookit,	Yes. Library	Yes
Formal methods	No	Input-Output Automata	Yes. Tuples and space sharing	Yes

Table 3.1: Ubiquitous Computing Technologies

topics such as the how components are modelled, the communication protocol and middleware between components and how components are physically represented.

The most common approach to represent components is by means of services. Generally, a service represents the functionality of a device. Sometimes there exists exceptions to this affirmation. For example, in the Aware Home project services are though of as a kind of actuator in the environment, and in Aura, services are considered as parts or steps of user activities. An interesting alternative to modeling components as services appears in Oxygen [18]. In this project, the basic building blocks are ”Pebbles” [19]. Pebbles are platform-independent software components, capable of being assembled dynamically from goals. In addition, Pebbles are under research at Cambridge University in order to incorporate this technology to the AutoHan project [2]. Topics such as the incorporation of the semantic web and the creation of tools for inferring future behaviour of Pebbles compositions are new challenges for this technology.

Commonly, communication between components is achieved through the use of standard Internet protocols such as TCP, UDP, HTTP and SOAP. Although the use of internet protocols seems very valuable to facilitate device interoperation, nowadays is difficult to find home appliances with capacity of engage in these protocols. This challenge would be solved, for example with the incorporation of X-10 devices and gateways between the X-10 and Internet networks. Based on internet protocols, the most sophisticated approach for component communication is the use of service discovery architectures. Service discovery architectures

Challenge	TEC http://taskcomputing.org/	Web Services NA	GAS NA
Horizontal Integration			
Network Protocol	HTTP, SOAP	HTTP, SOAP	Bluetooth, IEEE 802.11
Middleware	Web Services, UpnP	Web Services	GAS-OS
Device Functionality	Services	Services	eGadgets
Device Functionality representation	XML	XML	XML
Spontaneous Collaboration			
Syntactic agreement	OWL	OWL	Common attributes and tags
Semantic agreement	OWL-S. Input and output compatibility	OWL-S plus ontologies for context information	GAS ontology. Plugs and synapse
Domestic Use			
Type of approach	Intended to help users in their task letting them to focus in what they want to accomplish instead of how.	Focus on how a service flow can be automatically composed using syntactic, semantic and pragmatic knowledge.	Enable people to compose ubiquitous computing applications by combining services offered by everyday devices
Reliability			
Standard protocols	Yes	Yes	Yes
Helping tools	Yes. OWL-S API, pellet	Editor	Editor
Formal methods	No	No	No

Table 3.2: Ubiquitous Computing Technologies based on the Semantic Web

provide a framework capable of achieving seamless inter-device discovery and communication. As a consequence, research in ubiquitous computing points to service oriented architectures in which devices expose their functionality as a set of services [59, 91, 108].

The middleware used for components to cooperate differs between projects. Fortunately it is possible to classify them in two classes: standard midlewares such as CORBA, Web Services or UPnP used in Aura [1, 102] or TEC [24, 74] or, particular communication layers such as one.world [56] or Ninja [55], designed for Endeavour [7] or Portolano [20]. Standard middlewares have the advantage of ensuring broad support and the capability of integration with legacy systems. However, sometimes standard home appliances can not engage in standard protocols because the lack of resources. In this case, the utilization of custom middlewares, specially designed for low resources devices will solve this challenge. The choice of the middleware will depend on the kind of components to integrate.

It seems clear the trend is to model device functionality as services represented in XML. However, the schema adopted depends on the middleware used. For example, UPnP [28] based projects require the use of UPnP schema while the utilisation of Web Services requires the use of WSDL. The lack of a common schema to interpret device funtionality can lead to interoperability issues. To overcome this drawback, an approach to give uniformity for device representations

Challenge	Home_n http://architecture.mit.edu/house_n/	The Aware Home http://www.awarehome.gatech.edu/	The Adaptive Home http://www.cs.colorado.edu/~mozer/house/
Horizontal Integration			
Network Protocol	Ethernet, UDP, TCP/IP, HTTP	TPC/IP, HTTP	X-10
Middleware	Specific APIs	XML over HTTP	ACHE architecture.
Device Functionality	Applets, specific modules, cabinets	Context widgets	Device regulators, set point generators, predictors, state transformations and occupancy models.
Device Functionality representation	NA	XML	NA
Spontaneous Collaboration			
Syntactic agreement	NA	Common tags and attributes	Mathematical formulas
Semantic agreement	NA	Objects and relations between them	Valuation of the energy cost and discomfort offered by each device
Domestic Use			
Type of approach	A mean to automatically control the environment, but instead to help its occupants learn how to control the environment on their own. Attentive home	Creation of a smart home capable of enhancing and keeping independent the life of its occupants. Attentive home	Smart home capable of modifying itself by observing the lifestyle and aspirations of the home residents, and learning to and conform their needs. Adaptive home
Reliability			
Standard protocols	Yes	Yes	Yes
Helping tools	Sensor Toolkit, Kitchen design tool, context data tools,	Yes	No
Formal methods	No	No	No

Table 3.3: Smart Home Technologies

includes the use of the semantic web. Especially interesting are those projects that use semantic web for describing device functionality. In this case, the Resource Description Framework (RDF) [22], which is based on XML, is the support for developing service descriptions. RDF facilitates the automatic interpretation of information by means of a set of well known rules to represent information.

This section has presented different ways for meeting the requirement of horizontal integration. The next step involves the interpretation of the semantics of each component in order to understand its functionality. This challenge is the topic of the next section.

3.1.2 Spontaneous Collaboration

Proper spontaneous collaboration claims for components to cooperate without any planning in advance. To achieve this challenge, the syntactic and semantic agreement on components should be in accordance with the strategy for component composition.

Typically, the needed semantics to achieve spontaneous collaboration is obtained by means of the agreement in concepts through a common vocabulary. For example, in Aura, services are described in XML and services of the same type may share a vocabulary in the form of tags for referencing common attributes. In Aura, the key feature is to explicitly model user tasks as coalitions of abstract services. Examples of tasks are writing a paper, preparing a presentation, and examples of services are a video player or a text editor. Under this approach, the system can recognise when a task can be supported in an environment as it knows the services that compose the task.

In some projects, specially those ones for ubiquitous computing, the approach

of a common vocabulary is formalised through the use of ontologies, like in TEC or in GAS. The TEC technology is implemented with existing Internet standards; using web clients, OWL-S for semantic service descriptions, and UPnP and Web Services for service invocation. With a similar approach, the Web-Based Semantic Pervasive Computing Services project [72] proposes a ubiquitous computing scenario focusing on how a service flow can be automatically composed using syntactic, semantic and pragmatic knowledge. In this case, OWL-S and its combination with other ontologies give the semantic knowledge needed to understand service descriptions. Especially interesting is the use of the CONON ontology [103], which gives the needed pragmatic knowledge by means of representing user context information.

In general, when ontologies are used for giving semantics, services are composed by means of intelligent systems capable of understanding service descriptions. For example, is common the combination of services by means of goal based systems, which try to match services with compatible preconditions/effects, in a goal directed approach. Another technique for service composition is the utilisation of rule based systems, in which a set of rules govern how services may be composed. For example, imagine a system feed with a rule that connects all devices with compatible input/outputs. With this system, if the householder decides to switch on his TV and his digital camera, if the system detects that both devices have compatible input and outputs, they will be connected, offering to the user the service of watching his pictures on his TV.

An alternative way for achieving device composition is presented in the Adaptive Home [25]. In this project, the control system is implemented as a neural network which learns the behavioural patterns of home residents [77]. This approach has been considered for other smart home projects, as well [45]. With the information of behavioural patterns of home residents, the system performs the required actions to accommodate user necessities. Examples of system achievements include: detecting water usage patterns, such that hot water is never used in the middle of the day on weekdays, allowing the water heater to shut off at those times; inferring occupant whereabouts and activities and predicting when the occupants will return home and determining when to start heating the house so that a comfortable temperature is reached by the time the occupants arrive. In this project device descriptions are enhanced with measures of the cost and satisfaction feel by the user. These magnitudes are needed for the neural network which rules device composition.

As we have discussed, the typical way of achieving spontaneous collaboration is by means of an intelligent system capable of interpreting the semantics offered by home devices. The objective of this intelligent system is to offer to the householder

useful applications for their daily activities. This objective coincides with the requirement of design for domestic use of this thesis. How this requirement would be achieved is the topic of the next section.

3.1.3 Design for Domestic Use

This challenge establishes a clear frontier between smart home and ubiquitous computing projects. In general, smart homes are designed for domestic use, while ubiquitous computing projects are generally designed for an office environment.

For smart homes, the principal aim is to help users in their daily activities. However, there are still some small differences in the objectives of the various projects studied in this thesis. For example, the Home_n project [10] uses ubiquitous computing for empowering home residents with information to make decisions, keeping home residents active, and conscious of their sense of control [63]. As another example, the Aware Home project focus in the design of appropriate experiences for house residents, especially for elderly adults to keep independence. In the Aware Home project, researchers have developed applications to allow seamless communication between the home resident and its family [79, 100], memory remainders for daily tasks or home assistants to control home devices with simple voice commands or gestures.

In the case of smart homes, it is possible to differentiate between learning and attentive homes [33]. On the one hand, learning homes include projects in which a system records the patterns of behaviour of home residents to anticipate user needs and hence, control the technology, like the Adaptive Home. On the other hand, the attentive homes monitor constantly user activities and context in order to control home devices in anticipation to user needs. Examples of attentive home projects are Home_n and the Aware Home.

Commonly, ubiquitous computing applications are designed for office environments. In fact, the origin of ubiquitous computing was to enhance office activities. The utilisation of ubiquitous computing technologies for domestic use requires the adaptation of the semantic and pragmatic knowledge attached to these applications to a home environment. For example, the TEC project will require the specification of services for home appliances while the Web-Based Semantic Pervasive Computing Services project lacks of specific ontologies for representing the proper context of a home environment.

Shortening, it is possible to relate the way in which different projects tackle the requirement of design for domestic use with the users' concerns towards an smart home presented in table 1.3. Typically, these concerns include the need of keeping householders active and independent and the necessity of a smart home

to anticipate user needs. In fact, the need of reliable systems is another concern of potential smart home consumers. This requirement, is the topic of the next section.

3.1.4 Reliability

The objective of this requirement is to create trust in the final smart home consumer. In this discussion, the use of standard protocols and tools and the use of formal methods are considered factors to enhance the reliability of applications.

A common way to achieve reliability proposes the use of standard well known protocols for component communication, as it was introduced in section 1.3. In general terms, well known standard protocols benefit from wide use and testing, which improve their reliability.

Another approach to provide reliability involves the use of tools for developing ubiquitous computing scenarios and automatic code generation. It seems that the use of this kind of tools might reduce the amount of errors derived from software development. This is the case of the Toolkit for Rapid Prototyping of Context-Aware Applications, which belongs to the Aware Home project.

An interesting approach coming from the world of hardware design points to the use of formal methods in all stages of a smart home project, as it is proposed in the Portolano [20, 50] and the Oxygen projects [18]. This approach allows developers to predict how the system is going to behave, detect errors in the design process, and consequently achieve better levels of reliability in the final implementation [42].

The implementation of realiable systems have many advantages, being one of them the reduction of costs derived from system failures. Next section goes deeper on how a reduction of cost can be achieved when tackling the challenge of a smart home.

3.1.5 Cost

Costs of deploying and maintaining smart homes are one of the main concerns of potential home users. In fact, cost reduction may enhance the achievement of effective business models for ubiquitous computing applications, one of the challenges for ubiqutious computing systems. In this thesis, the cost reduction strategy focuses on two factors: Reliability and low cost implementations.

The implementation of reliable systems, as it is described in the previous section, will reduce developing, implementation and maintenance costs. The use of formal methods allows the detection of errors in the development stage, before system implementation and deployment. This will notably reduce costs as it is

much simpler to correct errors when the system is designed than once the system is implemented and deployed [81, 37].

The use of standard protocols for achieving reliability will benefit from seamless system integration, no learning curves and the use of well known tools and methods. However, sometimes the implementation of these protocols, specially those ones designed for Internet, might be complicated for simple home devices. For example, the use of a web service requires devices to host a web server, which might be very demanding for a simple light switch. The use of tools for system prototyping may reduce the development costs and achieve more reliable systems, as well [58].

Secondly, an implementation of smart home requires some kind of intelligence and resources deployed on home appliances in order to engage in a communication protocol. The lower this amount of intelligence, the smaller the cost of the final smart home implementation. In light of this, the achievement of simple communication protocols will allow the use of hardware close implementations, which benefit from low manufacturing costs.

This chapter has presented the different approaches found in the literature to tackle the challenge of a smart home, focusing on how the different projects studied meet the requirements presented in chapter 1. It seems that those requirements would be fit by the use of a service oriented architecture in which services are represented by an ontology and modeled with a formal method. The objective of the next chapter is to give a deeper point of view of this assertion.

Chapter 4

Smart Home Proposal

The analysis performed in the previous chapter has provided some guidelines for achieving a smart home. Firstly, the most widely accepted way of achieving horizontal integration is through the representation of device functionality as services which interact by means of standard Internet protocols and middlewares. Secondly, the use of ontologies combined with rule-based and goal systems seems to be the best approach to achieve spontaneous collaboration. Thirdly, in order for potential customers to perceive value from a smart home, it should be designed as flexible as possible, that is, capable of being attentive or learning, depending on user needs. In the fourth place, the combination of standard protocols, provision of tools for system development and the integration of formal methods provides the needed level of reliability for a smart home to succeed. Finally, the combination of all the guidelines formerly presented will reduce the cost of the final smart home.

4.1 Ontologies and Formal Methods

From the analysis done in Chapter 3, it seems that all requirements presented in this thesis for a smart home may be achieved through the combination of an ontology and a formal method. To reinforce this affirmation, figure 4.1 plots the requirements introduced for a smart home. Firstly, research in ubiquitous computing points to architectures in which components are modelled as services. Once established that components will be modelled as services, the second requirement, spontaneous collaboration, demands a rich semantic and syntactic description of such services. Thirdly, the design for domestic use requires the understanding of user activities and intention. Several algorithms have been proposed to achieve this goal, sharing all of them the necessity of modelling context information. It seems

that all these requirements can be fulfilled with the use of service-oriented architectures in which services and context information may be represented through ontologies [60].

At the bottom of figure 4.1, the reliability requirement suggests the study of a smart home under the discipline of distributed and embedded systems. On one hand, home device cooperation requests the adoption of distributed computing techniques. On the other hand, most of these devices participating in such kind of cooperation are controlled by embedded systems. During the 70s, both topics, distributed and embedded systems, started to be analysed with formal methods [67]. In fact, the Information Society Technologies Advisory Group (ISTAG), which acts as an European Commission consultant recommends the use of formal methods in the design process of ubiquitous technologies [53]. Accordingly, well known standards for home networks such as IEEE 1394 or HAVi have been designed with the help of formal methods [51]. Especially interesting are those formal methods which allow the automatic generation of code, that is, the final implementation of the specification.

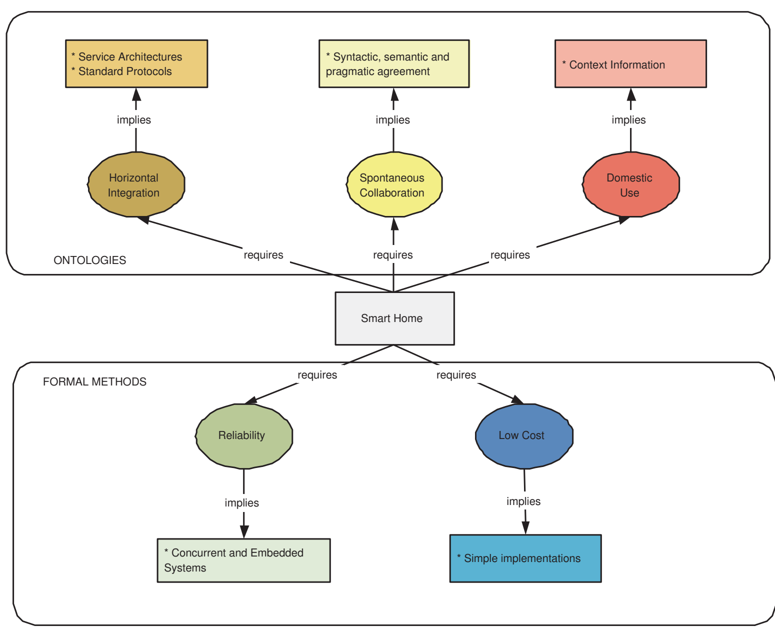


Figure 4.1: Ontologies and Formal Method Relation in an Smart Home Project

A similar challenge to horizontal integration and spontaneous collaboration has raised in the world of web services. Web service composition is a current area of research with important results for this thesis. In light of this, several projects attempt to facilitate web service composition. Business Process Execution Language for Web Services (BPEL) [3] and XLANG [31], together with OWL-S [17], are examples of these projects. All these projects model web services as

processes that may interact with each other. The main difference between BPEL, XLANG and OWL-S is that OWL-S is an ontology, while BPEL and XLANG are not. As a consequence, BPEL and XLANG lack the needed semantics for achieving automatic service composition [75].

In connection with the combination of ontologies and formal methods, the formal verification of web service orchestration has been justified because of the susceptibility of business processes to errors due to the high level of concurrency, distributed computing and dependency of external entities of these applications [35]. This challenge has been tackled applying formal analysis to OWL-S [80, 76] and BPEL models [95]. These projects translate composed services to Petri nets and process algebras respectively, mainly, for verification purposes. As it was discussed in chapter 2, we propose the utilisation of process algebras, because they provide better composability features than Petri nets [95].

4.2 Smart Home Proposal

Accordingly to the previous section, this work proposes a model in which home appliance services are offered by a set of embedded processes, which are composed in a dataflow network [90]. These processes are described by an ontology which is subsequently verified and implemented by means of a formal method, as shown in figure 4.2

The semantics is given by the Semantic Markup for Web Services (OWL-S), an upper ontology for describing web services and their composition. The chosen formal method is (Communicating Sequential Processes) CSP [62, 94]. It is a process algebra for describing concurrent systems whose component processes interact with each other by communication. CSP specifications are easily implemented as it is supported by several popular programming languages, such as

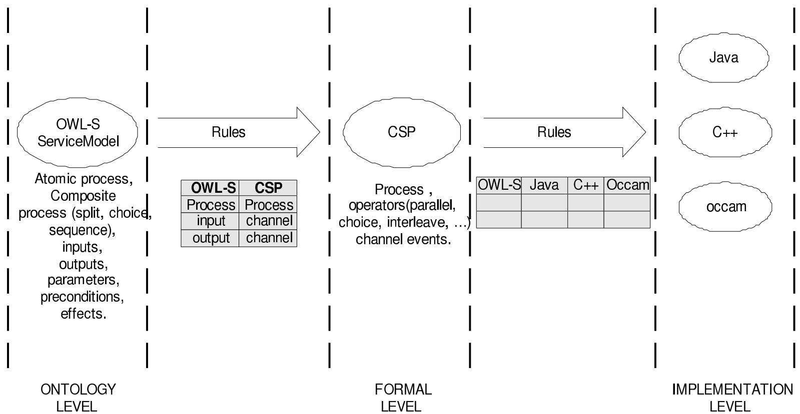


Figure 4.2: Smart Home Proposal

Java [105, 61] and C++ [39] or specific CSP based languages such as occam [26]. The aim is to find a procedure that, given a service description in the OWL-S service model, finds the most suitable implementation for the service [88]. The service model description is translated to CSP, which gives a formal view of the service. At this level, it is possible to check the correctness of the service in terms of deadlocks and other topics related with concurrent systems. Once the service is proved safe, with the use of predefined procedures it is possible to obtain the desired implementation.

Chapter 5

Semantic Home Device Services

The main purpose of this chapter is to achieve an ontological representation of home appliances services. Not only this representation should be rich enough for automatic service composition, but also capable of being safely implemented in a wide range of devices. We propose an upper ontology for describing user-centred device services, based on OWL-S and the context ontology CONON. The proposed layout facilitates the service implementation and provides the needed classes for developers to specify in which situations a service might be used.

An analogy occurs with systems specified in UML. Such kind of systems are comprised by not only a static representation of classes, objects, methods and their relations, but also the uses of cases of such elements. The static representation might be enough for the software engineer to obtain a system implementation by means of automatic code generator tools. The use of case diagrams will provide all the information necessary for understanding how the service will behave in particular situations.

The chapter starts with an introduction to OWL-S and CONON. An ontology for modelling home appliance devices is then presented.

5.1 Web Service Ontology (OWL-S)

OWL-S is a proposal based on OWL (Ontology Web Language) [27] which specifies an upper ontology for service composition, providing three different knowledge types about a service, as it is shown in figure 5.1.

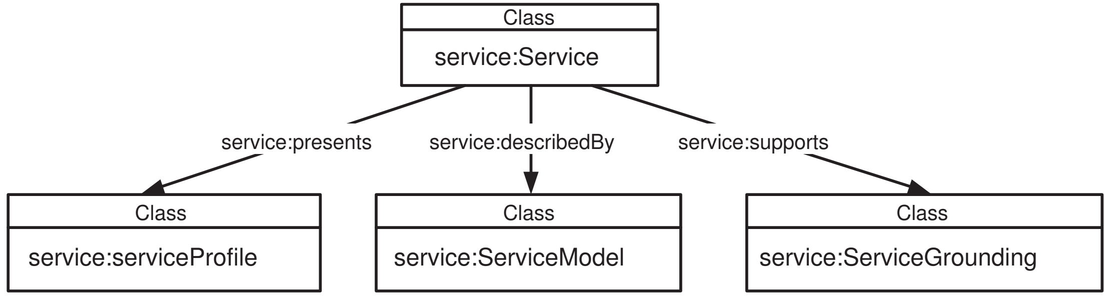


Figure 5.1: OWL-S Class Diagram

Class *service:ServiceProfile* [1] provides all the information about service capabilities . This class specifies a service as a set of inputs, outputs, preconditions and results. With this information, web services are composed by means of class *service:serviceModel*, which gives a process view of services. Once services are composed, class *service:ServiceGrounding* offers all details about their invocation.

Among other information related to a service, class *service:ServiceProfile* presents what function the service computes. This information is expressed in terms of the transformation that the service produces. Particularly, the profile specifies the inputs required by the service and the outputs generated. Moreover, the profile describes the required preconditions by the service and the expected effects that result from the service execution. This functionality is exposed through properties *profile:hasInput*[2] , *profile:hasOutput*, *profile:hasPrecondition* and *profile:hasResult*.

A service profile is a simplified view of a service, since it only gives information about what a service does. To give a more detailed perspective, the *service:ServiceModel* class describes services as processes. There exist atomic processes that only transform inputs to outputs, and composite processes that are composed by other processes (atomic or composite) using control constructs such as *Sequence*, $If - Then - Else$ or *Choice*.

The execution of an OWL-S service can be compared with a combination of remote procedure calls. The OWL-S grounding specifies all the semantics of the parameters to be provided when executing these calls, and the semantics that is returned in messages when the services succeed or fail. A software service user should be able to interpret the grounding class to understand what input is necessary to invoke the service, and what information will be returned.

[1]service is the namespace for `http://www.daml.org/services/owl-s/1.1/Service.owl`
[2]profile is the namespace for `http://www.daml.org/services/owl-s/1.1/Profile.owl`

5.2 CONON: The Context Ontology

CONtext ONtology (CONON) [103] is an OWL encoded upper ontology for modelling context in ubiquitous computing environments. Particular ubiquitous scenarios will extend the diagram presented in figure 5.2.

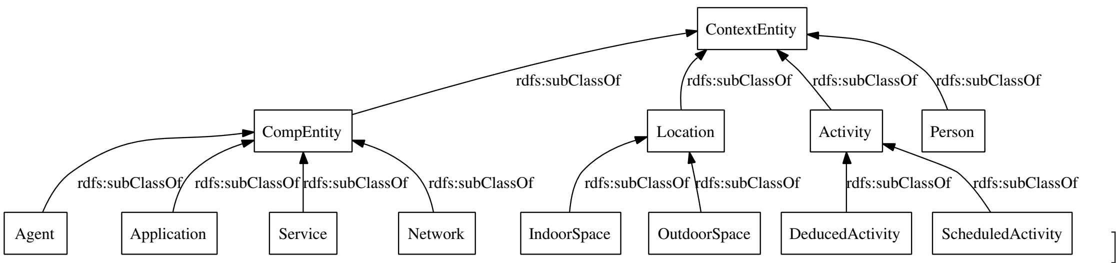


Figure 5.2: CONON Upper Ontology

As is shown in figure 5.2, the CONON authors assume that context information can be grouped in four main categories: computational entities, geographical position, people and activities. These four categories are represented by classes *CompEntity*, *Location*, *Person* and *Activity*, respectively.

Class *Location* represents the geographical position of the rest of CONON elements. This class has two subclasess, *IndoorSpace* and *OutdoorSpace*, which represent places inside and outside the home environment, respectively. People are represented by the class *Person* and their related activities by classes *DeducedActivity* and *ScheduledActivity*, which are subclasses of *Activity*. Finally, class *CompEntity* involves all the computational entities which may represent part of the user context, such as devices, services, applications and the network for these entities to communicate.

5.3 Service Representation

In this thesis, device functionality is published as services. For these services, we propose an ontological representation which is composed by two points of view. Firstly, a static representation based on OWL-S is intended to give enough syntactic and semantic information on how to use a service. In addition, this static information should be capable of being translated to CSP, in order to obtain the service implementation. Secondly, since the static representation does not provide enough knowledge for composing valuable services for home residents, we propose a dynamic service perspective based on the CONON ontology.

5.3.1 Static Service Representation

Our aim is to find a procedure such that, given a service description in OWL-S, a correct implementation for the service is obtained. It seems that this goal can be achieved by exploiting the similarities between OWL-S and CSP. Both of them specify systems in terms of processes and their interfaces (inputs and outputs in OWL-S and channels in CSP). However, not all OWL-S structures have their symmetrical in CSP and vice-versa. For this reason, we propose an upper ontology for developing services, mainly based on OWL-S, which instances can be automatically translated to CSP and subsequently, to a CSP based language in order to implement the service.

In our model, a service is responsible for offering the functionality of a device. Each service has a *state* and may offer a set of atomic procedures called *actions* to read or modify the state. Services may be interested in the state of other services, especially, when the state changes. For this reason, every service may output changes in its state through special outputs called *events* [3]. Finally, some services may be capable of streaming data. This feature is represented by *plugs*, which act as ports to input or output data.

To extend OWL-S to fit the proposed model, the first step involves subclassing *service:Service*, *profile:Profile*, *process:Proccess* and *grounding:Grounding* with *owlsx:LufService* [4], *owlsx:LufProfile*, *owlsx:LufProccess* and *owlsx:LufGrounding*, as it is shown in figure 5.3.

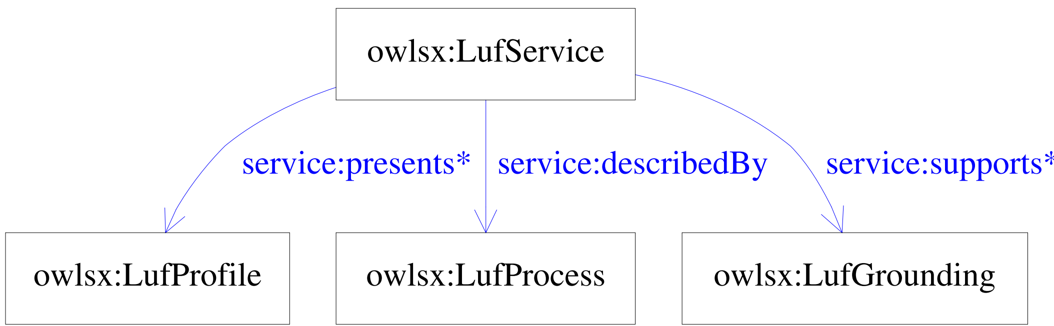


Figure 5.3: LufService Class Diagram

The principal elements of an instance of *owlsx:LufService*, such as the state, plugs, actions and events are exposed through class *owlsx:LufProcess*, as it is plotted in figure 5.4. The state, events, plugs and actions are attached to a service through properties *owlsx:hasStateVariable*, *owlsx:hasEvent*, *owlsx:hasPlug* and *owlsx:hasAction*, which range to instances of classes *owlsx:State*, *owlsx:Event*,

[3]This vision of events is different than the CSP definition of events

[4]owlsx is the namespace for `http://www.lboro.ac.uk/owlsextension.owl`

owlsx:Plug and *owlsx:Action*, respectively. The set of constrains of how a service can be invoked are stablised by means of a set of preconditions/effects, which are detailed in the next sections of this thesis.

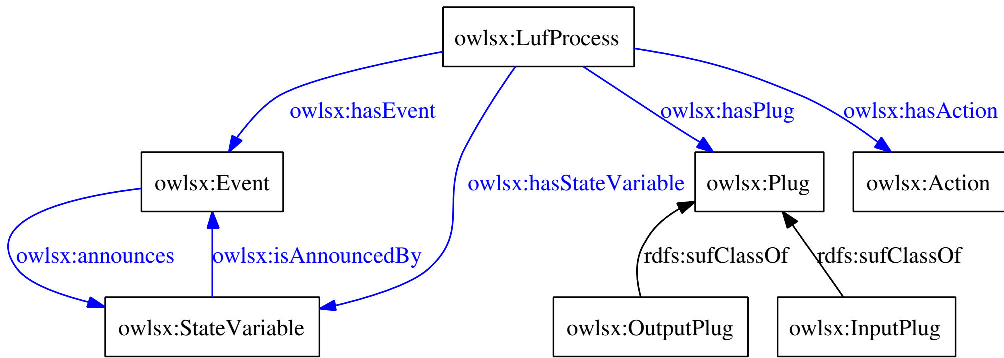


Figure 5.4: LufProcess Class Diagram

Events and state variables are related through property *owlsx:isAnnouncedBy*. The relation between actions and state variables is represented by rules of the style "if action A is invoked, then something occurs with state variable S". This kind of action consequence is represented as instances of classes *owlsx:SetStateResult* and *owlsx:GetStateResults* respectively. Both classes relate state variables, inputs, outputs and literal values by means of the bindings attached to their properties *owlsx:withStateVariableBinding* and *process:withOutputBinding*.

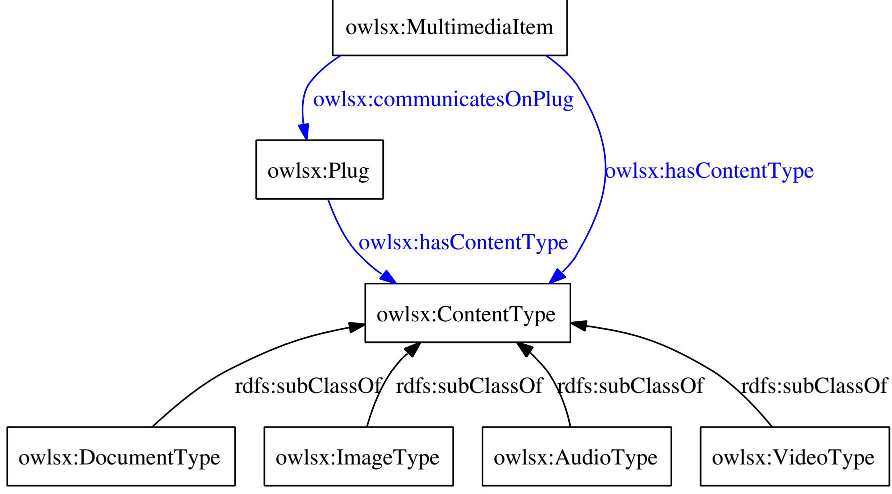


Figure 5.5: Data Communication

As is shown in figure 5.5, data communication is achieved with class *owlsx:Plug*. The data format associated to each plug is attached with property *owlsx:hasContentType*. In this work, four data formats are considered, represented with classes *owlsx:Audio*, *owlsx:Video*, *owlsx:Image* and *owlsx:Document*. The particular piece of data trans-

mitted on a plug is modelled as an instance of class *owlsx:MultimediaItem*. A multimedia item contains information about the data transmitted and it is related to a plug through property *owlsx:communicatesOnPlug*.

Consider a simple service for controlling a washing machine plotted in figure 5.6. This service has four operations and two state variables. Operations *OpenDoor* and *CloseDoor* are responsible for setting the value of the state variable *DoorStatus*, which represents whether the washing machine door is open or close. The running status of the washing machine is indicated by a state variable called *WashingStatus*, which is controlled by the operations *StartWashing* and *StopWashing*. The type of each state variable is indicated by the value of property *process:parameterType*. In addition, both state variables are initialised to their default values, as it is indicated by property *process:parameterValue*.

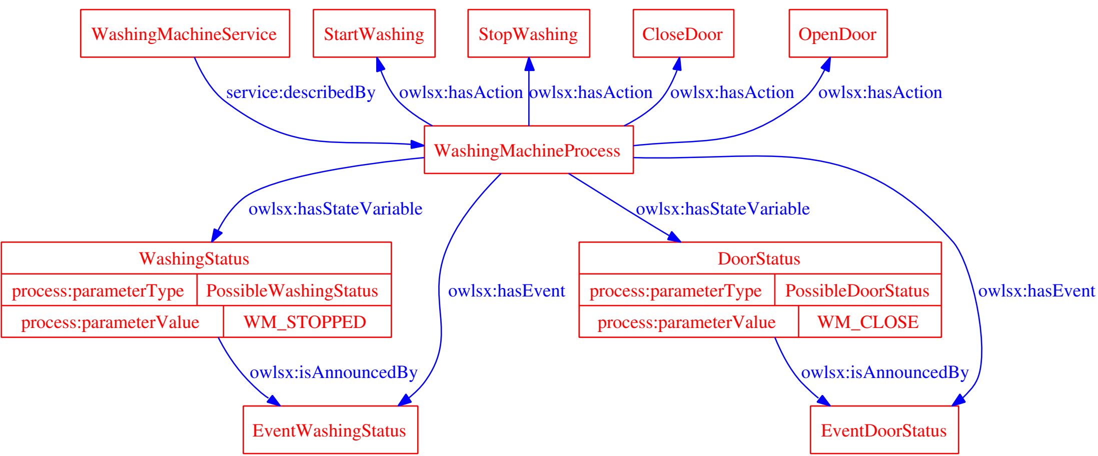


Figure 5.6: Washing Machine Service

5.3.2 Dynamic Service Representation

The static representation gives enough syntactic and semantic information about how to use a service. This information comprises aspects such as how a service can be interconnected, by means of its plugs, and what is the consequence of invoking determined actions on the state variables. However, this knowledge is not enough for composing valuable services for home residents, as it does not represent what user needs are. In order for developers to provide user-centred device services, service developers might specify the activities in which the device service might be involved. For example, a DVD might imply a user engaged in watching a film. A truly user-centred design demands considering all requirements for the home resident to use a service [99]. Continuing with the example, watching a film requires the sight and hearing senses from the user.

To allow developers to specify the activities related with services, we propose to extend the CONON ontology, paying special attention to the *Activity* class.

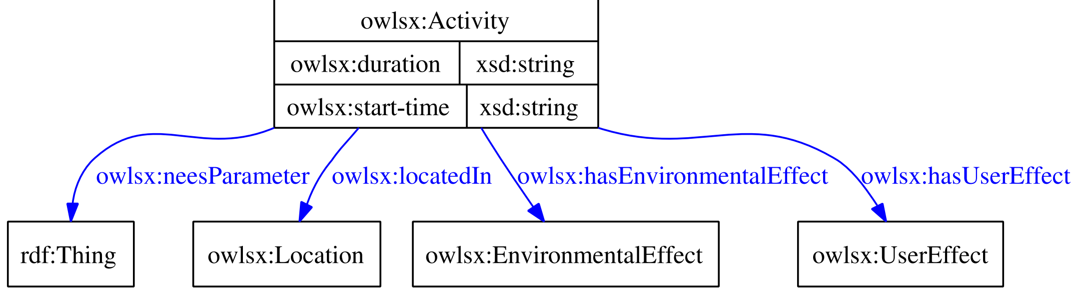


Figure 5.7: Activity Class Diagram

As is shown in figure 5.7, an activity might be located in a physical place through the *owlsx:locatedIn* property. Properties *owlx:duration* and *owlx:starttime* relate an activity to its duration and starting temporal point. In addition, property *owlsx:needsParameter* is used to attach a parameter to the activity. This parameter should be interpreted according to the semantics given to the type of activity considered.

Activities might have effects on the environment or on the user himself. Determining such activities and effects is complicated because of the complexity of the household environment [82]. For example, opening a window has the environmental effect of heating up or cooling, depending on the temperature difference between the room and the outside, while answering a phone has the user effect of keeping busy his speaking and listening skills. In this thesis, we have adopted a simple approach in which activity effects are modelled in the contex ontology by means of classes *owlsx:EnvironmentalEffect* and *owlsx:UserEffect* which are subclasses of *owlsx:ActivityEffect*, as is shown in figure 5.8.

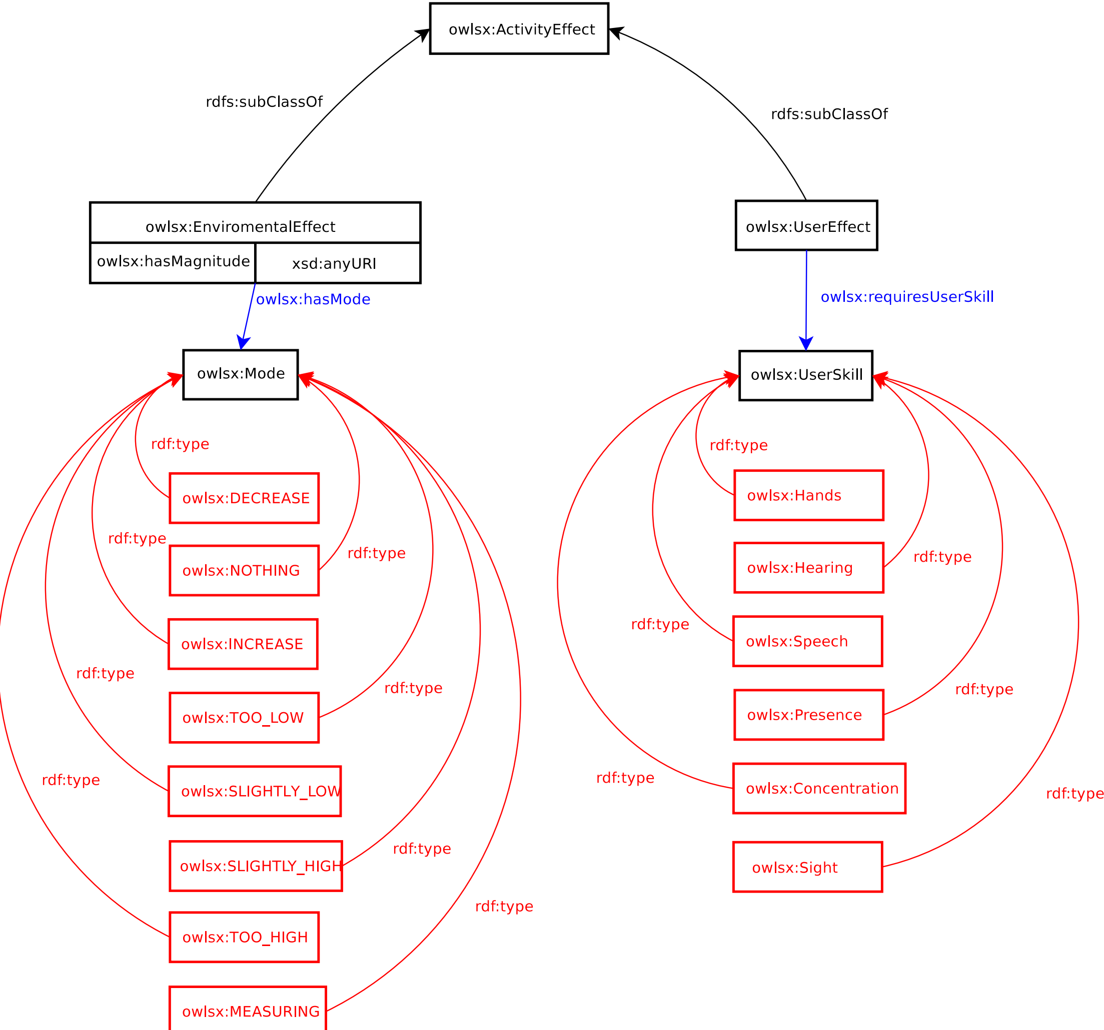


Figure 5.8: Activity Effects Class Diagram

In this work, activities are classified depending on the nature of the effect caused, as it is shown in figure 5.9. Firstly, a division between automation activities, class *owlsx:HomeAutomation*, and user activities, class *owlsx:UserActivity*, is defined.

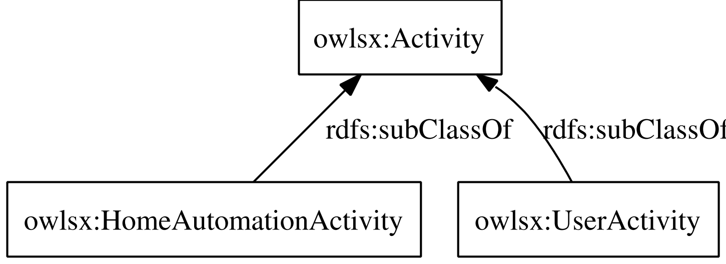


Figure 5.9: Types of Activities

Figure 5.10 represents the first type of activities, which belongs to the set of

tasks related the monitoring and control of some environmental variables inside a home, such as the temperature or the level of noise. These activities relates a state variable (property *owlsx:needsParameter*), a location and an environmental magnitude. The relation among these entities is determined by the particular subclass of *owlsx:HomeAutomation* and the type of environmental effect attached. This is the reason for defining classes *owlsx:SensingActivity*, *owlsx:ActuatorAutomation* and *owlsx:DiagnosisActivity*. The semantics given to these activities allows the differentiation of whether a service acts as a sensor, actuator or a diagnosis device. As an example, a service for increasing the temperature might have attached an actuator activity which indicates that the value of one of its state variables, and hence an action, will increase the temperature of a particular location. This activity should have attached an environmental effect with the values *owlsx : INCREASE* and *owlsx : Temperature* for properties *owlsx : hasMode* and *owlsx : hasMagnitude* respectively. A deep description of automation activities is presented in Chapter 8 with the implementation of an HVAC System.

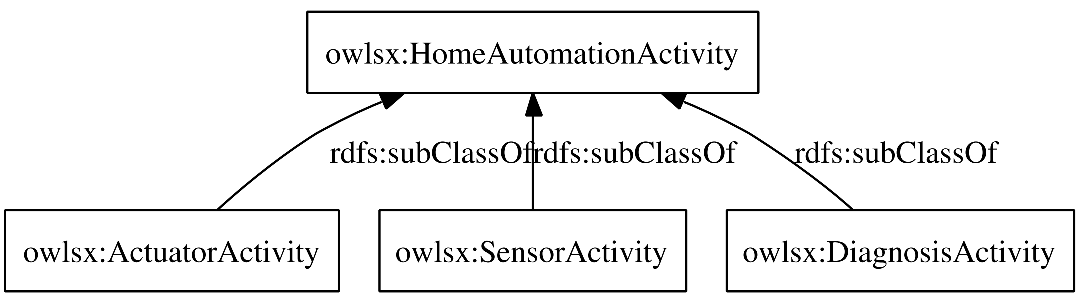


Figure 5.10: Home Automation Activity

The second type of activities, user activities, represents the rest of tasks related to the user. In this group, actions such as watching a television or listening to the radio are examples of user activities. These type of activities may have attached effects of type *owlsx:EnvironmentalEffect* or *owlsx:UserEffect*.

Each activity defines a life cycle defined by its state (property *owlsx:ActivityStatus*) and the state transitions (properties *owlsx:startTrigger*, *owlsx:suspendTrigger* and *owlsx:stopTrigger*). As indicated in figure 5.11, each activity should be active, suspended or stopped depending on the value of property *owlsx:ActivityStatus* which ranges from *owlsx:ACTIVE*, *owlsx:SUSPENDED* and *owlsx:STOPPED*.

Triggers encode information as a set of rules that determine the state of an activity. These rules define context information such as the value of service state variables or user locations.

The enhancement of activities with state and triggers allows manipulation of this information both with a forward rule-based and a goal systems. On the one

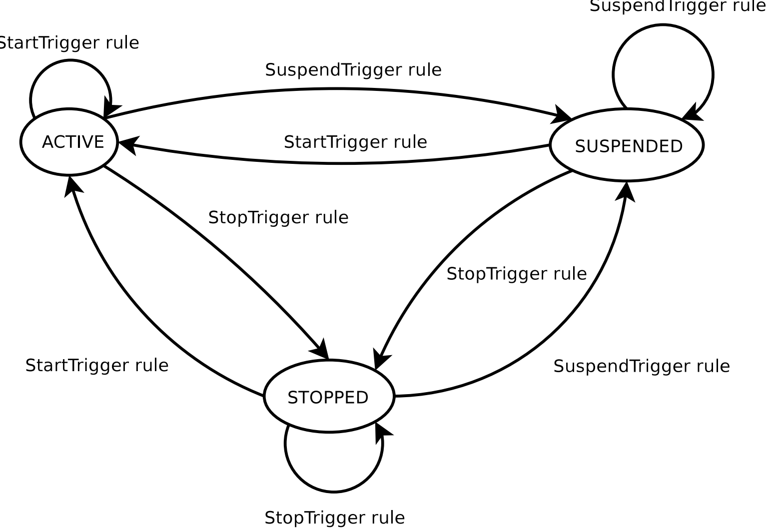


Figure 5.11: Activity Diagram

hand, with the forward schema, it is possible for a rule based system to inference the status of each activity through the knowledge of the trigger state. On the other hand, the status of an activity can be thought of as a goal which can be achieved setting the proper triggers to a particular value. This is the strategy adopted in the simulator introduced in Chapter 8.

Let us consider the user activity *WashingLaundry*, which is associated with the washing machine service presented in figure 5.6. This activity can be only active or stopped. Figure 5.12 indicates the transitions between these two states depending on the value of the state variable *WashingStatus*.

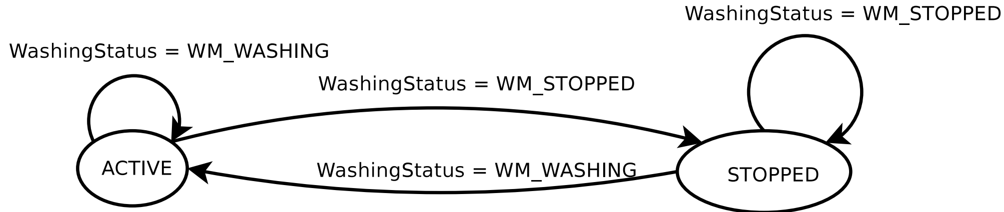


Figure 5.12: *WashingLaundry* Activity Diagram

Every time that the *WashingStatus* state variable is set to *WM_STOPPED*, a rule-based system may deduce that activity *WashingLaundry* is stopped. At the same time, a goal-based system may inference that the required procedure to stop this activity is to set to *WM_STOPPED* the value of the state variable *WashingStatus* through action *StopWashing*.

This chapter has introduced the semantic level of our approach. It was presented a upper ontology for modeling home devices services and their related con-

text information. The main objective of this part of our framework is to achieve the requirements of horizontal integration, spontaneous collaboration and design for domestic use. Next chapter completes this framework with the incorporation of the formal algebra CSP, which principal objective is the achievement of the requirement relative to reliability.

Chapter 6

Formal Specification of Home Appliance Services

This chapter presents a procedure by which, a CSP specification for a service is obtained given its description with the ontology presented in Chapter 5. This provides two benefits. Firstly, the formal specification of the static part of each service allows developers to automatically obtain service implementations based on CSP. Secondly, due to the CSP refinement technique, a formal specification of the service dynamic part permits determining how a service is going to behave in terms of the activities involved.

After a brief introduction to CSP, the rules for how a service should be translated to CSP are presented. These rules differentiate between the static and dynamic parts of the service. Finally, a technique based on CSP refinement is presented to predict the behaviour of services in a particular context.

6.1 Communicating Sequential Processes (CSP)

CSP (Communicating Sequential Process) is a process algebra for modelling concurrent systems devised by C.R.Hoare [62]. In CSP, the behaviour of a process is described by the sequence of events or actions that it may perform. Table 6.1 plots part of the notation used in CSP. In this table, it is assumed that a and b are elements of the sets A and B respectively, which are the alphabets of processes P and Q respectively. A process alphabet is the set of events that it is allowed to perform.

Important in CSP is the concept of *channel*. Events of the form $c!v$ stand for the transmission of message v on channel c. Each channel has a *type* which declares the set of values which can be passed on it. If T is the type of channel c, then the set of events related with c is $\{c!t \mid t \in T\}$. Based on this abstraction,

Table 6.1: CSP_M Operators

Operator	Behaviour
Prefixing $(->)$	$a \rightarrow P$ is a process that behaves like P after doing event a
Sequence $(;)$	$P; Q$ represents a process that behaves like Q after behaving like P
Choice $(\|)$	$a \rightarrow P \mid b \rightarrow Q$ is a process which can either engage in event a and then behave like P, or do event b and then behave like Q
Parallel $(\|\|)$	$P_A \,\|\|_B\, Q$ is a process that behaves as the concurrent composition of P and Q, but requires synchronisation between P and Q in the events belonging to the intersection of A and B

process $c!v \rightarrow P$ communicates the message v on channel c and then behaves like P. Its symmetrical is process $c?x : T \rightarrow P(x)$, which is ready to communicate any value of $x \in T$ and then behave like $P(x)$.

For example, every DVD player offers several commands to the user, such as play, pause and stop.

$$\begin{aligned} DVD_ACTION(play) &= doPlay \rightarrow SKIP \\ DVD_ACTION(pause) &= doPause \rightarrow SKIP \\ DVD_ACTION(stop) &= doStop \rightarrow SKIP \\ DVD_PLAYER &= c?x : T \rightarrow \\ & \quad DVD_ACTION(x); \\ & \quad DVD_PLAYER \end{aligned}$$

Typically, this functionality is accessed through a remote control.

$$\begin{aligned} DVD_REMOTE = \; & c!play \rightarrow DVD_REMOTE \mid \\ & c!pause \rightarrow DVD_REMOTE \mid \\ & c!stop \rightarrow DVD_REMOTE \end{aligned}$$

Both processes, the DVD player and the remote communicate on a channel called c of type $T = \{play, pause, stop\}$. Hence, the communication between both processes is expressed by:

$$DVD = DVD_REMOTE \;{}_c||_c\; DVD_PLAYER$$

In CSP it is said that a process P is refined by process Q when the relation $Q \sqsubseteq P$ is asserted. There are two types of refinement: traces and failures. On one

hand, trace refinement ($Q \sqsubseteq_T P$) determines if process P at most, engages in the same sequence of events as process Q does. On the other hand, failure refinement ($Q \sqsubseteq_F P$) determines if process P, at least, engages in the same sequence events as process Q does.

For example, process *Spec* represents the desired behaviour of the communication of the remote control and the DVD player, ensuring that actions invoked in the first process corresponds to actions in the second process.

$$\begin{aligned} Spec \;=\; & c.play \rightarrow doPlay \rightarrow Spec \mid \\ & c.pause \rightarrow doPause \rightarrow Spec \mid \\ & c.stop \rightarrow doStop \rightarrow Spec \end{aligned}$$

Because both tests $Spec \sqsubseteq_T DVD$ and $Spec \sqsubseteq_F DVD$ are asserted as valid, it is possible to ensure that process *DVD* satisfies the specification *Spec*, that is, *DVD* and *Spec* behave in the same way.

6.2 CSP Translation

Once the service is declared in OWL-S following the layout presented in Chapter 5, it is ready for being translated to a set of CSP equations. This section is intended to present the rules for doing such translation, based in previous works of how to translate Universal Modelling Language (UML) specifications and shared variables to CSP [46, 98].

A service has a static and a dynamic representation. On the one hand, the static representation is targeted to assist developers in the implementation stage of a service. It focuses on how the service composed by actions, state variables, events and plugs should be implemented. On the other hand, the dynamic representation of a service is a CSP formalisation of the activities related to the service. The static representation alone only ensures that the service is correctly implemented in terms of deadlock and livelock. Fortunately, due to the refinement technique, it is possible to predict what are the consequences of invoking determined services. These consequences are measured in terms of the activities that the user is engaged in.

6.2.1 Static Representation

The static representation of a service corresponds to a CSP process which schema is plotted in Figure 6.1.

In the static model, users can interact with the service through channels *call* and *return*. The first one serves for invoking the desired action on the service and

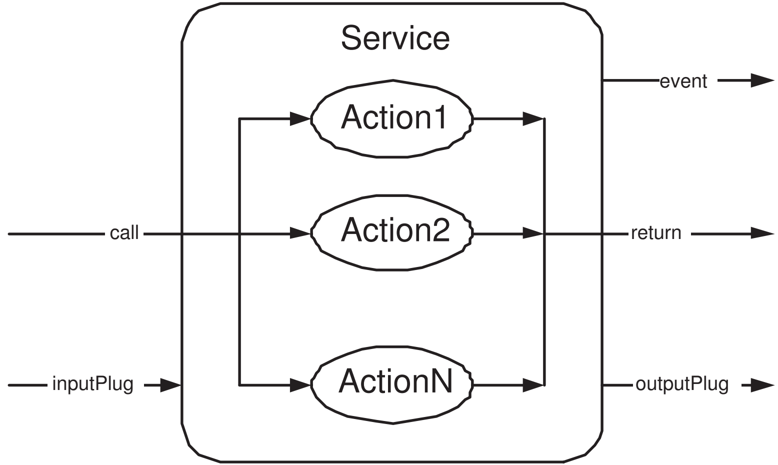


Figure 6.1: Static Representation of a LufService

sending the required input parameters. The return channel is used to retrieve the output results. Changes on the state may be communicated on channel *event*.

As an example, the washing machine service presented in figure 5.6 can be formalised in CSP with the following set of equations.

$$
\begin{aligned}
&PossibleWashingStatus ::= WM_WASHING \mid WM_STOPPED \\
&PossibleDoorStatus \quad ::= WM_OPEN \mid WM_CLOSE
\end{aligned}
$$

$$
\begin{aligned}
&VOCABULARY ::= action \mid event \mid parametervalue \\
&ACTION \quad ::= opendoor \mid closedoor \mid startwashing \mid stopwashing \\
&EVENT \quad ::= eventwashingstatus \mid eventdoorstatus
\end{aligned}
$$

$$
\begin{aligned}
WM_CALL_PROTOCOL &= \{(action, x) \mid x \in ACTION\} \\
WM_RETURN_PROTOCOL &= \{(action, x) \mid x \in ACTION\} \\
WM_EVENT_PROTOCOL &= \{(x, y, (z, w)) \mid x \in \{event\}, \\
&\quad y \in \{eventwashingstatus\}, \\
&\quad z \in \{parametervalue\}, \\
&\quad w \in PossibleWashingStatus\} \\
&\cup \\
&\quad \{(x, a, (z, b)) \mid x \in \{event\}, \\
&\quad a \in \{eventdoorstatus\}, \\
&\quad z \in \{parametervalue\}, \\
&\quad b \in PossibleDoorStatus\}
\end{aligned}
$$

$$
\begin{array}{l}
WM(ws, ds) = \\
\quad wm_call.(action, startwashing) \rightarrow \\
\quad (wm_event!(event, eventwashingstatus, \\
\quad\quad (parametervalue, WM_WASHING)) \rightarrow \\
\quad\quad Skip < ws \notin \{WM_WASHING\} > Skip); \\
\quad wm_return.(action, startwashing) \rightarrow \\
\quad WM(WM_WASHING, ds) \mid \\
\quad wm_call.(action, stopwashing) \rightarrow \\
\quad (wm_event!(event, eventwashingstatus, \\
\quad\quad (parametervalue, WM_STOPPED)) \rightarrow \\
\quad\quad Skip < ws \notin \{WM_STOPPED\} > Skip); \\
\quad wm_return.(action, stopwashing) \rightarrow \\
\quad WM(WM_STOPPED, ds) \mid \\
\quad wm_call.(action, opendoor) \rightarrow \\
\quad (wm_event!(event, eventdoorstatus, \\
\quad\quad (parametervalue, WM_OPEN)) \rightarrow \\
\quad\quad Skip < ds \notin \{WM_OPEN\} > Skip); \\
\quad wm_return.(action, opendoor) \rightarrow \\
\quad WM(ws, WM_OPEN) \mid \\
\quad wm_call.(action, closedoor) \rightarrow \\
\quad (wm_event!(event, eventdoorstatus, \\
\quad\quad (parametervalue, WM_CLOSE)) \rightarrow \\
\quad\quad Skip < ds \notin \{WM_CLOSE\} > Skip); \\
\quad wm_return.(action, closedoor) \rightarrow \\
\quad WM(ws, WM_CLOSE)
\end{array}
$$

For each state variable, a parameter for the main process is generated. The CSP type used over each parameter is derived from the *process:parameterType* property of each state variable. In this case, the enumerated classes *PossibleWashingStatus* and *PossibleDoorStatus* are mapped to the *PossibleWashingStatus* and *PossibleDoorStatus* CSP declarations. In order to interact with the service, the protocol of *wm_call* channel is the desired action to invoke followed by its parameters, those are, the OWL-S inputs of the actions. The same reasoning is applied for the return protocol, with the outputs of the action. Again, the CSP types are derived from the *process:parameterType* property of inputs and outputs. In this case, no inputs or outputs are specified. Every time that the state changes a message with the new value will be output on channel *wm_event*. Other processes interested on this event should listen on this channel.

The alphabets of channels, *wm_call*, *wm_return* and *wm_event* are lists of elements delimited by parentheses. These data structures are called S-Expressions [93] and are used in the Lisp programming language and as mark-up in communications protocols like IMAP. The use of S-Expressions allows seamless translation of data represented in the ontology [69] to CSP_M (CSP_M is the machine readable version of CSP which is the input to the model checker tools). The principal

advantage of this approach is the capability of extremelly simple devices (devices with low computational resources) of enganging in a very flexible communication protocol. For example, it is well known the flexibility of XML. However, it is also known the demanding computational resources in order to parse XML messages. The approach of S-Expressions tries to combine the flexibility of XML but with a less complex parsing schema.

The translation of an OWL-S declaration to a set of S-Expressions is simple. Every OWL-S declaration can be decomposed as a set of triples, since OWL-S is based on RDF. For each triple, a tuple is generated, which is a S-Expression itself. As said before, the protocol over channels are derived from the *process:parameterType* property of OWL-S parameters (inputs, outputs, state variables, events and plugs). We propose that each OWL-S parameter should be mapped to a tuple (a S-Expression) derived from the decomposition in triples of the argument hold by property *process:parameterType*. For example, figure 6.2 shows the OWL-S declaration for event *EventWashingStatus* of the washing machine service presented in figure 5.6.

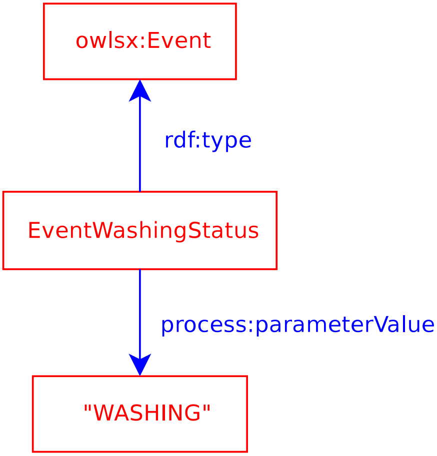


Figure 6.2: EventWashingStatus Diagram

The schema plotted in figure 6.2 can be represented with the following set of triples.

$$(rdf : type, EventWashingStatus, owlsx : Event)$$
$$(process : parameterValue, EventWashingStatus, WM_WASHING)$$

These triples are translated the following member of set $WM_EVENT_PROTOCOL$, which is the alphabet of channel wm_event.

$$(event, eventwashingstatus, (parametervalue, WM_WASHING))$$

In this example, triples of the form $(rdf : type, x, y)$ are translated to S-Expressions of the form (y, x). Triples of the form $(p \neq rdf : type, s, o)$ are translated to the S-Expression $(t, s, (p, o))$ where t is the type of s, that is, the triple $(rdf : type, s, t)$ is true. To generalise the procedures of translating from RDF triples to S-Expressions and vice-versa, we propose the following grammar.

$$
\begin{aligned}
< individual > &= (< type >, < name >, [< properties >]) \\
< properties > &= < property >, [< properties >] \\
< property > &= (< propertyName >, < propertyValue >) \\
< propertyValue > &= < string > | < individual > \\
< type > &= < string > \\
< name > &= < string > \\
< propertyName > &= < string > \\
< string > &= \text{any word consisting of letters, numbers, and special} \\
&\quad\ \text{characteres}
\end{aligned}
$$

This grammar is the output of a function f which translates a RDF triple to a S-Expression.

$$
\begin{aligned}
f(rdf : type, x, y) &= (y, x) \\
&\quad \text{if } x \text{ is of type } y \\
f(p, s, o) &= (f(rdf : type, t, s), (p, f(o))) \\
&\quad \text{if } s \text{ is of type } t \text{ and } p \neq rdf : type \\
f(o) &= o \qquad \text{if } o \text{ belongs to an enumerated class} \\
f(o) &= (f(rdf : type, o, r), \{(p_i, f(v_i))\}_{i=1 \ldots N}) \\
&\quad \text{if } o \text{ is of type } r \text{ and there exist } N \\
&\quad \text{triples of the form } (p_i, o, v_i) \text{ where } i = 1 \ldots N
\end{aligned}
$$

6.2.2 Dynamic Representation

The set of activities related to a service represents its dynamic part. Service activities are modelled with a transition diagram. In CSP, these diagrams are displayed as processes as it is presented in figure 6.3.

Figure 6.3: CSP Activity Process Diagram

Each activity communicates its state (*ACTIVE*, *SUSPENDED* or *STOPPED*) on channel *activity*. The transitions between activity states are governed by a set of rules called triggers. In this thesis, only triggers related with state variables are

considered for a CSP translation. Accordingly, activities are modelled as processes listening on the event channels. Activity processes have a state which is the set composed of the state variables which are the triggers of the activity considered.

As an example, let us consider the activities related to the washing machine service presented in figure 5.6. In this case, a new activity is introduced related to the task of collecting the laundry from the washing machine called *CollectLaundry*. These activities depends on the values of the state variables *WashingStatus* and *DoorStatus*. In this case, the activities of the washing machine are modelled as one process with a state composed with the values of the state variables *WashingStatus* and *DoorStatus*.

$$\begin{array}{ll} ACTIVITY & ::= washinglaundry \mid collectlaundry \\ ACTIVITYSTATUS & ::= ACTIVE \mid SUSPENDED \mid STOPPED \end{array}$$

$$\begin{array}{l} WM_Activities(WM_STOPPED, WM_CLOSE) = \\ \quad wm_event!(event, eventdoorstatus, (parametervalue, WM_OPEN)) \rightarrow \\ \quad activity.collectlaundry.ACTIVE \rightarrow \\ \quad WM_Activities(WM_STOPPED, WM_OPEN) \mid \\ \quad wm_event!(event, eventwashingstatus, (parametervalue, WM_WASHING)) \rightarrow \\ \quad activity.washinglaundry.ACTIVE \rightarrow \\ \quad WM_Activities(WM_WASHING, WM_CLOSE) \end{array}$$

$$\begin{array}{l} WM_Activities(WM_WASHING, WM_CLOSE) = \\ \quad wm_event!(event, eventdoorstatus, (parametervalue, WM_OPEN)) \rightarrow \\ \quad activity.collectlaundry.ACTIVE \rightarrow \\ \quad WM_Activities(WM_WASHING, WM_OPEN) \mid \\ \quad wm_event!(event, eventwashingstatus, (parametervalue, WM_STOPPED)) \rightarrow \\ \quad activity.washinglaundry.STOPPED \rightarrow \\ \quad WM_Activities(WM_STOPPED, WM_CLOSE) \end{array}$$

$$\begin{array}{l} WM_Activities(WM_STOPPED, WM_OPEN) = \\ \quad wm_event!(event, eventdoorstatus, (parametervalue, WM_CLOSE)) \rightarrow \\ \quad activity.collectlaundry.STOPPED \rightarrow \\ \quad WM_Activities(WM_STOPPED, WM_CLOSE) \mid \\ \quad wm_event!(event, eventwashingstatus, (parametervalue, WM_WASHING)) \rightarrow \\ \quad activity.washinglaundry.ACTIVE \rightarrow \\ \quad WM_Activities(WM_WASHING, WM_OPEN) \end{array}$$

$$
\begin{aligned}
&WM_Activities(WM_WASHING, WM_OPEN) = \\
&\ wm_event!(event, eventdoorstatus, (parametervalue, WM_CLOSE)) \rightarrow \\
&\ activity.collectlaundry.STOPPED \rightarrow \\
&\ WM_Activities(WM_WASHING, WM_CLOSE) \mid \\
&\ wm_event!(event, eventwashingstatus, (parametervalue, WM_STOPPED)) \rightarrow \\
&\ activity.washinglaundry.STOPPED \rightarrow \\
&\ WM_Activities(WM_STOPPED, WM_OPEN)
\end{aligned}
$$

The dynamic perspective of the washing machine service is offered by process *WM_Activities*. This process communicates with process *WM* by means on channel *wm_event* and broadcast activity status through channel *activity*.

6.2.3 Service Refinement

Refinement is the technique which permits to check of whether a process satisfies a specification. In this proposal, the refinement procedure is used to verify if particular actions taken on a service correspond with the desired activities. This For example, the refinement can be used to check if every time that the action *startwashing* is invoked on service *WM* the activity *WashingLaundry* is active.

Firstly, the whole service specification is constructed as the combination of the static and dynamic parts.

$$
\begin{aligned}
&WM_Init = \\
&\ WM(WM_STOPPED, WM_CLOSE) \\
&\ [\![\, wm_event \,]\!] \\
&\ (activity.washinglaundry.STOPPED \rightarrow \\
&\ activity.collectlaundry.STOPPED \rightarrow \\
&\ WM_Activities(WM_STOPPED, WM_CLOSE)) \\
&\ \setminus \{wm_event\}
\end{aligned}
$$

The communication of events *activity.washinglaundry.STOPPED* and *activity.collectlaundry.STOPPED* is intended to broadcast the initial state of the activities. The following equation is a process called *WashingLaundry_Start* which invokes action *startwashing*.

$$
\begin{aligned}
&WashingLaundry_Start = \\
&\ wm_call!(action, startwashing) \rightarrow \\
&\ wm_return?(action, startwashing) \rightarrow \\
&\ STOP
\end{aligned}
$$

For being executed action *startwashing*, processes *WM* and *WashingLaundry_Start* should be synchronised on channels *wm_call* and *wm_return*.

$$
\begin{aligned}
&WashingLaundry_StartWM = \\
&\ WM_Init \\
&\ [|\{wm_call, wm_return\}|] \\
&\ WashingLaundry_Start
\end{aligned}
$$

The specification of process *WashingLaundry_StartWM* declares that the final state of activity *WashingLaundry* should be $ACTIVE$. This behaviour is expressed by means of process *WashingLaundry_StartSpec*.

$$
\begin{aligned}
&WashingLaundry_StartSpec = \\
&\ activity.washinglaundry.STOPPED \rightarrow \\
&\ activity.collectlaundry.STOPPED \rightarrow \\
&\ activity.washinglaundry.ACTIVE \rightarrow STOP
\end{aligned}
$$

The first two events in *WashingLaundry_StartSpec* correspond to the initial state of the activities. Finally, the refinement tests needed to prove true are:

$$
\begin{aligned}
&WashingLaundry_StartSpec \sqsubseteq_T WashingLaundry_StartWM \setminus TOTAL \\
&WashingLaundry_StartSpec \sqsubseteq_F WashingLaundry_StartWM \setminus TOTAL
\end{aligned}
$$

where $TOTAL$ represents the set of CSP events that are hidden. In this case, the representation of set $TOTAL$ is:

$$TOTAL = \{wm_call, wm_return\}$$

Because both test are asserted to true, it is possible to confirm that the result of invoking action *startwashing* is to set the state of activity *WashingLaundry* to $ACTIVE$.

Refinement tests can be combined to check more complex services. For example, imagine an intelligent system that has decided the following sequence of activities. First, starting activity *CollectLaundry* to introduce new clothes onto the washing machine. Later, stopping activity *CollectLaundry* by closing the washing machine door and later, starting activity washing machine. This behaviour is expressed in CSP by means of process *Full_WashingSpec*.

$$
\begin{aligned}
&Full_WashingSpec = \\
&\ activity.washinglaundry.STOPPED \rightarrow \\
&\ activity.collectlaundry.STOPPED \rightarrow \\
&\ activity.collectlaundry.ACTIVE \rightarrow \\
&\ activity.collectlaundry.STOPPED \rightarrow \\
&\ activity.washinglaundry.ACTIVE \rightarrow \\
&\ STOP
\end{aligned}
$$

The implementation for this specification involves opening and later closing the washing machine door. Once the door is closed, the washing machine can start washing the laundry. These actions are carried out by process *Full_Washing*.

$$
\begin{array}{l}
Full_Washing = \\
\quad wm_call!(action, opendoor) \rightarrow \\
\quad wm_return?(action, opendoor) \rightarrow \\
\quad wm_call!(action, closedoor) \rightarrow \\
\quad wm_return?(action, closedoor) \rightarrow \\
\quad wm_call!(action, startwashing) \rightarrow \\
\quad wm_return?(action, startwashing) \rightarrow \\
\quad STOP
\end{array}
$$

In order to execute the actions declared in process *Full_Washing*, this process should be synchronised with process *WM_Init* on channels *wm_call* and *wm_return*.

$$
\begin{array}{l}
Full_WashingWM = \\
\quad WM_Init \\
\quad |[\{wm_call, wm_return\}]| \\
\quad Full_Washing
\end{array}
$$

In this case, the tests we needed to prove are:

$$
\begin{array}{l}
Full_WashingSpec \sqsubseteq_T Full_WashingWM \setminus TOTAL \\
Full_WashingSpec \sqsubseteq_F Full_WashingWM \setminus TOTAL
\end{array}
$$

Process *Full_WashingWM* satisfies the specification *Full_WashingWMSpec* because both tests are asserted to true.

The refinement technique can be used to detect that a process does not satisfy a specification. Imagine that process *Full_Washing* is changed to produce process *Full_WashingFail*.

$$
\begin{array}{l}
Full_WashingFail = \\
\quad wm_call!(action, opendoor) \rightarrow \\
\quad wm_return?(action, opendoor) \rightarrow \\
\quad wm_call!(action, closedoor) \rightarrow \\
\quad wm_return?(action, closedoor) \rightarrow \\
\quad wm_call!(action, stopwashing) \rightarrow \\
\quad wm_return?(action, stopwashing) \rightarrow \\
\quad STOP
\end{array}
$$

In this case, process *Full_WashingFail* does not satisfy the specification *Full_WashingSpec*.

This is because action *stopwashing* is invoked instead action *startwashing*. Because of this, activity *WashingLaundry* is never started, so the specification is not satisfied.

This chapter has proposed a scheme for meeting the requirement of reliability. This requirement is addressed by means of a formal method, which allow the identification of system errors during the design stage, and the prediction of system behaviour. Next chapter extends this formality introducing how the high level services specifications in OWL-S and CSP can be implemented in a CSP based communication system.

Chapter 7

Implementations of Home Appliance Services

This chapter presents how a service interface can be implemented taking into account not only the technology used, but also the network protocol used to facilitate service communication. Firstly, different solutions to implement CSP specifications are presented, focusing on how channels and process are represented. Secondly, since service implementations can not be completely abstracted from the underlying network, a grounding ontology based on the UDP protocol is introduced. Finally, the washing machine service presented in previous chapters is implemented in Java following its CSP specification.

7.1 CSP implementations

Since CSP was first described by Hoare, it has evolved considerably. As a consequence, several projects have produced tools to implement systems directly in a CSP style. This is the case of the occam language, the Java APIs JCSP and CTJ, and the C++ library CSP++. The rest of the section is intended to provide an overview of these CSP implementations.

7.1.1 occam

Occam [26] (it should be written always in lowercase, occam) is an imperative procedural language originally designed for programming in the INMOS transputer microprocessors. Fortunately, nowadays there exist occam compilers for other platforms such as Linux.

One of the principal characteristics of occam is that indentation and formatting are crucial for parsing the code. CSP Process operators such as sequence (;),

parallel (‖) and choice (|) have their equivalence in occam with the reserved words SEQ, PAR and ALT respectively.

Occam processes communicate through channels. The nomenclature is the same as in CSP, $c!v$ and $c?v$ stands for sending and receiving message v on channel c respectively.

One of the principal advantages of occam is that it is possible to achieve lightweight implementations. Occam works as an assembler language for transputer microprocessors. Particularly interesting is the 2.5 version of occam known as Kent Retargettable occam Compiler (KRoC) [107]. KRoC was developed by the University of Kent and offers new features such as recursion, mobile channels and the possibility of interconnecting process over a TCP/IP network [97]

7.1.2 Java CSP (JCSP)

The Java CSP (JCSP) library [12, 105], developed by the university of Kent, provides concurrency and communication primitives that allow the construction of CSP-based programs within a single Java Virtual Machine.

Since JCSP is a Java API, all CSP elements are introduced as classes. Each process is encapsulated on an object that implements the *CSProcess* interface. A JCSP process is implemented as an independent thread of execution. The thread scheduling is done by the Java Virutal Machine (JVM), which makes the use of JCSP in real applications dependant on the operating system.

Communication between processes is achieved by channel objects, which act as data pipes between threads. Any message is communicated by reference but integer types. There are one to one, one to many, many to one and many to many channels. Several buffering policies can be specified on channels such as infinite or zero buffering. JCSP processes can be combined in a CSP style by means of classes, Sequence, Parallel, and Alternative for CSP operators (;), (‖) and (|).

JCSP is a free library for academic purposes. There also exists a commercial version called JCSP Network edition [106] which allows communication between processes residing in different virtual machines. This commercial edition supports the principal network protocols, such as TCP/IP, USB, IEEE 1394 and IEEE 1355.

7.1.3 Communicating Threads for Java (CTJ)

CTJ [6, 61] is a Java extension for supporting real time systems. This API was developed by the University of Twente and has many similarities with JCSP. In CTJ, processes are encapsulated as objects which may synchronise on channels

objects. Like the other CSP implementations, CJT processes can be composed in sequence, parallel and choice.

In contrast to to JCSP, CTJ integrates its own real-time kernel, not relying thread scheduling to the JVM. In CTJ, channels are always any to any and messages are passed by value, instead by reference. This message passing policy makes communication between processes independent if the channel is based on shared memory or a network protocol.

By default, CJT channels use shared memory as the communication mechanism. The default communication policy under a channel can be modified by specifying and object of class *LinkDriver* on the channel constructor. The standard CTJ distribution offers several implementations of *LinkDriver* class that support communication between TCP and UDP networks.

7.1.4 C++ CSP

C++CSP [39] is a library which provides an Object Oriented API similar to JCSP but with a fast kernel support. C++CSP supports channels, processes, parallel communication and alternatives.

C++ CSP gives support to the concept of mobile objects. This feature is supported by means of a “smart pointer” class that ensures that there exists only one reference to the mobile object. This technique gives the illusion that the object “moves” between processes.

Distributed applications can be implemented in C++CSP, as well. There exists a C++CSP network edition [40], which supports TCP/IP networking and it is based on KRoC.Net.

7.1.5 Handle C

Handle-C [9] is a programming language based on C which provides concurrency and communication based on CSP. Handle-C programs are designed to be compiled and mapped to reconfigurable hardware platforms such as FPGAs.

7.2 CSP Service Implementation

In service architectures, especially the one treated in this thesis, services are abstractions for accessing device functionality. Sometimes, a service may be the device itself, such as a service for switching on or off the light represented in CSP and implemented directly in hardware. However, it most of the cases, services and devices represent different entities. In this case, a service implementation requires

the consideration of the software and hardware characteristics of the represented device, such as operating system, memory, or power consumption.

In order to homogenise all device characteristics, it is assumed that all devices functionality can be accessed by a library of functions that can be called from C or Java code. In this case, a service implementation is like a gateway to access a device library, as it is plotted in figure 7.1

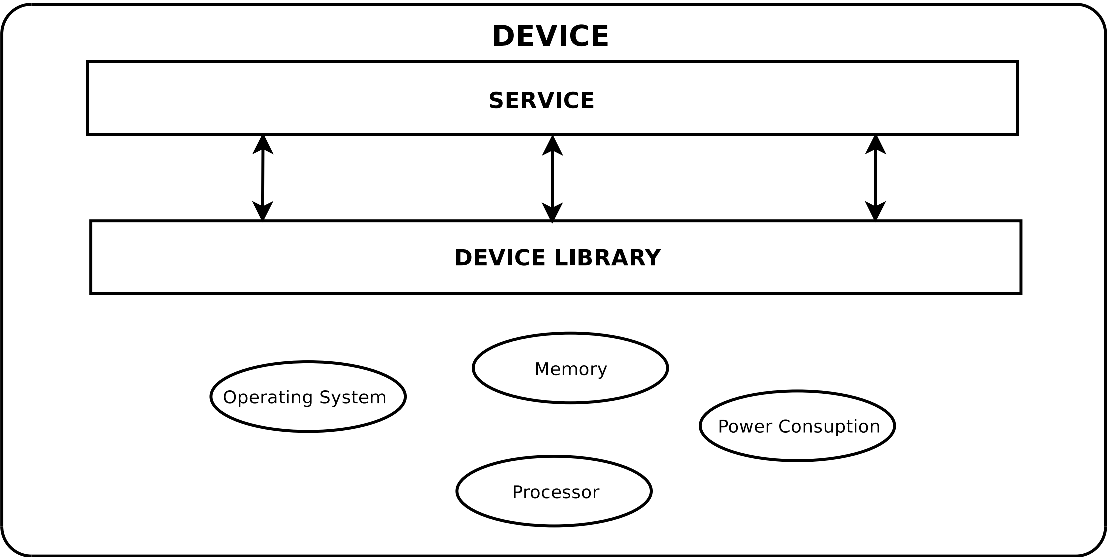


Figure 7.1: Relation between Services and Devices

With this assumption, the rest of the section is intended to present how a service should be implemented. This procedure is explained by the implementation in JCSP of the washing machine service presented in previous chapters.

7.2.1 JCSP Service Implementation

The following section is a simplification of how a service should be implemented. Topics such as the service registration and the interaction service-device are omitted to keep concepts clear.

The first aspect to consider is that it is only needed to implement the service static part. Each service is modelled as a process providing three channels, *call*, *return* and *event*. Plugs are not implemented as JCSP channels. How plugs are implemented is presented in next sections.

In JCSP, a process is represented by a class which implements the interface *CSProcess*. In this case, the implementation for the washing machine should be started with the lines of code presented in listings 7.1.

```
1  public class WashingMachineService implements CSProcess{
2     private ChannelInput callChannel;
3     private ChannelOutput returnChannel;
4     private ChannelOutput eventChannel;
5
6     private String washingstatus = "WM_STOPPED";
7     private String doorstatus = "WM_CLOSE";
8
9     public WashingMachineService(
10          ChannelInput callChan,
11          ChannelOutput returnChan,
12          ChannelOutput eventChan,
13    ){
14          callChannel = callChan;
15          returnChannel = returnChan;
16          eventChannel = eventChan;
17    }
18
19    public void run() {
20       // Body of the service
21    }
22 }
```

Listing 7.1: Washing Machine service declaration in JCSP

Channels *call*, *return* and *event* are represented by objects *callChannel*, *returnChannel* and *eventChannel*. *ChannelInput* and *ChannelOutput* define the interfaces for reading and writing messages on channels respectively. The use of interfaces avoids giving details of how channels are implemented. This abstraction makes the presented service declaration independent of the mechanism for channel communication.

The body of the service is encapsulated in the method *run*, which is declared in interface *CSProcess*. The string attributes *washingstatus* and *doorstatus* represent the *WashingStatus* and *DoorStatus* state variables, respectively. Both variables are initialised to their default values.

Process *WashingMachineService* should be responsible for waiting on object *callChannel* for action requests. According to the received action, the service may change some state variable values and, as a consequence, write on object *eventChannel* such changes. Finally, the service should communicate the ending of the requested action and its returning values on object *returnChannel*. This behaviour is codified in the method *run* as it is shown in listings 7.2.

```
1  public void run (){
2    while (true) {
3      String action = callChannel.read().toString();
4      StringTokenizer st = new StringTokenizer(action, "() \n");
5      if (st.countTokens() == 2) {
6        String actionType = st.nextToken();
7        String actionName = st.nextToken();
8        if (actionType.equals(
9         "http://www.lboro.ac.uk/owlsextension.owl#Action")) {
10         if (actionName.equals(base + "StartWashing")) {
11           if (!washingstatus.equals("WM_WASHING") {
12             // Take actions on the device
13             washingstatus = "WM_WASHING";
14             eventChannel.write(washingstatus);
15           }
16           returnChannel.write(action);
17         }else if (actionName.equals(base + "StopWashing")){
18           if (!washingstatus.equals("WM_STOPPED") {
19             // Take actions on the device
20             washingstatus = "WM_STOPPED";
21             eventChannel.write(washingstatus);
22           }
23           returnChannel.write(action);
24         }else if (actionName.equals(base + "OpenDoor")){
25           if (!doorstatus.equals("WM_OPEN") {
26             // Take actions on the device
27             doorstatus = "WM_OPEN";
28             eventChannel.write(doorstatus);
29           }
30           returnChannel.write(action);
31         }else if (actionName.equals(base + "CloseDoor")){
32           if (!doorstatus.equals("WM_CLOSE") {
33             // Take actions on the device
34             doorstatus = "WM_CLOSE";
35             eventChannel.write(doorstatus);
36           }
37           returnChannel.write(action);
38         }
39       }
40     }
41 }
```

Listing 7.2: Washing Machine service run method

The *run* method consists on an infinite loop waiting for reading action requests on object *callChannel*. Actions are codified as S-Expressions, so the way of parsing them is to decompose each S-Expression as a set of tokens. This task is done by the standard class *StringTokenizer* of the *java.util* package. Once the correct action is determined by the set of conditional statements, the service takes the corresponding actions on the device by using its corresponding API. After this

operation, the service actualises the corresponding state variable and writes its new value on object *eventChannel*.

7.2.2 Channel Implementation

The previous section has introduced how to implement a service. However, the presented implementation does not specify how channels are instantiated. By default, the JCSP standard API provides classes for instantiating channels objects which use shared memory as the communication policy. However, in a real ubiquitous computing scenario, devices may not reside in the same machine, therefore, it may be impossible for them to share memory. Consequently, an underlying network is needed to facilitate device communication.

As was introduced in chapter 3, there exist many networks for device communication, ranging from specific home automation networks to well known Internet protocols. In this work, the network protocol chosen to evaluate service intercommunication is UDP. The nature of a home network suggests the use of a lightweight protocol such as UDP. In fact, UDP shares common features with another efficient protocols such as IEEE 1355, which has been proposed as solution for inter-device communication [86]. The UML diagram plotted in figure 7.2 shows how UDP

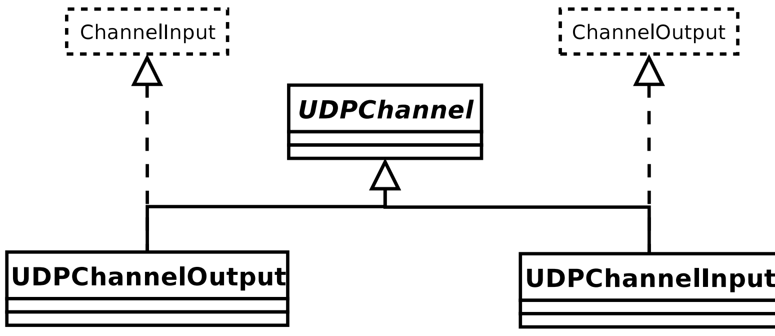


Figure 7.2: UDP Channel Implementation

channels are implemented. The management of the UDP protocol and channel synchronisation is based on the UDP link driver offered by CTJ and it is encapsulated in the abstract class *UDPChannel*. Channel instances should be created by means of classes *UDPChannelInput* and *UDPChannelOutput* which implement the *ChannelInput* and *ChannelOutput* interfaces respectively.

In UDP, applications are identified by an IP address and a port number. Coincidentally, we propose the use of an IP address per service and a port number per input channel in a service. This means that for a given service the *call* channel is identified by the service host name and a port number. Output channels do not have any port assigned because UDP is based on datagrams and not in

connections. In UDP, when one application needs to send a message to a receiver application, the first one will output a datagram which includes the host and port number of the remote application. Because UDP is connectionless, there is no need to establish a connection between the output and input applications, which involves the use of port numbers in both sides.

To manage output communication on a device, each service should be run in parallel with a communication manager process. The communication manager of each service is responsible for relating each *return* and *event* channel to input ports in other services, when required. For each output channel, the communication manager keeps a list with the destination ports. Every time that a new message is written on an output channel, the connection manager broadcasts the message to all the ports contained in the corresponding list. Each communication manager has an input channel called *connect* which serves to request a communication between and output channel in the managed service and a remote port.

For example, let us imagine that the washing machine service implemented in the previous section resides on a device with a host name called “washingmachine.home”. Channels *call* and *connect* are represented by objects *callChannel* and *connectChannel* and assigned to ports 50000 and 50001, respectively. The piece of Java code that instantiates the service is presented in listings 7.3.

```
final int callPort = 50000;
final int connectionPort = 50001;
UDPChannelInput callChannel = new UDPChannelInput(callPort);
One2OneChannel returnChannel = new One2OneChannel();
Any2OneChannel eventChannel = new Any2OneChannel();
UDPChannelInput connectChannel = new UDPChannelInput(
    connectionPort);

WashingMachineService washingMachineService =
    new WashingMachineService(callChannel,
                              returnChannel,
                              eventChannel);
WashingMachineCM washingMachineCM =
    new WashingMachineCM(connectChannel,
                         returnChannel,
                         eventChannel);

Parallel par = new Parallel();
par.addProcess(washingMachineService);
par.addProcess(washingMachineCM);
par.run();
```

Listing 7.3: Washing Machime instantation

7.3 Service Grounding

OWL-S was originally designed to provide an ontological perspective of web services. In a similar way that the OWL-S class *service:Grounding* gives details of how a web service should be accessed, it is needed a grounding class which indicates how to interact with the services presented in this thesis. There needs to develop a set of classes which indicate how channels *call*, *return* and *event* are accessed.

The first step to develop a service grounding is to determine what the mechanism for accessing a service is. In the standard OWL-S ontology there exists class *grounding:WSDLGrounding* [1], which is a specification of how OWL-S services with WSDL interfaces should be reached. In this work, UDP is the network protocol chosen.

7.3.1 UDP Service Grounding

UDP utilises ports to allow application-to-application communication. For each input channel we propose the use of an UDP port. Therefore, the main purpose of the service grounding is to indicate the UDP ports for each channel.

As is shown in figure 7.3, for each service there exists an instance of class *owlsx:LufGrounding*. The meat of the process grounding is represented by class *owlsx:LufProcessGrounding* which is attached to class *owlsx:LufGrounding* by means of property *owlsx:hasLufProcessGrounding*. The data needed to access a channel implemented with the UDP protocol is encapsulated with class *owlsx:UDPChannel*. This class has only two properties *owlsx:host* and *owlsx:port*, which represent the IP address of the machine hosting the device and the port in which the channel is reading data.

[1] grounding is the namespace for `http://www.daml.org/services/owl-s/1.1/Grounding.owl`

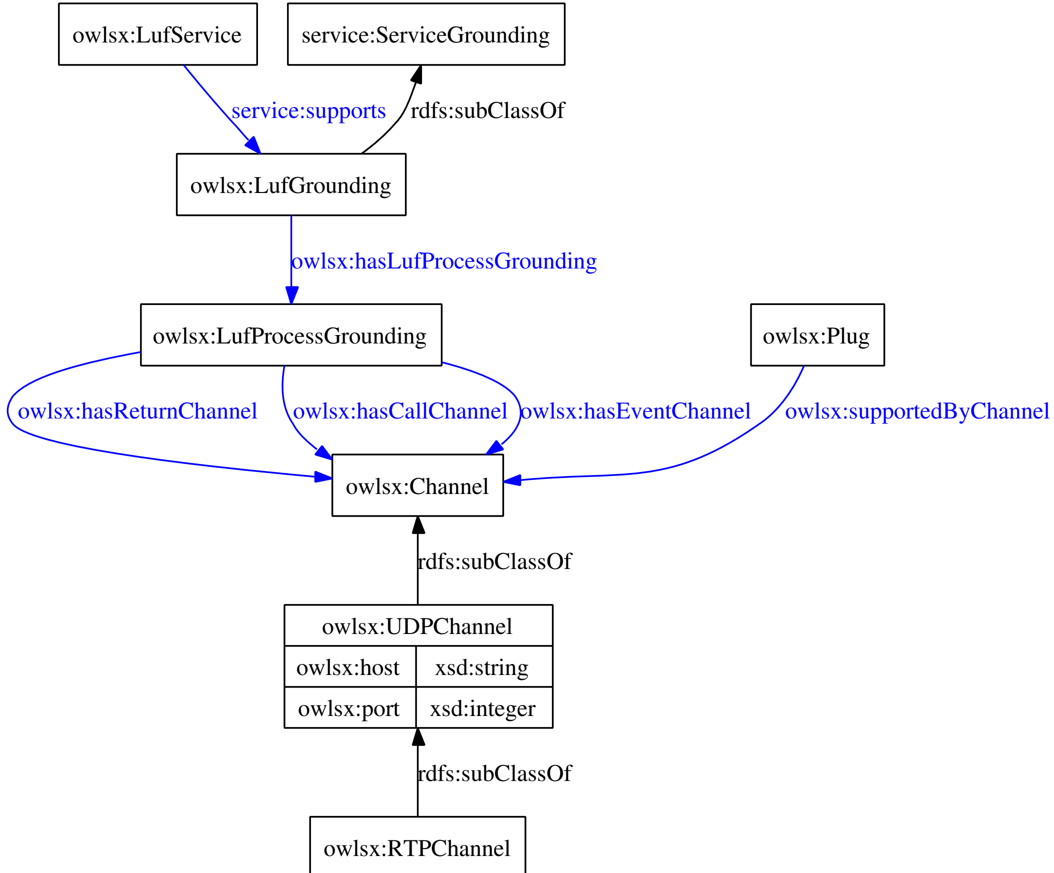


Figure 7.3: LufGrounding Class

In this thesis and for simulation purposes, plugs are implemented using the Java Media Framework (JMF) [13] which allows the programming of multimedia applications. The communication of audio video frames between devices is achieved through the Real Time Transport Protocol (RTP) [83] which relies on UDP. For this reason, class *owlsx:RTPChannel* has been introduced. In addition, a plug is related with a channel through property *owlsx:supportedByChannel*.

At the end of this chapter, the framework for implementing smart home services is presented. Firslty, chapter 5 has introduced how an ontology is used to model the semantics of home devices funtionality. In chapter 6 it was indicated how this semantic description of home devices can be translated to the process algebra CSP. This chapter has introduced how these CPS specifications can be implemented in Java. In order to offer a proof of concept of this framework, next chapter presents a simulator in which several service composition are presented.

Chapter 8
Examples of Home Appliance Services

Previous chapters have introduced device services represented in OWL-S capable of being composed (by choreography or orchestration) to offer valuable advantages in every day life for home residents. In addition, such representation is derived from the formal algebra CSP, which allows developers to implement device services in a systematic way. This chapter is intended to work as a proof of concept of such device representation.

Firstly, a service orchestration node for service composition is presented. This node is called central node and is responsible for the registration of new services and context information and for the composition and execution of new services. Secondly, a simulator hosting several devices and capable of generating user context information is introduced.

Finally, several home subsystems implemented with the proposed theory are analysed in the presented simulator. All of these examples meet the requirements for an smart home, preserving the ad-hoc nature of the approach of this thesis. The choosen examples shown are ordered in ascending order accordingly to their ad-hoc level. In light of this, the first example is an HVAC system in which all participating services are registered when the example is started. The second and the third examples include a simulated home user that moves between different rooms, discovering new devices and increasing the ad-hoc nature of the example. Finally, the last example is totally ad-hoc, due to the ”hot” connection and detection of new devices in the home network.

8.1 Simulator

The simulator is a Java based application for reproducing a home environment. The interaction between the simulator and the developer is achieved by a graphical

user interface. As shown in figure 8.1, the left hand side of the interface reproduces a house plan consisting on a living room, a home office, a master bedroom, a bathroom, a small bedroom for children, a kitchen and a corridor. All rooms are equipped with windows and doors and may contain several devices which are represented as services. The right hand side serves to monitor and control some context information related to environmental variables such as temperature or noise.

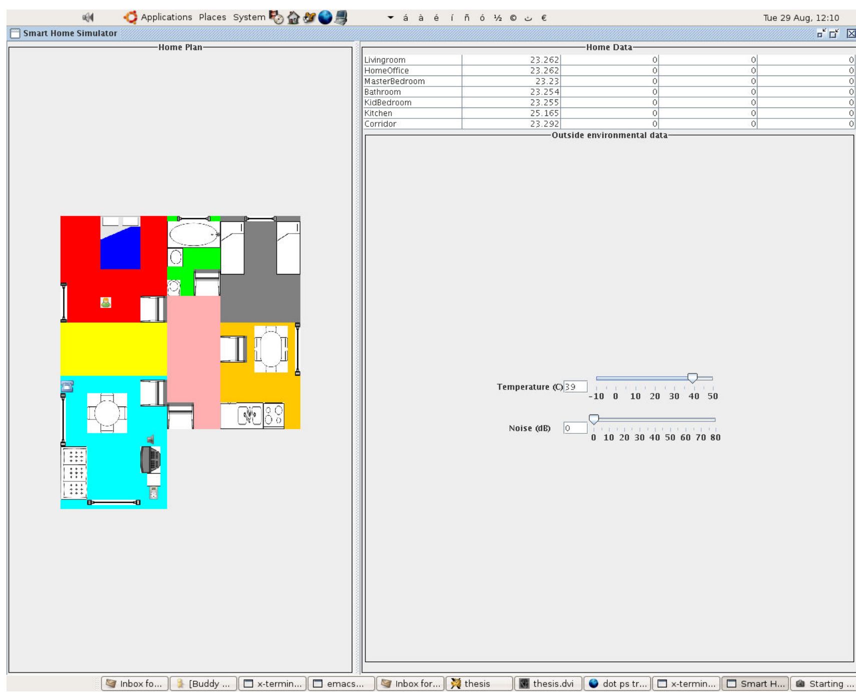


Figure 8.1: Simulator

The person using the simulator can generate context information related to user location and device services. The home occupant is represented by an icon which can be translated among rooms by using the second button of the mouse. Device functionality can be accessed making a single click with the mouse on the desired device, as it is shown in figure 8.2 for the Washing Machine device.

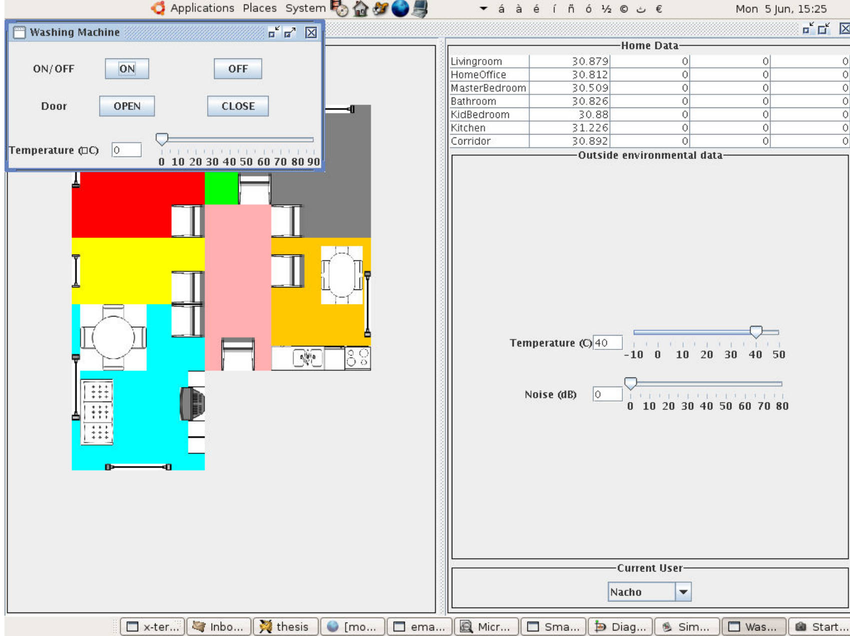


Figure 8.2: Washing Machine Interface

Each device in the simulator hosts at least one service representing its functionality. Service orcherstration is achieved by another Java application called the central node, which is described in the next section. In order for the central node to be aware of the context information generated in the simulator, each service connects its *event* channel to the *registryEvent* channel in the central node as it is shown in figure 8.3. In this case, actions in the simulator will be transformed in useful context information in the central node. With this information, the central node deduces the state of the activities in which the virtual home resident is engaged, determines a set of goals for the user and executes the set of actions needed to achieve such goals.

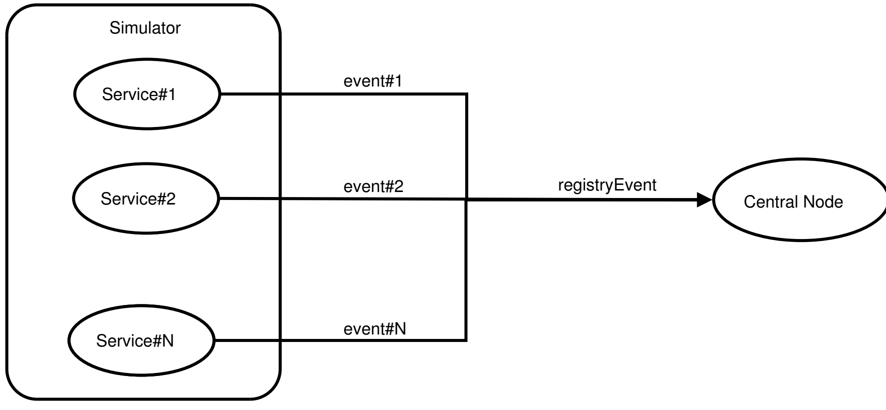


Figure 8.3: Simulator and Central Node Connection

8.2 Orchestration Node

Device services exhibit their functionality as a set of state variables, events, actions, plugs and activities. This information is needed in order to combine the execution of several actions to achieve a predetermined goal.

In service orchestration, which is usually used in private environments, a central node takes control over the device services and coordinates the execution of different actions on the services involved in the operation. To achieve this task, the central node should be aware of not only the services deployed in the home environment, but also the context information. Accordingly, we propose a central node which acts as a service register, service composer and context information register.

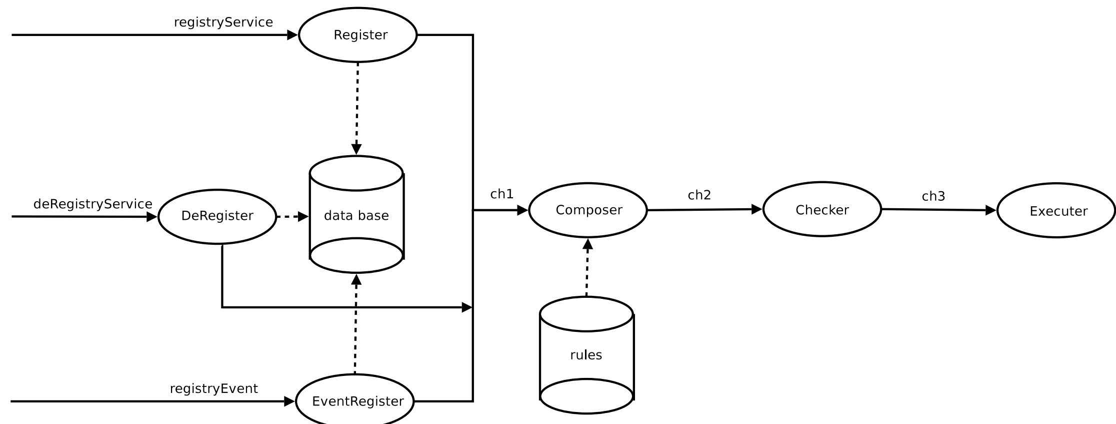


Figure 8.4: Central Node Process Diagram

Figure 8.4 displays the CSP-process model of the central node. The *Register* process is in charge of listening on channel *registryService* for new services ready to check. For services to be deregistered there exists a channel *deRegistryService* and a *DeRegister* process. The *Register* process integrates an OWL-S to CSP parser in order to check if the service is free of deadlock and livelock. Once the service has been proved safe, it is stored in the database and a signal to process *Composer* is send through channel $ch1$. The *Composer* process integrates a rule based system implemented in the Java Expert System Shell (Jess) [14], which is in charge of orchestrate registered services. Jess is arule engine and scripting language written entirely in the Java language. Jess supports the development of rule-based expert systems which can be tightly coupled to code written in the Java language.

Service composition is governed by a set of pre-programmed rules. These rules are derived from the ontological description of services and, it is presented in next sections, they allow a goal oriented approach for service composition. The services generated by the *Composer* process are sent to process *Checker* to prove

that the new services satisfy a set of specifications attached to each service. If a service satisfies the specification, is sent to the *Executer* process for its execution. Changes in state variables and context information are registered in the central node by process *EventRegister*. This process waits for new events on channel *registryEvent* for being stored in the data base. Each time that a new event is received, the *Composer* starts the reasoning engine to find out new services.

8.2.1 Service Composition

The meat of the node is inside process *Composer*, which has an rule-based engine embedded. This engine is implemented in Jess and is fed with the available services, context information, and a set of rules for composing services.

The basic idea is to manage the information about services and context information both, with a forward and backward engine.

Firstly, the information about the relation between actions and state variables is codified as a rule which has as the antecedent the invocation of the action and as a consequent, the change in the involved state variable. For example, in the washing machine service the relation between action *StartWashing* and the state variable *WashingStatus* is codified with rule $startwashingrule - 1$.

$$\begin{array}{l} startwashingrule - 1 = \\ (process : process, ?perform, StartWashing) \\ \Rightarrow \\ (process : parameterValue, WashingStatus, WM_WASHING) \end{array}$$

With rule $startwashingrule{-}1$, every time that action *StartWashing* is invoked, the value of the state variable *WashingStatus* will be set to $WM_WASHING$. In the ontology, this rule is attached to action *StartWashing* by means of an instance of class $process : Result$, as it is indicated in figure 8.5.

By reversing $startwashingrule - 1$ rule, it is also possible to obtain a kind of goal system capable of inferencing that *StartWashing* is the needed action to set to $WM_WASHING$ the state variable *WashingStatus*. This relation can be indicated with rules $startwashingrule - 2$ and $startwashingrule - 3$.

$$\begin{array}{l} startwashingrule - 2 = \\ (need - triple(process : parameterValue, WashingStatus, WM_WASHING)) \\ \Rightarrow \\ (need - triple(process : process, ?perform, StartWashing)) \end{array}$$

$$
\begin{aligned}
& startwashingrule - 3 = \\
& (need - triple(process : process, ?perform, StartWashing)) \\
& \Rightarrow \\
& executeAction(StartWashing)
\end{aligned}
$$

Basically, function *executeAction* places its argument, which is an action, in a queue which will be sent to the *Executer* process. All actions in this queue will be composed in a sequence process.

The derivation of these set of rules done for the washing machine service is done for all actions attached to a service. This task is done by process *Register* for each new service. With this information the system will always know which actions to invoke in order to set the desired value for an state variable.

The same reasoning is applicable for activities and triggers as well. Firstly a forward reasoning can be applied to inference the state of the activity based on the value of the triggers. For example, for activity *WashingLaundry* of the washing machine service, it will be needed to generate the following set of rules.

$$
\begin{aligned}
& washinglaundry - start - 1 = \\
& (process : parameterValue, WashingStatus, WM_WASHING) \\
& \Rightarrow \\
& (owlsx : activitystatus, WashingLaundry, ACTIVE)
\end{aligned}
$$

StartWashing
process:hasResult
StartWashingResult
process:hasEffect
StartWashingEffect
expr:expressionBody | (process:parameterValue, WashingStatus, WM_WASHING)
expr:expressionLanguage
owlsx:Jess

Figure 8.5: StartWashing Effect

$$washinglaundry - stop - 1 = \\ (process : parameterValue, WashingStatus, WM_STOPPED) \\ \Rightarrow \\ (owlsx : activitystatus, WashingLaundry, STOPPED)$$

Both rules, $washinglaundry - start - 1$ and $washinglaundry - stop - 1$ indicate that if the value of state variable *WashingStatus* is *WM_WASHING*, activity *WashingLaundry* is active and if the value is *WM_STOPPED* the activity is stopped. The name of activity triggers take more sense with this point of view, because they are the conditions for firing the rules which will change activity status.

By reversing these rules, is possible to obtain a backward version, as well.

$$washinglaundry - start - 2 = \\ (need - triple(owlsx : activitystatus, WashingLaundry, ACTIVE)) \\ \Rightarrow \\ (need - triple(process : parameterValue, WashingStatus, WM_WASHING))$$

$$washinglaundry - stop - 2 = \\ (need - triple(owlsx : activitystatus, WashingLaundry, STOPPED)) \\ \Rightarrow \\ (need - triple(process : parameterValue, WashingStatus, WM_STOPPED))$$

With these new rules, it is possible to establish a system is which goals are established as states in activities. For example, it is possible to inference that *StartWashing* is the needed action to active *WashingLaundry* activity. When fact

$$need - triple(owlsx : activityStatus, StartWashing, ACTIVE)$$

is inserted in the knoledge base, then rule $washinglaundry - start - 2$ will be fired, inserting in the engine fact

$$need - triple(process : parameterValue, WashingStatus, WM_WASHING)$$

which will fire rule $startwashingrule - 2$, which will fire rule $startwashingrule - 3$, which will perform action $executeAction(StartWashing)$ which will execute action *StartWashing*

Accordingly, in order to set an activity to a desired status, the only needed procedure is to insert in the knowledge base a fact of the form:

$$need - triple(owlsx : activityStatus, ?targetActivity, ?targetStatus)$$

To that end, we have created three functions, *activeActivity*, *suspendActivity* and *stopActivity* which set to *ACTIVE*, *SUSPENDED* and *STOPPED* the status of the activity which is the argument of these functions.

Based on this idea, this analysis is done by process *Register* for each new process. The rules that govern service composition are designed to manage the state of activities based on context information. The result of firing these rules is a chain of actions composed as a sequence of processes and executed by the *Executer* process.

8.3 Examples of Home Device Services

The following section presents and HVAC System, an AV System and a Communication System. Each system comprises a set of devices with their respective services and activities associated and a set of rules stored in the central node to manipulate such services. All systems are tested with the simulator to check if the approach presented in this thesis is capable of meet the requirements for a smart home introduced in chapter 1. Before results from examples are presented, it is possible to advance that horizontal integration is achieved by the implementation as independent process of each service offered by each device. The focus of the examples is in showing how our approach satisfies reliability (HVAC system example), and hence the low cost requirement, and spontaneuos collaboration and design for domestic use (AV and communication systems examples)

Because of performance issues, the *Checker* process is disabled, so services generated for the central node are executed directly, without being tested with the refinement technique. This is because the time needed to do a model checking analysis is extremely high for an ubiquitous application. Overcoming this problem will be a future line of research beyond this project.

8.3.1 HVAC System

HVAC (heating, ventilation and air-conditioning) generally refers to the climate control of a building. The HVAC System considered in this section is based on the UPnP standard for HVAC systems [29] and is responsible for heating and cooling the home plan presented in the simulator section, involving the following devices:

- **Heat Pump**. It is the source of heat or cold. There is only one for all rooms and it can be in one of three modes: heating, cooling, or off.

- **Valve**. It is responsible for controlling when the warm or cold air enters in a room. There is only one valve per room. A valve is allowed to hold one of these two values: close and open.

- **Temperature Sensor**. Is responsible for measuring the temperature of a room. There is one per room.

- **Temperature Selector**. There is a temperature selector device per room to allow the user to set the desired temperature.

Semantic Representation of an HVAC System

Each of the devices formerly presented hosts a service to accesses its functionality. The *HeatPumpService* resides in the Heat Pump device. The service has a state variable called *PumpMode* which can have only there values: *HP_HEATING*, *HP_COOLING* and *HP_OFF*. This state variable is accessed by actions *StartHeating*, *StartCooling* and *Stop*.

The *ValveControllerService* represents the functionality of the Valve device. The service offers two actions called *CloseValve* and *OpenValve* which set the value of the state variable *ValveState* to *V_CLOSE* and *V_OPEN* respectively. This service has two activities of type *ActuatorActivity* attached, called *Heat_Room* and *Cool_Room*. These activities describe the effects on the environment when they are active. Taking activity *Heat_Room*, the attachments of values *owlsx* : *INCREASE* and *owlsx* : *Temperature* to properties *owlsx* : *hasMode* and *owlsx* : *hasMagnitude*, indicate that whenever this activity is active the temperature will be increased.

The state of activity *Heat_Room* depends on the value of the state variables *PumpMode* and *ValveState*. Table 8.1 shows that every time that the Heat pump

Table 8.1: Heat Room Activity Triggers

ActivityStatus	*ValveState*	*PumpMode*
ACTIVE	*V_OPEN*	*HP_HEATING*
SUSPENDED	*V_CLOSE*	*HP_HEATING*
STOPPED	*V_CLOSE*	*HP_OFF*
	V_OPEN	*HP_OFF*
	V_OPEN	*HP_COOLING*

is in the *HP_HEATING* mode and the corresponding valve is open the activity is activated. The activity is stopped whenever the Heat pump is off or cooling and the valve open. Finally, when the valve is closed the activity is suspended.

The Temperature Sensor device hosts the *TemperatureSensorService* service. It has only one state variable called *CurrentTemperature* which is not controlled by any action. The *TemperatureSensorService* is associated with an activity of type *SensorActivity*. This kind of activity relates a state variable, a location and an environmental magnitude. The semantics given to a *SensorActivity* determines that the state variable involved represents the environmental magnitude on the specified location. For example, figure 8.6 illustrates how activity *MeasuringTemperature* relates the state variable *CurrentTemperature* with the temperature in location *LivingRoom*.

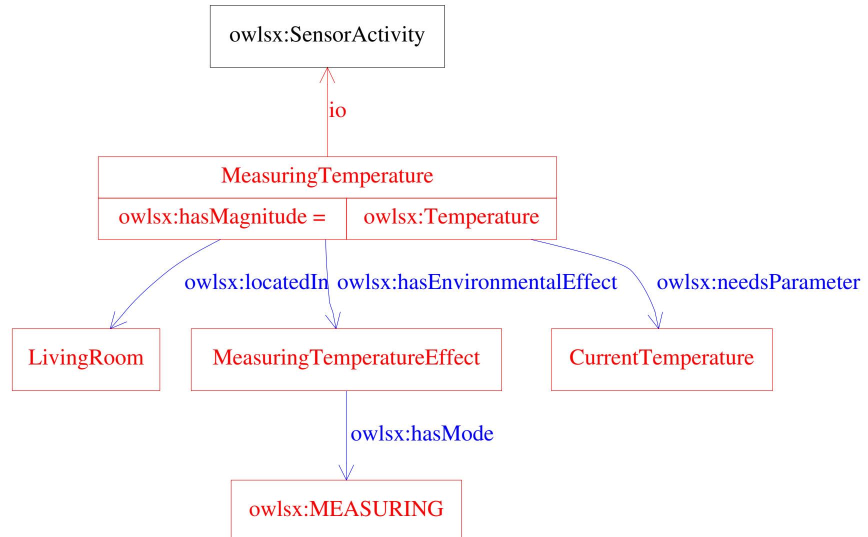


Figure 8.6: Activity associated with measuring the livingroom temperature

The *TemperatureSelectorService* displays the functionality of the temperature selector device. It has a state variable called *TemperatureSetPoint* which is accessed by action *SetTemperatureSetPoint*. This service has attached four activities of type *DiagnosisActivity*, which indicate how the home resident perceives the room temperature. For example, there is an activity called *Room_Slightly_Cold* which specifies that the room temperature is lower than the indicated in the variable *TemperatureSetPoint*. In addition, activity *Room_Cold* communicates that the temperature is two degrees lower than in the variable *TemperatureSetPoint*. Both activities assert that the room is cold. While the first one suggests that the room is a little bit cold, the second one specifies that is too cold. The same reasoning is applied to activities *Room_Slightly_Hot* and *Room_Hot*.

Rules to Control the HVAC System

The HVAC System is controlled by the central node with a set of general purpose rules [52]. For cooling a house there are two rules. The first one is called $decrease-magnitude-1$ and declares that every time that a room is slightly hot and there is

an activity that is increasing the temperature, such activity should be suspended or stopped. The second rule is, $decrease - magnitude - 2$, for starting an activity which decreases the room temperature whenever a room is declared too hot. These two rules are formalised in the following lines of code.

$$
\begin{aligned}
& decrease - temperature - 1 = \\
& (rdf : type, ?activity1, owlsx : DiagnosisActivity) \wedge \\
& (owlsx : activityStatus, ?activity1, owlsx : ACTIVE) \wedge \\
& (owlsx : hasEnvironmentalEffect, ?activity1, ?effect1) \wedge \\
& (owlsx : hasMagnitude, ?effect1, owlsx : Temperature) \wedge \\
& (owlsx : hasMode, ?effect1, owlsx : SLIGHTLY_HIGH) \wedge \\
& (owlsx : locatedIn, ?activity1, ?location) \wedge \\
& (rdf : type, ?activity2, owlsx : ActuatorActivity) \wedge \\
& (owlsx : activityStatus, ?activity2, owlsx : ACTIVE) \wedge \\
& (owlsx : hasEnvironmentalEffect, ?activity2, ?effect2) \wedge \\
& (owlsx : hasMagnitude, ?effect2, owlsx : Temperature) \wedge \\
& (owlsx : hasMode, ?effect2, owlsx : INCREASE) \wedge \\
& (owlsx : locatedIn, ?activity2, ?location) \wedge \\
& \Rightarrow \\
& add - temp - rooms - ok(?location) \\
& suspendActivity(?activity2) \\
& if(is - home - temp - ok)then\ stopActivity(?activity2)
\end{aligned}
$$

$$
\begin{aligned}
& decrease - temperature - 2 = \\
& (rdf : type, ?activity1, owlsx : DiagnosisActivity) \wedge \\
& (owlsx : activityStatus, ?activity1, owlsx : ACTIVE) \wedge \\
& (owlsx : hasEnvironmentalEffect, ?activity1, ?effect1) \wedge \\
& (owlsx : hasMagnitude, ?effect1, owlsx : Temperature) \wedge \\
& (owlsx : hasMode, ?effect1, owlsx : TOO_HIGH) \wedge \\
& (owlsx : locatedIn, ?activity1, ?location) \wedge \\
& (rdf : type, ?activity2, owlsx : ActuatorActivity) \wedge \\
& \neg(owlsx : activityStatus, ?activity2, owlsx : ACTIVE) \wedge \\
& (owlsx : hasEnvironmentalEffect, ?activity2, ?effect2) \wedge \\
& (owlsx : hasMagnitude, ?effect2, owlsx : Temperature) \wedge \\
& (owlsx : hasMode, ?effect2, owlsx : DECREASE) \wedge \\
& (owlsx : locatedIn, ?activity2, ?location) \wedge \\
& \Rightarrow \\
& rem - temp - rooms - ok(?location) \\
& startActivity(?activity2)
\end{aligned}
$$

With these two rules, the temperature of each room oscillates between the value indicated in the state variable *TemperatureSetPoint* plus minus one Celsius degree. The results of firing these are different sequence processes which will set to *HP_COOLING* or *HP_OFF* the heat pump or set to *V_OPEN* and *V_CLOSE* the corresponding valve.

Similarly, to heat a room there exist rules $increase - temperature - 1$ and $increase - temperature - 2$. Another feature of these rules is that they are designed to avoid transitions of the Heat Pump from cooling to heating and vice-versa, which can damage seriously the heat pump.

Formal Specification of the HVAC System

Generally, the presented rules generate services which are responsible to set the correct mode in the heat pump and to open or close the corresponding valves. In this section, the CSP model for these services is presented. Firstly, the models for a Heat Pump and a Valve Controller are presented. Secondly, a service responsible to cool a room is introduced. Finally, the introduced service is studied with the refinement technique.

The HVAC System is composed by several services. However, to keep clear concepts, only the Valve Controller and the Heat Pump are considered.

Firstly, the static representation of the HVAC system comprises the following equations.

$$\begin{aligned}
&PossibleHeatPumpState ::= HP_OFF \mid HP_COOLING \mid HP_HEATING \\
&PossibleValveState \quad ::= V_OPEN \mid V_CLOSE
\end{aligned}$$

$$\begin{aligned}
&VOCABULARY ::= action \mid event \mid parametervalue \\
&V_ACTION \quad ::= openvalve \mid closevalve \mid \\
&HP_ACTION \quad ::= startcooling \mid startheating \mid stop \\
&V_EVENT \quad ::= eventvalvestate \\
&HP_EVEN \quad ::= eventheatpumpstate
\end{aligned}$$

$$\begin{aligned}
V_CALL_PROTOCOL &= \{(action, x) \mid x \in V_ACTION\} \\
V_RETURN_PROTOCOL &= \{(action, x) \mid x \in V_ACTION\} \\
V_EVENT_PROTOCOL &= \{(x, y, (z, w)) \mid \\
&\quad x \in \{event\}, \\
&\quad y \in \{eventvalvestate\}, \\
&\quad z \in \{parametervalue\}, \\
&\quad w \in PossibleValveState\}
\end{aligned}$$

$$\begin{aligned}
HP_CALL_PROTOCOL &= \{(action, x) \mid x \in HP_ACTION\} \\
HP_RETURN_PROTOCOL &= \{(action, x) \mid x \in HP_ACTION\} \\
HP_EVENT_PROTOCOL &= \{(x, y, (z, w)) \mid \\
&\quad x \in \{event\}, \\
&\quad y \in \{eventheatpumpstate\}, \\
&\quad z \in \{parametervalue\}, \\
&\quad w \in PossibleHeatPumpState\}
\end{aligned}$$

For the Heat Pump there exists the *HP* process:

$$
\begin{aligned}
&HP(hps) = \\
&\quad hp_call.(action, startcooling) \rightarrow \\
&\quad (hp_event!(event, eventheatpumpstate, \\
&\qquad (parametervalue, HP_COOLING)) \rightarrow \\
&\qquad Skip < hps \notin \{HP_COOLING\} > Skip); \\
&\quad hp_return.(action, startcooling) \rightarrow \\
&\quad HP(HP_COOLING) \mid \\
&\quad hp_call.(action, startheating) \rightarrow \\
&\quad (hp_event!(event, eventheatpumpstate, \\
&\qquad (parametervalue, HP_HEATING)) \rightarrow \\
&\qquad Skip < hps \notin \{HP_HEATING\} > Skip); \\
&\quad hp_return.(action, startheating) \rightarrow \\
&\quad HP(HP_HEATING) \mid \\
&\quad hp_call.(action, stop) \rightarrow \\
&\quad (hp_event!(event, eventheatpumpstate, \\
&\qquad (parametervalue, HP_OFF)) \rightarrow \\
&\qquad Skip < hps \notin \{HP_OFF\} > Skip); \\
&\quad hp_return.(action, stop) \rightarrow \\
&\quad HP(HP_OFF)
\end{aligned}
$$

And for the Valve, there exists the V process:

$$
\begin{aligned}
&V(vs) = \\
&\quad v_call.(action, openvalve) \rightarrow \\
&\quad (v_event!(event, eventvalvestate, \\
&\qquad (parametervalue, V_OPEN)) \rightarrow \\
&\qquad Skip < vs \notin \{V_OPEN\} > Skip); \\
&\quad v_return.(action, openvalve) \rightarrow \\
&\quad V(V_OPEN) \mid \\
&\quad v_call.(action, closevalve) \rightarrow \\
&\quad (v_event!(event, eventvalvestate, \\
&\qquad (parametervalue, V_CLOSE)) \rightarrow \\
&\qquad Skip < vs \notin \{V_CLOSE\} > Skip); \\
&\quad v_return.(action, closevalve) \rightarrow \\
&\quad V(V_CLOSE)
\end{aligned}
$$

Secondly, only the Valve Controller services has related activities.

$$
\begin{aligned}
&V_Activities(HP_OFF, V_CLOSE) = \\
&\ v_event!(event, eventvalvestate, (parametervalue, V_OPEN)) \rightarrow \\
&\ V_Activities(HP_OFF, V_OPEN) \mid \\
&\ hp_event!(event, eventheatpumpstate, (parametervalue, HP_HEATING)) \rightarrow \\
&\ activity.heatroom.SUSPENDED \rightarrow \\
&\ V_Activities(HP_HEATING, V_CLOSE) \mid \\
&\ hp_event!(event, eventheatpumpstate, (parametervalue, HP_COOLING)) \rightarrow \\
&\ activity.coolroom.SUSPENDED \rightarrow \\
&\ V_Activities(HP_COOLING, V_CLOSE)
\end{aligned}
$$

$$
\begin{aligned}
&V_Activities(HP_OFF, V_OPEN) = \\
&\ v_event!(event, eventvalvestate, (parametervalue, V_CLOSE)) \rightarrow \\
&\ V_Activities(HP_OFF, V_CLOSE) \mid \\
&\ hp_event!(event, eventheatpumpstate, (parametervalue, HP_HEATING)) \rightarrow \\
&\ activity.heatroom.ACTIVE \rightarrow \\
&\ V_Activities(HP_HEATING, V_OPEN) \mid \\
&\ hp_event!(event, eventheatpumpstate, (parametervalue, HP_COOLING)) \rightarrow \\
&\ activity.coolroom.ACTIVE \rightarrow \\
&\ V_Activities(HP_COOLING, V_OPEN)
\end{aligned}
$$

$$
\begin{aligned}
&V_Activities(HP_HEATING, V_CLOSE) = \\
&\ v_event!(event, v_eventvalvestate, (parametervalue, V_OPEN)) \rightarrow \\
&\ activity.heatroom.ACTIVE \rightarrow \\
&\ V_Activities(HP_HEATING, V_OPEN) \mid \\
&\ hp_event!(event, hp_eventheatpumpstate, (parametervalue, HP_COOLING)) \rightarrow \\
&\ activity.coolroom.SUSPENDED \rightarrow activity.heatroom.STOPPED \rightarrow \\
&\ V_Activities(HP_COOLING, V_CLOSE) \mid \\
&\ hp_event!(event, hp_eventheatpumpstate, (parametervalue, HP_OFF)) \rightarrow \\
&\ activity.heatroom.STOPPED \rightarrow \\
&\ V_Activities(HP_OFF, V_CLOSE)
\end{aligned}
$$

$$
\begin{aligned}
&V_Activities(HP_HEATING, V_OPEN) = \\
&\ v_event!(event, eventvalvestate, (parametervalue, V_CLOSE)) \rightarrow \\
&\ activity.heatroom.SUSPENDED \rightarrow \\
&\ V_Activities(HP_COOLING, V_CLOSE) \mid \\
&\ hp_event!(event, eventheatpumpstate, (parametervalue, HP_COOLING)) \rightarrow \\
&\ activity.coolroom.ACTIVE \rightarrow activity.heatroom.STOPPED \rightarrow \\
&\ V_Activities(HP_COOLING, V_OPEN) \mid \\
&\ hp_event!(event, eventheatpumpstate, (parametervalue, HP_OFF)) \rightarrow \\
&\ activity.heatroom.STOPPED \rightarrow \\
&\ V_Activities(HP_OFF, V_OPEN)
\end{aligned}
$$

$$
\begin{aligned}
&V_Activities(HP_COOLING, V_CLOSE) = \\
&\ v_event!(event, eventvalvestate, (parametervalue, V_OPEN)) \rightarrow \\
&\ activity.coolroom.ACTIVE \rightarrow \\
&\ V_Activities(HP_COOLING, V_OPEN) \mid \\
&\ hp_event!(event, eventheatpumpstate, (parametervalue, HP_HEATING)) \rightarrow \\
&\ activity.coolroom.STOPPED \rightarrow activity.heatroom.SUSPENDED \rightarrow \\
&\ V_Activities(HP_HEATING, V_CLOSE) \mid \\
&\ hp_event!(event, eventheatpumpstate, (parametervalue, HP_OFF)) \rightarrow \\
&\ activity.coolroom.STOPPED \rightarrow \\
&\ V_Activities(HP_OFF, V_CLOSE)
\end{aligned}
$$

$$
\begin{aligned}
&V_Activities(HP_COOLING, V_OPEN) = \\
&\ v_event!(event, eventvalvestate, (parametervalue, V_CLOSE)) \rightarrow \\
&\ activity.coolroom.SUSPENDED \rightarrow \\
&\ V_Activities(HP_COOLING, V_CLOSE) \mid \\
&\ hp_event!(event, eventheatpumpstate, (parametervalue, HP_HEATING)) \rightarrow \\
&\ activity.coolroom.STOPPED \rightarrow activity.heatroom.ACTIVE \rightarrow \\
&\ V_Activities(HP_HEATING, V_OPEN) \mid \\
&\ hp_event!(event, eventheatpumpstate, (parametervalue, HP_OFF)) \rightarrow \\
&\ activity.coolroom.STOPPED \rightarrow \\
&\ V_Activities(HP_OFF, V_OPEN)
\end{aligned}
$$

Finally, the refinement technique allows to check if the service is going to behave as expected.

$$
\begin{aligned}
&HVAC_Init = \\
&\ (HP_Init \mid\mid\mid V_Init) \\
&\ [|\{hp_event, v_event\}|] \\
&\ (activity.coolroom.STOPPED \rightarrow activity.heatroom.STOPPED \rightarrow \\
&\ V_ActivitiesInit) \\
&\ \setminus \{hp_event, v_event\}
\end{aligned}
$$

The presented scenario is for a situation in wich is needed to cool a room when the heat pump is in the OFF model and the corresponding valve is closed. The result of firing rules $decrease - temperature - 1$ and $decrease - temperature - 2$ is the sequence of actions $OpenValve$ and $StartCooling$, as it is indicated in process $HVAC_Start_Cooling$.

$$
\begin{aligned}
&HVAC_Start_Cooling = \\
&\ v_call!(action, v_openvalve) \rightarrow \\
&\ v_return!(action, v_openvalve) \rightarrow \\
&\ hp_call?(action, hp_startcooling) \rightarrow \\
&\ hp_return!(action, hp_startcooling) \rightarrow \\
&\ STOP
\end{aligned}
$$

$$
\begin{array}{l}
HVAC_Start_Cooling_System = \\
\quad HVAC_Init \\
\quad [|\{v_call, v_return, hp_call, hp_return\}|] \\
\quad HVAC_Start_Cooling
\end{array}
$$

$$
\begin{array}{l}
HVAC_Start_Cooling_Spec = \\
\quad activity.coolroom.STOPPED \rightarrow \\
\quad activity.heatroom.STOPPED \rightarrow \\
\quad activity.coolroom.ACTIVE \rightarrow \\
\quad STOP
\end{array}
$$

$$
\begin{array}{rcl}
EVENT & = & \{hp_event, v_event\} \\
CALL & = & \{v_call, hp_call\} \\
RETURN & = & \{v_return, hp_return\} \\
TOTAL & = & CALL \cup RETURN \cup EVENT
\end{array}
$$

$$
\begin{array}{l}
HVAC_Start_Cooling_Spec \sqsubseteq_T HVAC_Start_Cooling_System \setminus TOTAL \\
HVAC_Start_Cooling_Spec \sqsubseteq_F HVAC_Start_Cooling_System \setminus TOTAL
\end{array}
$$

The refinement tests are asserted, so the service is going to cool a room. In the definition of $HVAC_Start_Cooling_Spec$, the first to events communicated are the initial values of the activities.

HVAC System Simulation Results

The HVAC system presented has been tested in the simulated home. Figure 8.7 shows how the temperature of each room of the simulated home is controlled by the HVAC system. The plot shows that after the transitory (caused for all the services being registered), the temperature of all home rooms is kept between 20 and 22 Celsius degrees, being the value of the temperature selector 21 degrees.

The rules presented include the use of three functions, $is - home - temp - ok$, $add - temp - rooms - ok$ and $rem - temp - rooms - ok$. The first one returns a true value if all the controlled rooms in the smart home are not with temperature values neither too high nor low. The other two functions serve to monitor how many rooms are in the correct range of temperature. The addition of these functions ensures an efficient use of the HVAC System, being the heat pump in the OFF mode whenever all the temperatures of the home rooms are inside the control values. Table 8.2 plots the amount of time in which the heap pump is in each mode and the average temperature of the home depending on the outside temperature. In addition, it is possible to ensure that there was not any transition between the COOLING to the HEATING mode and vice-versa, which will may damage seriously the heat pump. The analysis of the data presented in table 8.2 shows

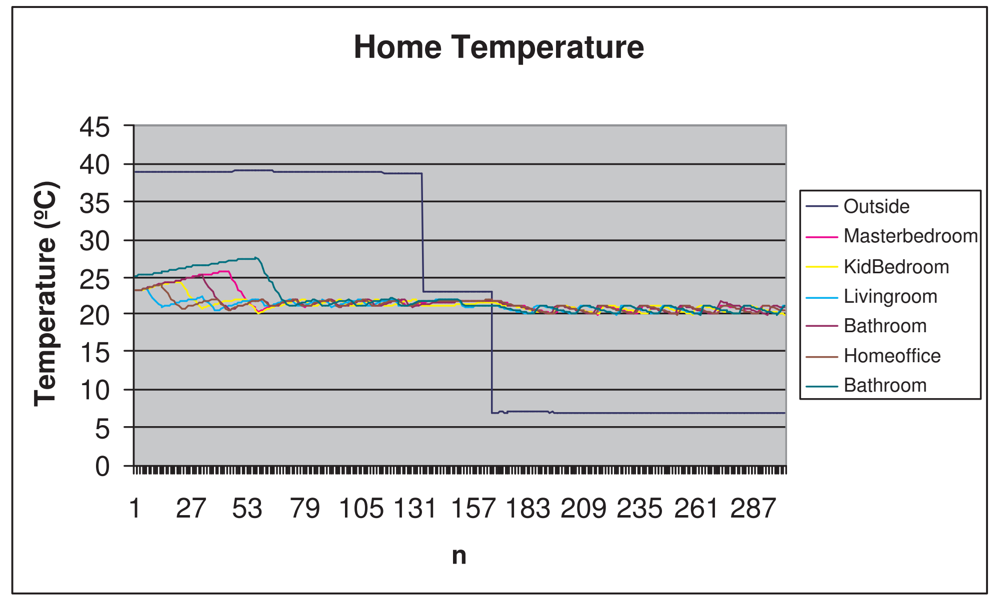


Figure 8.7: Home Temperature Evolution

Table 8.2: Heat Pump Evolution

Outside Temp (°C)	Home Temp (°C)	COOLING	OFF	HEATING
40	21.46	87.51 %	14.29%	0.00%
23	21.55	12.50 %	87.50%	0.00%
7	21.59	0.00 %	57.89%	42.11%

that the closer the outside temperature is to the target temperature (21 °C), the more time the heat pump is in the OFF mode, saving energy and hence, the users money.

The focus of this example is on demostrating how a several devices can cooperate y a safe way. In fact, more that 30 threads run in the same virtual machine without any problem. The example was run for a full night, without any degradation of the system, showing the stability of the approach. The example shows how the incormporation of CSP allows the implementation of stable and reliable systems composed of an important number of threads in a seamless way.

8.3.2 AV System

The Audio Video System presented in this thesis is inspired by UPnP MediaSender and MediaRenderer [15] specifications and it is composed by a DVD_Player, a Screen and an Amplifier.

By default, not all services of this subsystem are registered in the central node. This is indicated with the X icon presented in some devices in the living

room simulated. To register all the services embedded in a device, it is needed to press the second button on the mouse on the device icon and choose de Register option. The same operation is required for deregistering a registered device, but choosing the DeRegister option. This approach is intended to show the ad-hoc nature of the implemented system, showing how new devices are registered on the system.

Semantic Representation of the AV System

The *DVD_Player* hosts a service called *DVD_PlayerService* which offers four operations, *Play*, *Pause* and *Stop* for controlling the state variable *TransportState* and *SetMultimediaItem*, for setting the value of the state variable *CurrentMultimediaItem*, which is of type *MultimediaItem*. The service offers two output plugs, one for video and other for audio. This service has two activities related, *WatchingVideo* and *ListeningAudio*, one for the task of watching a video and another for listening audio. The state of both activities depends on the value of the state variable *TransportState* and whether the output plugs are connected or not, as it is shown in table 8.3. These activities, *WatchingVideo* and *ListeningAudio*, have several

Table 8.3: State Diagram for Activity WatchingVideo

ActivityStatus	*TransportState*
ACTIVE	*PLAYING*
SUSPENDED	*PAUSED*
STOPPED	*STOPPED*

effects in the user. Figure 8.8 indicates that activity *WatchingVideo* requires the sight sense from the user, while activity *ListeningAudio* requires the hearing sense. The other two devices, the Screen and the Amplifier have attached two services, *ScreenService* and *AmplifierService*. The first service offers an input plug of type *VideoType* while the second one an input plug of type *AudioType*. These services do not have any activity attached to them.

Rules to Control the AV System

The principal rules for controlling the AV System are intended to establish connections between compatible plugs. To that end, rule $connect - plug$ connects plugs only if they have the same type.

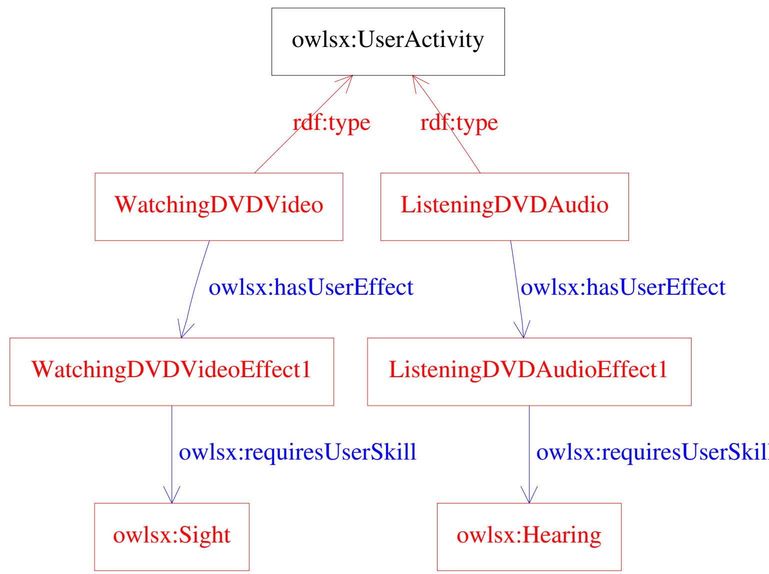


Figure 8.8: Activities Related with the DVD Service

$$
\begin{aligned}
& connect - plug = \\
& \ (service : describedBy, ?inputService, ?inputProcess) \wedge \\
& \ (owlsx : hasInputPlug, ?inputProcess, ?inPlug) \wedge \\
& \ (owlsx : hasContentType, ?inputPlug, ?contentT) \wedge \\
& \ (service : presents, ?inputService, ?inputProfile) \wedge \\
& \ (owlsx : hasLocation, ?inputProfile, ?location) \wedge \\
& \ (owlsx : isLocatedIn, ?user, ?location) \wedge \\
& \ (service : describedBy, ?outputService, ?outputProcess) \wedge \\
& \ (owlsx : hasInputPlug, ?outputProcess, ?outPlug) \wedge \\
& \ (owlsx : hasContentType, ?outputPlug, ?contentT) \wedge \\
& \ (owlsx : communicatesOnPlug, ?multimediaItem, ?outputPlug) \wedge \\
& \ (owlsx : hasContentType, ?multimediaItem, ?contentT) \wedge \\
& \ \Rightarrow \\
& \ bind(?binding, concat(?inPlug, ?outPlug, ?contentT, \text{“}InputPlugB''') \\
& \ (rdf : type, ?binding, owlsx : InputPlugBinding) \\
& \ (process : toParam, ?binding, ?inPlug) \\
& \ bind(?valueOf, concat(\text{“}ValueOf'', ?outPlug) \\
& \ (process : theVar, ?valueOf, ?outPlug) \\
& \ (process : valuesource, ?binding, ?valueOf)
\end{aligned}
$$

The presented rule introduces context information since not only are compatible plugs for establishing a connection needed, but also the presence of the user in the same location as the input device. Services and locations are related by means of the service profile, through property *owlsx:hasLocation*. For example, there is not sense in establishing a connection between the DVD and the Screen if the user is not in the same place as the Screen. In addition, compatible plugs are determined by the value of state variable *CurrentMultimediaItem*, which is linked with an output plug through property $owlsx : communicatesOnPlug$. The

result of the rule is an instance of class *process* : *Binding* which represents the connection between the two plugs.

Formal Representation of the AV System

The formal specification of the AV System is rather more complicated than the other systems because the triggers of the related activities depends, not only on state variables, but also on connections. For that reason, the only formal model for this system corresponds to the static view of the services involved, those are, the *DVD_PlayerService*, the *ScreenService* and the *AmplifierService*.

AV System Simulation Results

The rule that governs the establishment of connections considers the type of the multimedia file selected in the DVD Service. Given an user in the same location as the screen and the amplifier device, when an item composed of audio and video is selected, the central node will establish two connections, one between the DVD and the amplifier for the audio and another between the DVD and the screen for the video. This case of use is presented in figure 8.9.

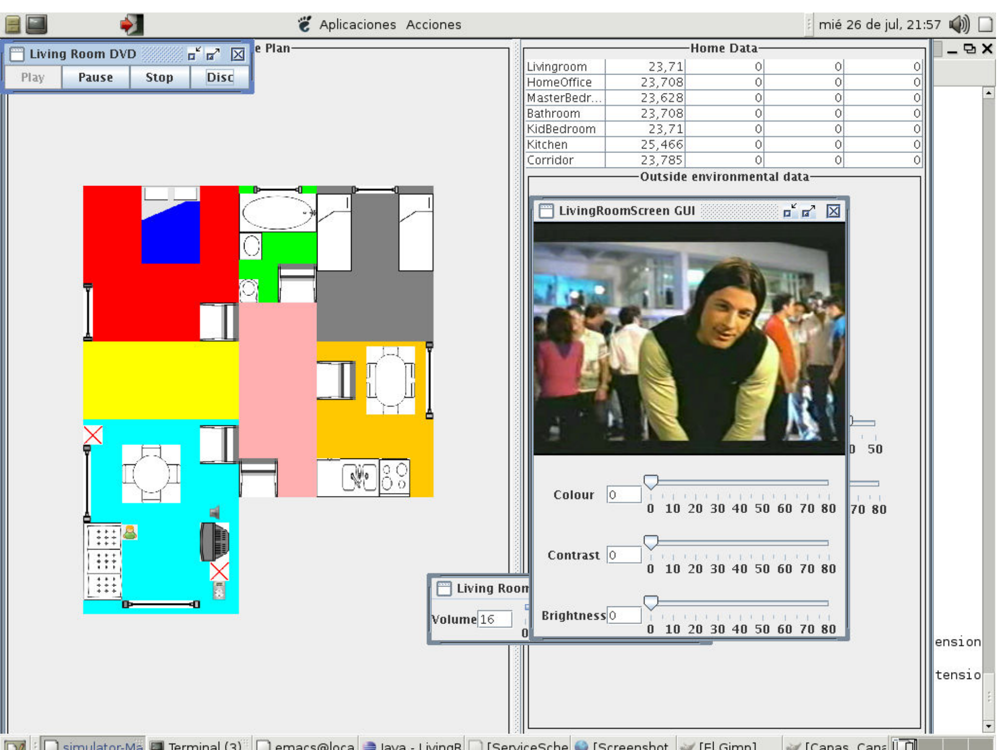


Figure 8.9: Simulation of the AV System

The ad-hoc nature of this example comes from two different sources. Firstly, every time that the user changes its location, new devices are available for offering

useful services. Secondly, due to the ad-hoc nature of the PDA introduced, the example shows how new devices are detected and as a consecuence, how new services offered. Opposite, the example shows how services are removed every time that some of its components disappear from the home network.

8.3.3 Communication System

The communication system is simple and it is formed only by one device which corresponds with a telephone. The purpose of this system is to show how different systems can interact between them.

Semantic Representation of the Communication

A service called *PhoneService* with all the functionality is hosted by the device. This service has a state variable called *PhoneState* which is controlled through two operations *Answer* and *Hangup*, to set its value to *SPEAKING* and *IDLE* respectively.

Only one activity is associated with this service, called *UserSpeaking*. This activity is of type *UserActivity* and requires the hearing sense of the user whenever is active.

Rules for Controlling the Communication System

In this thesis, only one rule is dedicated to control the communication system. This rule is intended to schedule incompatible activities. We have defined that two activities are incompatible when they require the same skill from the user, that is, they have the same value for property *owlsx* : *requiresUserSkill*. The intention of this policy is to adapt activities to user context following ubiquitous computing patterns [71, 5].

Particularly, whenever two active activities require the hearing sense from the user, the implemented rule will suspend the oldest one, that is, the activity which was active before.

$$
\begin{aligned}
&incompatible - activities = \\
&(owlsx : activityStatus, ?activity1, ACTIVE) \wedge \\
&(owlsx : hasUserEffect, ?activity1, ?userEffect1) \wedge \\
&(owlsx : requiresUserSkill, ?userEffect1, HEARING) \wedge \\
&(owlsx : activityStatus, ?activity2, ACTIVE) \wedge \\
&(owlsx : hasUserEffect, ?activity2, ?userEffect2) \wedge \\
&(owlsx : requiresUserSkill, ?userEffect2, HEARING) \wedge \\
&isOlder(?activity1, ?activity2) \\
&\Rightarrow \\
&suspendActivity(?activity1)
\end{aligned}
$$

The consequence of rule $incompatible - activities$ is the suspension of the oldest activity which is determined with the function $isOlder$.

Formal Representation of the Communication System

The CSP specification of service *PhoneService* and its related activity follows the same patterns as the previous systems presented in this chapter.

Simulation Results of the Communication System

The simulation of the communication system shows its interaction with the AV System. As a scenario, an activity *ListeningAudio* is started. From the schema plotted in figure 8.8, this activity requires the hearing sense from the user. When this activity is active, action *Answer* is performed on service *PhoneService* which starts activity *UserSpeaking*. This event fires rule $incompatible - activities$ which suspends activity *ListeningAudio* by invoking action *Pause* on service *DVD_Service*. Figure 8.10 illustrates this situation, showing how the *PhoneService* is ready to be hang up and the *DVD_Service* is waiting for button play to be pressed. The rule works in such as way that in the opposite situation, when the *UserSpeaking* is activated before activity *ListeningAudio*, the first one is not suspended, since this activity does not offer a suspend trigger.

8.3.4 More Examples. Ac-Hoc Composition

Previous examples may seem a little bit static. To emphasise the ad-hoc character of this work, another scenario is presented, involving mobile devices.

Let us imagine a home resident watching his favourite DVD film in his living room alone. Before the end of the film, the user decides to go to bed, leaving the living room and heading to the master bedroom. The system detects that the user is not in the same location as the screen and hence, pauses the DVD by ordering to suspend activity *WatchingDVD*. When the user arrives at the master bedroom, he changes his mind and decides to keep watching the previous film,

but on his PDA. To that end, he switches on his PDA, which encapsulates two services, one called *PortableScreen* with an input plug of type video, and another called *PortableSpeaker* with an input plug of type audio. Subsequently, the system discovers the PDA and, by means of rule $connect - plug$, two connections will be established between the DVD and the PDA, one for video and another for audio. In addition, the system is aware that the connections between the DVD, the Screen and the Amplifier are not necessary, so it disconnects these services. Now the user is ready to continue watching the film in his PDA.

To achieve this goal, a couple of new rules are needed.

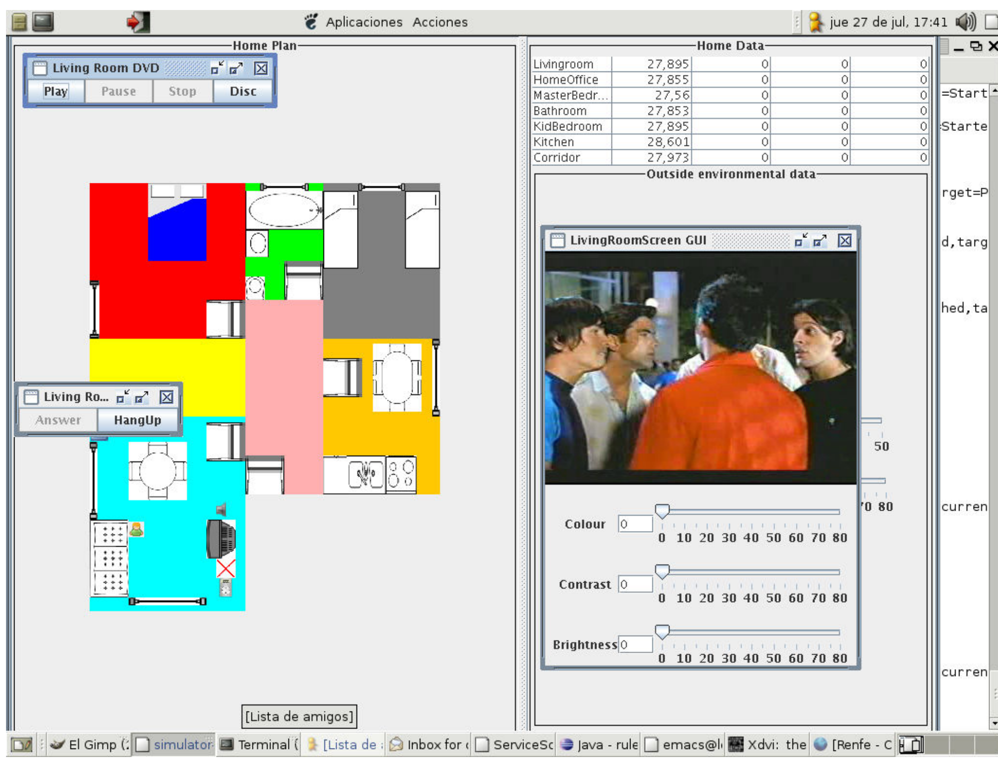


Figure 8.10: Simulation of the Communication System

$$
\begin{array}{l}
incompatible - connection = \\
(owlsx : isLocatedIn, ?user, ?userLoc) \wedge \\
(rdf : type, ?activity, owlsx : UserActivity) \wedge \\
(owlsx : needsParameter, ?activity, ?multimediaItem) \wedge \\
(owlsx : communicatesOnPlug, ?multimediaItem, ?outputPlug) \wedge \\
(process : theVar, ?valueOf, ?outputPlug) \wedge \\
(process : valueSource, ?outputPlug, ?oldBinding)) \\
(process : toParam, ?oldBinding, ?oldInputPlug) \wedge \\
(owlsx : hasInputPlug, ?oldInputProcess, ?oldInputPlug)) \\
(service : describedBy, ?oldInputService, ?oldInputProcess) \wedge \\
(service : presents, ?oldInputService, ?oldInputProfile) \wedge \\
(owlsx : hasLocation, ?oldInputProfile, ?oldLoc) \wedge \\
(process : toParam, ?newBinding, ?newInputPlug) \wedge \\
(owlsx : hasInputPlug, ?newInputProcess, ?newInputPlug)) \\
(service : describedBy, ?newInputService, ?newInputProcess) \wedge \\
(service : presents, ?newInputService, ?newInputProfile) \wedge \\
(owlsx : hasLocation, ?newInputProfile, ?userLoc) \\
\Rightarrow \\
deleteConnection(?oldBinding)
\end{array}
$$

$$
\begin{array}{l}
incompatible - activities - 2 = \\
(rdf : type, ?activity, owlsx : UserActivity) \wedge \\
(owlsx : activityStatus, ?activity, ACTIVE) \wedge \\
(owlsx : hasUserEffect, ?activity, ?userEffect) \wedge \\
(\ (owlsx : requiresUserSkill, ?userEffect, HEARING) \vee \\
\ \ (owlsx : requiresUserSkill, ?userEffect, SIGHT)) \\
(owlsx : locatedIn, ?activity, ?activityLoc) \wedge \\
isEmpty(?activityLoc) \\
\Rightarrow \\
suspendActivity(?activity)
\end{array}
$$

The first rule $incompatible - connection$ is in charge of deleting a connection when it is not needed. This is the case of the connection between the DVD and the Screen when the user is in the masterbedroom and there is another connection between the DVD and the PDA. It seems that the intention of the user is to use the connection in which the input device is located in the same place as the user. This makes the other connection unnecessary.

The second rule $incompatible - activities - 2$ suspends user activities that require the user presence whenever the user leaves the same location as the activity has.

This example focuses on how the system adaptes the services offered to the user accordingly with its current activities, that is, the design for domestic use.

This chapter is intended to show how the proposal works by means of several examples. Next chapter summarises this experience and all the information

presented in this thesis in order to present the conclusions extracted to the reader.

Chapter 9

Conclusions, Contributions, and Future Work

In the previous chapters, it was proposed a framework for implementing ad-hoc service composition in a home network considering the requirements established in chapter 1. The proposal involves describing rich semantic services, which are correctly implemented and capable of being deployed in heterogeneous devices.

9.1 Conclusions

This thesis has presented a novel framework for allowing seamless device cooperation in ad-hoc environment. In fact, this thesis has presented the integration of an ontology with a formal method in order to accomplish an smart home implementation. This smart home implementation meets a set of requirements needed for that kind of systems to succed. Those requirement are found in the literature and comprise the following topics: horizontal integration, spontaneous collaboration, design for domestic use, reliability and low cost.

Horizontal integration is achieved by modelling home device functionality (components) as services. Such services are described in OWL-S and a context based ontology which gives a rich semantic and pragmatic knowledge about device functionality. Due to the process view of OWL-S, such services can be specified in the formal algebra CSP for service verification and implementation. With this architecture, services act as the perfect building blocks for developing ubiquitous applications. In order to facilitate communication, we have implemented a comunication protocol based on S-Expressions which facilitate the integration of simple and complex devices.

A great advantage of OWL-S is its process of standardisation at W3C. With the advent of Internet, this feature will allow a seamless integration of home device

services with services outside the home environment, as they are described in the same language. The only difference between the services described in this thesis and the traditional OWL-S services implemented in WSDL is the grounding interface. To overcome this challenge, the only needed update is to provide a WSDL interface for home devices, as the profile and service model remain the same. Another solution will propose the codification of SOAP messages with S-Expressions.

The needed syntactical, semantic and pragmatic agreements for spontaneous or ad-hoc collaboration are provided by the use of OWL, OWL-S and CONON. This knowledge is managed by a rule based system which outputs new service combinations to help users in their daily activities. As it is known, OWL-S is an effort for achieving automatic web service composition. The use of OWL-S permits the reuse of this vast knowledge to be applied in a smart home. Artificial Intelligence techniques such as HTN planning can be applied in this project as they have performed successfully with OWL-S [101].

The design for domestic use is the more open topic in this work. In the proposed simulator, neither an attentive nor learning strategy has been proposed. The aim of the simulator is two show how the knowledge supplied by services might be used to achieve service composition, and not as a final implementation of the smart home. As it has been introduced in the literature review chapter, several strategies can be adopted to achieve this goal. In light of this, an important effort is being carried out at Loughborough University with the aim of modelling user intention through the use of the semiotics concept. It seems that this technique can be applied as an upper layer on this thesis to improve the domestic use of this project [44, 57].

One of the main reason for the introduction of CSP in this project is the need of the achievement of a high level of reliability in the final implementation of the smart home. The use of CSP allows the detection of typical problems in distributed applications such as deadlocks and livelocks plus the prediction of the behaviour of the system due to the refinement technique. This test can be done automatically thanks to the model checker tools available, such as FDR2 and Probe [8]. In light of this and in connection with the provision of supporting tools, a tool for specifying services in the ontology introduced in chapter 5 has been proposed. The tool is an extension of the OWL-S pluging [49] for Protege [21] which provides a connection with FDR2 to check if the services specified are deterministic and free of deadlock and livelock [89].

The cost reduction is intended to stem from the achievement of the previous requirements. Another reason of the introduction of CSP is the possibility of obtaining lightweight implementations, close as hardware as much as possible. This

feature, plus the inclusion of S-Expressions, may reduce the final cost of the devices capable of supporting the technology proposed in this thesis. In addition, the existence of automatic tools for code generation from CSP specifications [87, 85] will permit a rapid development of applications. Although multiple solutions can be used for providing communication between different implementations of CSP such as RMI or CORBA [92], the utilisation of a protocol in S-Expressions allows better interoperation between different service implementations, independently of their nature, as XML does.

9.2 Contributions

The principal contribution of this thesis is the achievement of a framework capable of spontaneous and correct integration of heterogeneous components in an ad-hoc environment.

Although ontologies and formal methods have been combined before for web service composition, this is the first time that OWL-S, context ontologies and CSP are combined. In addition, the application of this combination is beyond the web services world, being applicable to any ad-hoc environment. The flexibility given by the OWL-S ontology, plus the wide range of possible CSP implementations bring several advantages when developing ubiquitous computing applications.

Firstly, the enhanced semantics achieved with the combination of OWL-S and CSP establishes a direct path between the ontological representation to the final implementation of the service, opening the door for the creation of tools for automatic code generation. For example, this means that will be possible to establish a clear relation between the OWL-S representation of a light switch and its hardware implementation in an FPGA programmed in Handle-C.

Secondly, the correspondence between a OWL-S representation and its CSP specification permits the verification of service behaviour before its implementation. In addition, intelligent systems may validate service compositions specified in OWL-S with the refinement technique, understanding how the service is going to behave before its execution. This aspect becomes particularly interesting when studying unsuspected behaviours between different subsystems, difficulty known as feature interaction problem [68, 70]. Both possibilities enhance to a great extent the reliability of the final system and hence, reduce the costs derived from system failures.

Thirdly, the inclusion of S-Expresion as the framework for message communication allows a flexible and lightweight protocol for service communication. This scheme notably helps in the achievement of the requirements of horizontal integration, spontaneous collaboration and low cost implementation as it combines

the independence of XML with the advantages of a lightweight message parsing procedure.

In the fourth place, a JCSP based service implementation has been presented in order to show how it is possible to obtain service implementations from service specifications in CSP.

Finally, a framework was proposed to understand what are the key requirements when an smart home needs to be implemented. Accordingly, this thesis proposes a methodology for the combination of formal methods and ontologies. The aim of this methodology is to procure the seamless combination of an ontology and a formal method which best satisfies the requirements for an smart home.

9.3 Future Work

The future lines of research from this project are addressed to overcome the principal problems found.

Firstly, the verification of services compositions is a expensive task in time and resources, making sometimes impracticable the refinement analysis before service execution. Solutions to overcome this problem points to the optimisation of the CSP service definitions in order to increase the performance of the model checker.

Secondly, the refinement technique takes only into account context information related to state variables. It would be worth the incorporation of techniques to translate more context information to CSP, and hence, improve the verification of the final system. However, this line of research has to consider the time constrain needed for the model checker and explained in the previous paragraph.

Finally, it will be desirable to develop a set of mechanisms for the achievement of a service implementation from the OWL-S representation. Special interest should be taken to hardware implementations, as they will notably reduce the costs derived from a final smart home product.

Glossary

CSP (Communicating Sequential Processes) A mathematical theory for describing Parallel applications developed by C.A.R.Hoare. It is possible to prove correctness of programs described using CSP.

Formal Method Collection of mathematical structures, together with a precise syntax for defining instances of those structures and organising them into domain-specific abstractions. It should be associated with a method for eliciting these abstractions and transforming them.

JCSP (Java CSP) JCSP is a binding of the occam/CSP parallel computing model for Java. Basic packages provide processes, channels, parallel and choice (ALT) constructors. A channel interface to the Java AWT components is also included.

Jess (Java Expert System Shell) The Java Expert System Shell. You write declarative rules, if/then, and the system generates code that does artificial reasoning to find solutions to your constraints. The Jess language is a sort of Javafied Lisp.

Jini Sun's protocol for devices to identify each other using TCP/IP protocol. It will be used in small devices like telephones. It allows you to plug a new device into the system while everything is running. The device automatically finds out about everything else on the net and vice versa. The device can create Java objects that can be passed around the net. This allows other devices on the net to start using the new device and vice versa without needing to install any software.

occam The occam language was designed to implement the CSP model for the Transputer processor.

Ontology A branch of study concerned with the nature and relations of being, or things which exist.

OWL (Ontology Web Language) A set of markup languages which are designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL ontologies describe the hierarchical organization of ideas in a domain, in a way that can be parsed and understood by software. OWL has more facilities for expressing meaning and semantics than XML, RDF, and RDF-S, and thus OWL goes beyond these languages in its ability to represent machine interpretable content on the Web. OWL is part of the W3C recommendations related to the Semantic Web.

OWL-S (Ontology Web Language for Services) a core set of markup language constructs for describing the properties and capabilities of Web services in unambiguous, computer-interpretable form. OWL-S is based on ontologies of objects and concepts defined using OWL.

Petri Net A Petri net (also known as a place/transition net or P/T net) is one of several mathematical representations of discrete distributed systems. As a modeling language, it graphically depicts the structure of a distributed system as a directed bipartite graph with annotations.

RDF (Resource Description Framework) RDF is designed to provide an infrastructure supporting Meta Data across many WWW-based activities. RDF is the result of a number of Meta Data communities bringing together their needs to provide a robust and flexible architecture for supporting Meta Data on the Internet and the WWW. Example applications include site maps, content ratings, stream channel definitions, search engine data collection, digital library collections, and distributed authoring. RDF allows different application communities to define the Meta Data property set that best serves the needs of each community. RDF provides a uniform and interoperable means to exchange the Meta Data between programs and across the WWW. Furthermore, RDF provides a means for publishing both a human-readable and a machine-understandable definition of the property set itself. RDF uses XML as the transfer syntax in order to leverage other tools and code bases being built around XML.

Semantic Web The Web of data with meaning in the sense that a computer program can learn enough about what the data means to process it.

S-Expression A s-expression is the basic syntactic unit of Lisp in its textual form: either a list, or Lisp atom. Many Emacs commands operate on sexps. The term 'sexp' is generalized to languages other than Lisp, to mean a syntactically recognizable expression.

SOAP (Simple Object Access Protocol) A lightweight, XML based protocol for passing objects between components in a decentralized distributed environment. The SOAP protocol includes an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing datatypes, and a convention for representing remote procedure calls and responses. SOAP may use HTTP or other protocols as the transport mechanism.

UDDI (Universal Description, Discovery and Integration) business registry and repository for storing information about businesses and the electronic services they offer. UDDI creates a standard interoperable platform that enables companies and applications to quickly, easily and dynamically find and use Web Services over the internet.

UPnP (Universal Plug and Play) Universal Plug and Play is a networking architecture developed by a consortium of companies to ensure easy connectivity between products from different vendors. UPnP devices should be able to connect to a network automatically, handling identification and other processes on the fly. The standards developed by the UPnP Forum are media-, platform-, and device-independent.

URI (Universal Resource Identifier) The string (often starting with http:) that is used to identify anything on the Web.

W3C A standards organization which produced the standards for XML, XSL, and HTTP among many others.

Web Services A service is a component performing a task, perhaps over a network. A web service can be identified by a URI. Its public interfaces and bindings are described using XML. It's definition can be discovered by clients who can interact with the web service using it's definition.

WSDL (Web Service Description Language) WSDL is an XML vocabulary for describing network services as a set of endpoints operating on messages. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format. Note that the acronym WSDL is commonly pronounced "WizDel".

WSML (Web Service Modeling Language) a language framework for semantic Web services, based on the conceptual model of WSMO. WSML provides means to describe semantic Web services and its related aspects, i.e. ontologies, web Services, goals, and mediators. Those descriptions aim at automating Web service related tasks such as discovery, mediation and invocation

WSMO (Web Service Modeling Ontology) WSMO is a meta-model for Semantic Web services related aspects.

XML (eXtended Markup Language) A text based markup language that is fast becoming the standard for data interchange. The language is extensible because you are free to use any tags you wish to describe the data. XML is a descendant of SGML.

Bibliography

[1] Aura Project at the Carnegie Mellon University. http://www-2.cs.cmu.edu/~aura/.

[2] AUTOHAN Web Site. Available at: http://www.cl.cam.ac.uk/Research/SRG/HAN/AutoHAN/.

[3] Business Process Execution Language for Web Services version 1.1, (BPEL4WS). Available at: http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

[4] CABA Standards and Protocols. Available at: http://www.caba.org/standard/index.html.

[5] Chart of Patterns in Ubiquitous and Context Computing. Available at: http://kettle.cs.berkeley.edu/ubicomp.

[6] Communicating Threads for Java (CTJ). http://www.ce.utwente.nl/javapp/.

[7] Endeavour Project at the University of Berkeley. Available at: http://endeavour.cs.berkeley.edu.

[8] Formal Systems Europe Ltd. Available at: http://www.fsel.com.

[9] Handel-C, Software Compiled System Design. Available at: http://www.celoxica.com/methodology/handelc.asp.

[10] The Home_n Project. Available at http://architecture.mit.edu/house_n/intro.html.

[11] ISO/IEC JTC1 SC25 WG1 Home Electronic System Standards. Available at: http://hes-standards.org/.

[12] Java Communicating Sequential Processes (JCSP). Available at: http://wotug.ukc.ac.uk/parallel/languages/java/jcsp/.

[13] Java Media Framework API (JMF). Available at: http://java.sun.com/products/java-media/jmf/.

[14] Jess, The Rule Egine for the Java Platform. Available at: http://www.jessrules.com.

[15] MediaServer v 1.0 and MediaRenderer v 1.0. Available at: http://www.upnp.org/standardizeddcps/mediaserver.asp.

[16] OSGi Web Site. Available at: http://www.osgi.org/.

[17] OWL-based Web Service Ontology, OWL-S. Available at: http://www.daml.org/services/owl-s/.

[18] Oxygen Project at the Massachusetts Institute of Technology. Available at: http://oxygen.lcs.mit.edu.

[19] Pebbles in Oxygen and Autohan. Available at: http://www.cl.cam.ac.uk/Research/SRG/HAN/pebbles/.

[20] Portolano Project at the University of Washington. Available at: http://portolano.cs.washington.edu.

[21] Protege. Available at: http://protege.standford.edu/.

[22] Resource Description Framework (RDF). Available at: http://www.w3c.org/RDF/.

[23] Semantic Web at W3C. Available at: http://www.w3.org/2001/sw/.

[24] Task Computing. http://taskcomputing.org/.

[25] The Adaptative House at Colorado University. Available at: http://www.cs.colorado.edu/~mozer/house/.

[26] The Occam Archive. http://vl.fmnet.info/occam/.

[27] The Web Ontology language (OWL). Available at: http://www.w3.org/2004/OWL/.

[28] UPnP Forum. Available at: http://www.upnp.org.

[29] UPnP Specification of HVAC Systems (HVAC 1.1). Available at: `http://www.upnp.org/standardizeddcps/hvac.asp`.

[30] Web Service Modeling Ontology, WSMO. Available at: `http://www.wsmo.org`.

[31] Web Services for Business Process Design (XLANG). Available at: `http://www.gotdotnet.com/team/xml-wsspecs/xlang-c/default.htm`.

[32] X-10 Forum. Available at: `http://www.x10.com/support/tech_index.html`.

[33] *Inside the Smart Home*, chapter 2. Harper, R, 2003.

[34] J-R Abrial, S. A Schuman, and B. Meyer. *A Specification Language, in On the Construction of Programs*. Cambridge University Press, 1980.

[35] J. Arias Fisteus. *Definition of a Formal Model for the Verification of Business Processes*. PhD thesis, Universidad Carlos III de Madrid, Madrid, Spain, Sep 2005. In Spanish.

[36] G. Banavar and A. Bernstein. Software Infrastructure and Design Challenges for Ubiquitous Computing Applications. *Communications of the ACM*, 45(12):92–96, Dec 2002.

[37] B. Boehm and V. R Basili. Software Defect Reduction Top 10 List. *IEEE Computer*, 34(1):135–137, Jan 2001.

[38] J. Bohn, V. Coroama, M. Langheinrich, F. Mattern, and M. Rohs. Social, Economic, and Ethical Implications of Ambient Intelligence and Ubiquitous Computing. In W. Weber, J. Rabaey, and E. Aarts, editors, *Ambient Intelligence*, pages 5–29. Springer-Verlag, 2005.

[39] N. Brown and P.H. Welch. An Introduction to the Kent C++CSP Library. In *CPA 03: Communicating Process Architectures 2003*, Concurrent Systems Engineering Series, pages 139–156, Enschede, Netherlands, Sep 2003. IOS Press.

[40] N. C. Brown. C++CSP Networked. In *CPA 04: Communicating Process Architectures 2004*, pages 185–200, Oxford, UK, Sep 2004. IOS Press.

[41] L Cardelli and A.D Gordon. Mobile Ambients. In *First International Conference on Foundations of Software Science and Computation Structure*, March 1998.

[42] H. C. B. Cheng. Applying Formal Methods in Automated Software Development. *Journal of Computer and Software Engineering*, 2(2):137–164, 1994.

[43] E. Chirstopoulou and A. Kameas. GAS Ontology: An Ontology for Collaboration among Ubiquitous Computing Devices. *International Journal of Human - Computer Studies*, 62(5):664 – 685, May 2005.

[44] J. H. Connolly, I. W. Phillips, and L. Hawizy. A Semiotic Framework for Research into Self-Configuring Computer Networks. In *IWOS 05: Proceedings of the 8th International Workshop on Organisational Semiotics*, Tolouse, France, Jun 2005.

[45] D. Cook, M. Youngblood, E. Heierman, K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja. Mavhome: An Agent-Based Smart Home. In *PerCom 03: Procceedings of 1st IEEE International Conference ov Pervasive Computing and Communications*, pages 521 – 524, Fort Worth, Texas, USA, Mar 2003. IEEE Computer Society.

[46] J. Davies and C. Crichton. Concurrency and Refinement in the Unified Modeling Language. *Formal Aspects of Computing*, 15(2-3):118–145, November 2003.

[47] N. Davies and H. Gellersen. Beyond Prototypes: Challenges in Deploying Ubiquitous Systems. *IEEE Pervasive Computing*, 1(1):26–35, Jan 2002.

[48] W. K. Edwards and R. E. Grinter. At Home with Ubiquitous Computing: Seven Challenges. In *UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing*, Atlanta, Georgia, USA, Sep-Oct 2001. Springer-Verlag.

[49] D. Elenius, G. Denker, D. Martin, F. Gilham, J. Khouri, S. Sadaati, and R. Senanayake. The OWL-S Editor - A Development Tool for Semantic Web Services. In *ESWC 05: Proceedings of the 2nd European Semantic Web Conference*, pages 78–92, Heraklion, Greece, May 2005. Springer Berlin.

[50] M. Esler, J. Hightower, T. Anderson, and G. Borriello. Next Century Challenges: Data-Centric Networking for Invisible Computing. The Portolano Project at the University of Washington. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 256–262, Seattle, Washington, United States, Aug 1999. ACM Press.

[51] W. Fokkink, I. van Langevelde, and Y. Usenko. How can I be Sure that my DVD Player Understands my TV? *ERCIM News 47*, pages 34–35, Oct 2001. Available at: `http://www.ercim.org/publication/Ercim_News/enw47/fokkink.html`.

[52] E. Friedman-Hill. *Jess in Action : Java Rule-Based Systems (In Action Series)*. Manning Publications, 2002.

[53] A. Fuggetta. Software Technologies, Embedded Systems and Dis- tributed Systems. A European Strategy Towards an Ambient Intelligent Environment. Technical report, Information Society Technologies (IST), Jun 2002.

[54] W. Green, D. Gyi, R. Kalawsky, and D. Atkins. Capturing User Requirements for an Integrated Home Environment. In *NordiCHI '04: Proceedings of the Third Nordic Conference on Human-Computer Interaction*, pages 255–258, Tampere, Finland, Oct 2004. ACM Press.

[55] S. D. Gribble, M. Welsh, R. von Behren, E. A. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross, B. Zhao, and R. C. Holte. The Ninja Architecture for Robust Internet-Scale Systems and Services. *IEEE Computer Networks. Special Issue on Pervasive Computing*, 35(4):473–497, Mar 2001.

[56] R. Grimm, J. Davis, E. Lemar, M. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, and D. Wetherall. System Support for Pervasive Computing Applications. *ACM Transactions on Computer Systems*, 24(2):421–486, Nov 2004.

[57] L. Hawizy, J. H. Connolly, and I. W. Phillips. Intention Modeling: A Semiotic View. In *Proceedings of the IADIS International Conference. Applied Computing*, San Sebastián, Spain, Feb 2006.

[58] C. Heitmeyer. A panacea or Academic Poppycock: Formal Methods Revisited. In *HASE '05: Proceedings of the Ninth IEEE International Symposium on High-Assurance Systems Engineering*, pages 3–7, Heidelberg,Germany, Oct 2005. IEEE Computer Society.

[59] S. Helal. Standars for Service Discovery and Delivery. *IEEE Pervasive Computing*, 1(3):95–100, 2002.

[60] S. Helal. Programming Pervasive Spaces. *IEEE Pervasive Computing*, 4(1):84–87, Mar 2005.

[61] G.H Hilderink, A.W.P Bakkers, and J.F Broenink. A Distributed Real-Time Java System Based on CSP. In *Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Mar 2000.

[62] Hoare, C.A.R. *Communicating Sequential Processes*. Prentice Hall International, 2003.

[63] S. S. Intille. Designing a Home of the Future. *IEEE Pervasive Computing*, 1(2):76–82, Apr 2002.

[64] IST Advisory Group (ISTAG). Ambient Intelligence: From Vision to Reality. In G. Riva, F. Vatalaro, F. Davide, and M Alcañiz, editors, *Ambient Intelligence*. IOS Press, 2005.

[65] L. M. Jessup and D. Robey. The Relevance of Social Issues in Ubiquitous Computing Environments. *Communications of the ACM*, 45(12):88–91, Dec 2002.

[66] L. Jiang, L. Da-You, and B. Yang. Smart Home Research. In *Third International Conference on Machine Learning and Cybernetics*, Shanghai, China, Aug 2004.

[67] S. D. Johnson. Formal Methods in Embedded Design. *IEEE Compututer Society*, 36(11):104–106, Nov 2003.

[68] D. O. Kech and P. J. Kuehn. The Feature and Service Interaction Problem in Telecomunications Systems: A Survey. *IEEE Transactions on Software Engineering*, 24(10):779–796, Oct 1998.

[69] S. Koide, J. Aasman, and S Haflich. OWL vs. Object Oriented Programming. In *SWESE 05: Proceedings of the International Workshop on Semantic Web Enabled Software Engineering*, Galway, Ireland, Nov 2005.

[70] M. Kolberg, E. H Magill, and M. Wilson. Compatibility Issues between Services Supporting Networked Appliances. *IEEE Communications Magazine*, 41(11):136–147, Nov 2003.

[71] J. A. Landay and G. Borriello. Design Patterns for Ubiquitous Computing. *IEEE Compututer Society*, 36(8):93–5, Aug 2003.

[72] Y. Lee, S. A. Chun, and J. Geller. Web-Based Semantic Pervasive Computing Services. *IEEE Intelligent Informatics Bulletin*, 4(2):4–15, Dec 2004.

[73] K. Lyytinen and Y. Yoo. Issues and Challenges in Ubiquitous Computing. *Communications of the ACM*, 45(12):63–66, Dec 2002.

[74] R. Masuoka, B. Parsia, Y. Labrouu, and E. Sirin. Ontology-Enabled Pervasive Computing Applications. *IEEE Intelligent Systems*, 18(5):68–72, feb 2003.

[75] S. McIlraith and D. Mandell. Comparison of DAML-S and BPEL4WS. Technical report, Knowledge Systems Lab, Stanford University, 2002.

[76] D. Moldt and J. Ortmann. A Conceptual and Practical Framework for Web-Based Processes in Multi-Agent Systems. In *AAMAS'04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1464–1465, New York, New York, Aug 2004. IEEE Computer Society.

[77] M. C. Mozer. The Neural Network House: An Environment that Adapts to its Inhabitants. In *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*, pages 110–114, Menlo, Park, CA, 1998. AAAI Press.

[78] T. Murakami. Establishing the Ubiquitous Network Environment in Japan. Technical Report 66, Nomura Research Institute, Jul 2003.

[79] E.D. Mynatt, J. Rowan, S. Craighill, and A. Jacobs. Digital Family Portraits: Providing Peace of Mind for Extended Family Members. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, Seattle, Washington, USA, Jun 2001. ACM Press.

[80] S. Narayanan and S. A. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 77–88, Honolulu, Hawaii, USA, May 2002. ACM Press.

[81] M. Newman. Software Errors Cost U.S. Economy $59.5 Billion Annually NIST Assesses Technical Needs of Industry to Improve Software-Testing. Technical report, National Institute of Standards and Technology (NIST), 2002.

[82] J. O'Brien, T. Rodden, M. Rouncefield, and J Hughes. At Home with the Technology: An Ethnographic Study of a Set-Top-Box Trial. *ACM Transactions in Computer-Human Interactaction*, 6(3):282–308, Sep 1999.

[83] C. Perkins. *RTP: Audio and Video for the Internet*. Addison-Wesley, 2003.

[84] J. L. Peterson. Petri Nets. *ACM Computing Surveys*, 9(3):223–252, 1977.

[85] J. D. Phillips and G. S. Stiles. An Automatic Translation of CSP to Handel-C. In *CPA 04: Communicating Process Architectures 2004*, pages 19–38, Oxford, UK, Sep 2004. IOS Press.

[86] K. Pugh. Configuration, Discovery and Mapping of a Home Network. *IEE Proceedings - Software*, 150(2):155–160, Apr 2003.

[87] V. Raju, L. Rong, and G. S. Stiles. Automatic Conversion of CSP to CTJ, JCSP, and CCSP.

[88] J.I. Rendo and I.W. Phillips. Pervasive Computing with OWL-S and a Formal Method. Conference Supplement of the UBICOMP 05 hold in Tokyo, Japan, Sep 2005.

[89] J.I. Rendo Fernández and I.W. Phillips. Ad-Hod Networking with OWL-S and CSP. In *IEEE IS 06: Proceedings of 3rd IEEE Conference on Intelligent Systems*, London, UK, Sep 2006. IEEE Computer Society.

[90] J.I Rendo Fernández, I.W. Phillips, and A.W. Lawrence. A CSP Based Ontology for a Smart Home. In *Proceedings of the IADIS International Conference WWW/Internet 2004*, Madrid, Spain, October 2004.

[91] G. G. Richard. Service Advertisement and Discovery: Enabling Universal Device Cooperation. *IEEE Internet Computing*, 4(5):18–26, Sep 2000.

[92] A. Ripke, A.R Allen, and Y. Feng. Distributed Computing using Channel Communications in Java. In *CPA 03: Communicating Process Architectures 2003*, Concurrent Systems Engineering Series, pages 1–16, Enschede, Netherlands, Sep. IOS Press.

[93] R. Rivest. S-Expressions. Available at: `http://theory.lcs.mit.edu/~rivest/sexp.txt`, Nov 1997.

[94] A. W Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1997.

[95] G. Salaün, L. Bordeaux, and M. Schaerf. Describing and Reasoning on Web Services using Process Algebra. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, San Diego, California, USA, Jun 2004. IEEE Computer Society.

[96] W Schonfeld. Interacting Abstract State Machines. In *the 28th Annual Conference of the German Society of Computer Science, Technical Report, Magdeburg University*, 1998.

[97] M. Schweigler, F. Barnes, and P.H. Welch. Flexible, Transparent and Dynamic occam Networking with KRoC.net. In *CPA 03: Communicating Process Architectures 2003*, pages 199–224, Enschede, Netherlands, Sep 2003. IOS Press.

[98] F. Scuglik. Formal Specification of Shared Variables Using CSP. In *ECBS'04: Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, Brno, Czech Republic, May 2004. IEEE Computer Society.

[99] B. Sharpe. Information Appliances. An Introduction. Available at: `http://www.appliancestudio.com/publications/whitepapers/ApplianceIntro.pdf`, Jun 2001. Apliance Studio Ltd. White Paper.

[100] I. Siio, J. Rowan, and E. Mynatt. Peek-a-drawer: Communication by Furniture. In *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems*, pages 582–583, Minneapolis, Minnesota, USA, Apr 2002. ACM Press.

[101] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN Planning for Web Service Composition Using SHOP2. *Journal of Web Semantics*, 1(4):377–396, 2004.

[102] J. P. Sousa and D. Garlan. Aura: An Architectural Framework for User Mobility in Ubiquitous Computig Environments. In *Software Architecture: System Design, Development and Maintance. Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*, pages 29–43, Montreal, Canada, Aug 2002. Kluwer, B.V.

[103] X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology Based Context Modeling and Reasoning using OWL. In *PERCOMW '04: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, Orlando, FL, USA, Orlando,FL, USA 2004. IEEE Computer Society.

[104] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):66–75, Jan 1991.

[105] P. H. Welch. Java Threads in the Light of occam/CSP. pages 259–284, Apr 1998.

[106] P.H. Welch, J.R. Aldous, and J. Foster. CSP networking for java (JCSP.net). In *ICCS 02: International Conference on Computational Science*, pages 695–708, Apr 2002.

[107] P.H Welch and D.C. Wood. The Kent Retargetable occam Compiler. In *Parallel Processing Developments, Proceedings of WoTUG 19*, pages 143–166, Nottingham, UK, Mar 1996. IOS Press.

[108] F. Zhu, M.W. Mutka, and L.M. Ni. Service Discovery in Pervasive Computing Environments. *IEEE Pervasive Computing*, 4(4):81–90, Oct 2005.