Vrije Universiteit Brussel

FACULTY OF SCIENCE AND BIO-ENGINEERING SCIENCES
DEPARTMENT OF COMPUTER SCIENCE
THE WEB & INFORMATION SYSTEMS ENGINEERING LABORATORY

An Ambient Information System for Smart Lab Environments

Graduation thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Applied Sciences and Engineering: Computer Science

Marius-Gheorghe Hîrjanu

Promoter: Prof. Dr. Beat Signer
Advisor: Reinout Roels

Academic year 2015-2016

Vrije Universiteit Brussel

FACULTEIT WETENSCHAPPEN EN BIO-INGENIEURSWETENSCHAPPEN
VAKGROEP COMPUTERWETENSCHAPPEN
THE WEB & INFORMATION SYSTEMS ENGINEERING LABORATORY

An Ambient Information System for Smart Lab Environments

Masterproef ingediend in gedeeltelijke vervulling van de eisen voor het behalen van de graad Master of Science in de Ingenieurswetenschappen: Computerwetenschappen

Marius-Gheorghe Hîrjanu

Promotor: Prof. Dr. Beat Signer
Begeleider: Reinout Roels

Academiejaar 2015-2016

Abstract

Nowadays we are facing a rapidly growing trend of intelligent ambient systems that can assist in our daily tasks and bring a more comfortable way of access to digital information. By embedding it into academic environments such as research laboratories and meeting rooms they could improve the way people study, teach and research different topics. An ambient system consists of computer software and a network of electronic devices, sensors and web services (facts), with the goal to create environments sensitive and responsive to the presence of people. Because of the variety of software and hardware that exist today, it is still hard to find an optimal system that suits all of our needs. Some of the actual software frameworks are restricted to a specific operating platform, type of communication interface or only work with a limited set of sensors and devices. Others are not extensible or do not support some features that are essential in an ambient information system.

The objective of this thesis is to investigate the current application of ambient intelligence paradigm in academic environments and determine an optimal context-aware framework model that goes beyond these limitations. As a result of this study, a framework called "BeAware" has been built. BeAware is an extensible framework that works on top of the classical network configuration via REST Web Services and WebSockets, and consists of a user interface and a core. The user interface allows users to add facts and create rules on that facts through a friendly graphical interface. The core provides real-time processing, reasoning and ensure a communication with added facts. The framework is able to identify people present in academic environments via their smartphones. It allows the creation of rules that are active only when that user is physically present, rules that may be migrated to other ambient environments, but also (among others) allows users to share rules between users, to auto-discover devices through the network and to control multimedia presentations. Last but not least, the framework allows to quickly deploy ambient systems as it was demonstrated in our university research laboratory by three different use cases.

Declaration of Originality

I hereby declare that this thesis was entirely my own work and that any additional sources of information have been duly cited.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this thesis has not been submitted for a higher degree to any other University or Institution.

Acknowledgements

Foremost, I would like to express my appreciation and gratitude to my promoter Prof. Dr. Beat Signer and my advisor Reinout Roels for giving me the opportunity to develop this thesis under their supervision at *The Web & Information Systems Engineering (WISE) Laboratory*. Their continuous help and valuable advice made the realisation of this project possible.

Secondly, a very special thanks goes out my boss Frank De Muynck, who offered me financial and moral support and gave me the chance to apply the practical study skills in his company. Also to my workmates, especially to Akos Nagy who always has been there to give help and guidance.

Next, I take this opportunity to express profound gratitude to my beloved family. I warmly appreciate and thank my parents for their love and their material and spiritual support in all aspects of my life. Special thanks go to my sister Gabriela, who always gives invaluable support and encourage me to become the best version of myself, to my brother-in-law and to my girlfriend Paulina.

Furthermore, I would like to thank my friends for always being there for me and cheering me up when I needed it.

Finally, I would like thank you for taking the time to read my work.

Contents

List of Abbreviations

AD Ambient Display

AIS Ambient Information System

AJAX Asynchronous JavaScript and XML

AmI Ambient Intelligence

API Application Programming Interface

C2S Client-Server (network communication model)

CAF Context Awareness Framework

CM Context Manager

CMS Content Management System

CRUD Create, Read, Update, Delete

CSS Cascading Style Sheets

CTT Concurrent Task Trees

DUMMBO Dynamic Ubiquitous Mobile Meeting Board

FSR Force-Sensing Resistor

GPS Global Positioning System

GUI Graphical User Interface

GYRO Gyroscope

HCI Human Computer Interaction

HDMI High-Definition Multimedia Interface

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IoT Internet of Things

IP Internet Protocol

IR Infrared

JSON JavaScript Object Notation

LAN Local Area Network

LCD Liquid-Crystal Display

LED Light-Emitting Diode

MAC Media Access Control

MIT Massachusetts Institute of Technology

MVC Model-View-Controller

ORM Object-Relational Mapping

PC Personal Computer

PDA Personal Digital Assistant

PHP PHP: Hypertext Preprocessor

REST Representational State Transfer

RFID Radio-frequency identification

RGB Red, Green and Blue (colour model)

RSS Rich Site Summary

S2S Server-Server (network communication model)

SOA Service Oriented Architecture

UML Unified Modelling Language

URL Uniform Resource Locator

USB Universal Serial Bus

VPN Virtual Private Network

VUB Vrije Universiteit Brussel

WAN Wide Area Network

WISE The Web & Information Systems Engineering

WOA Web-Oriented Architecture

WoT Web of Things

WWW World Wide Web

XML Extensible Markup Language

YAML YAML Ain't Markup Language

1 Introduction

"The most profound technologies are those that disappear. They weave themselves into the fabrics of everyday life until they are indistinguishable from it." [1]

1.1 Research Context

Humans have multiple ways of perceiving and processing digital information. Via the most common sensory perceptions (sight, hearing, touch) humans can easily receive information from systems such as Ambient Information Systems (AIS). Moreover people have a sophisticated capability to process multiple input streams of information concurrently [2,3].

An AIS is a system consisting of computers and electronic devices interconnected in a network with the goal of creating an intelligent ambient environment that is sensitive and responsive to the presence of people. In this context, the intelligent environment is given by the fact that the devices exchange information in the network and take decisions without any direct intervention from the users. The concept is defined as Ambient Intelligence (AmI) — a recent paradigm of Artificial Intelligence in which the computing

power is used to support the daily user activities and make the life of the users more comfortable. With the evolution of technology in the last decade, computers and electronic devices became considerably reduced in size which have made possible a significant development for AmI [4]. Today's devices even permit to be camouflaged in any kind of environment without us noticing or consciously thinking about them. The concept of autonomous decision making or AmI has been integrated in various devices surrounding us such as: mobile phones, programmable washing machines, heating systems, parking assist systems, GPS navigation systems, traffic light systems or energy savings systems. With all of these there are still many challenges and research work that has to be done [5,6,7]. The challenge that is treated in this dissertation is related to an AIS for research laboratory environments.

The project *Ambient Information System for Smart Lab Environments* is mainly focusing on Human Computer Interaction (HCI), in exact terms — the interaction between the users from the academic environment (teachers, students, researchers), computer and electronic devices (e.g. temperature sensors, GPS, Kinect[1], TV displays and projectors) and information using Web Services (e.g. Google Calendar[2], Dropbox[3], news). The system is aware about the users from multi-room laboratories and takes smart decisions (for instance: turn on/off the display, show information on the display, adjust the temperature) in order to support them in their daily life tasks. To establish the communication between devices but also to have a personalised system decision making meeting the users expectations, the needed prototype framework has been developed.

The framework architecture uses the existing physical network configuration and is based on REST Web Services [8] and Publish-Subscribe architecture [9,10]. It consists of an engine that processes the information in real-time and a user interface that allows the users to input custom user configurations and interact with the engine.

The interface provides the following functionalities:

a) define and manage rules in a graphical interface
b) share the rules with other users via four particular methods (email, user account, web address and social media — Facebook[4]/Twitter[5])

[1]Kinect: https://www.microsoft.com/en-us/kinectforwindows
[2]Google Calendar: http://www.google.com/calendar
[3]Dropbox: https://www.dropbox.com
[4]Facebook: https://www.facebook.com
[5]Twitter: https://www.twitter.com

c) define and manage facts (smart devices or web services)
d) handle the devices auto-discovered through the network
e) monitor the users presence and list the acquisition values and rules effect
f) manage the users and users permission
g) manage the system configuration

The engine consists of the following modules:

a) rule engine (processes the rules)
b) data acquisition (collects the values from the sensors or web services)
c) trigger devices (sends the value to the device based on the rule result)
d) auto-discover devices (permits to scan and find the devices through the network)
e) user presence (discovers and identifies if the user id present in one of the labs)
f) control display (controls what to be shown on the display at any moment)

The system allows rules to be set up so that they are only triggered when a specific user is detected in the lab. Furthermore, the same rule can be applied in multiple laboratories and triggered by the local system in the lab where the user is sensed. In this way, the same context of lab environment is automatically adapted to the user rules preferences. A basic example can be considered as following: *two students use the university laboratory to complete their projects. They have particular temperature preferences. By having defined their own rules with a distinct temperature value, the system will automatically adjust the ambient temperature for the user that is currently working in the lab. In case that one of the user moves to another lab, the system will adjust the temperature adequately.*

A demo of the framework prototype has been deployed in the The Web & Information Systems Engineering (WISE) Laboratory. However, this system can not only be integrated in the WISE laboratory from Vrije Universiteit Brussel (VUB) but also in any other academic laboratories around the world.

1.2 Problem Statement and Research Question

With the multitude of hardware and software choices that exist today, it is still not easy to find an optimal framework for building an AIS that comes with all the desired features "out of the box". Some frameworks that have been built, work only on a specific field, operating system, type of interface or communication protocol. Others require particular programming languages, advanced programming skills for simple configurations or need a lot of third-party software to make it working.

On the other side, the existing frameworks for AIS purpose, are limited in the number of features and do not offer support for some characteristics that are really usable (e.g. existing frameworks do not offer the possibility to use the same rule in a large number of ambient spaces and do not allow to share rules with other users). Furthermore, some frameworks are restricted to a specific set of devices they allow to connect or they do not permit to extend for future development.

For this purpose we conducted a literature research in which we examine the related ambient systems and frameworks. We firstly reviewed some Ambient Environments (Section 2.2.1) and Ambient Displays (Section 2.2.2) to get background on the existing applications in the field. Secondly, we analysed 10 different frameworks (Section 2.3) and identified the main characteristics needed for designing our extensible framework. For this purpose we came up with the following questions:

- **Question 1**: *What are the main difficulties of the actual frameworks in order to set up collaborative environments using AIS?*
- **Question 2**: *What are the popular modalities in which the users can interact with AIS?*
- **Question 3**: *What aspects should we take into account when designing and building AIS?*
- **Question 4**: *What are the latest technologies, devices, ideal architecture and possible solutions for implementation of an AIS into an educational institution?*

1.3 Objectives

The research in this thesis covers topics that are strongly related to ambiance systems and the following aspects will be examined in detail:

- determine the current state of the art and current trends of AIS
- study the interaction of the users with the AIS in collaborative environments
- develop a framework using the latest web technologies focusing on a few characteristics unmet in other frameworks studied
- implement and realise an AIS demo in the WISE laboratory
- contribute to the technological evolution of AIS and HCI

1.4 Thesis Structure

This thesis is composed of seven chapters and organised as follows:

Chapter 1 is the introduction; this chapter is divided in four sections: research context, problem statement and research question, objectives and it ends with the description of how the thesis is structured.

Chapter 2 provides the findings of our literature study which we have conducted in order to answer our research questions.

Chapter 3 contains system requirements analyses and defines the classes of users in order to build the new framework that is called BeAware.

Chapter 4 covers the system modelling based on requirements and presents the CTT user tasks, ORM conceptual schema and UML class diagram.

Chapter 5 is about the technical implementation of the framework which contains the framework architecture, system core and GUI.

Chapter 6 is the chapter of this thesis in which are presented three scenarios of how the framework can work.

Chapter 7 presents a general conclusion of this thesis and describes our contribution, limitations, the possible challenges and future work.

2

Literature Review

2.1 Overview

In this chapter we would like to present some related work that has been done with regards to our topic and our findings that support the conducted research. We will start this chapter by discussing a few applications of ambient systems: 3 ambient environments (ambientROOM, Meeting Pot and AwareNews) and 4 ambient displays (Hello.Wall, AuraOrb, Ambient Timer and Flower-shaped Ambient Avatar), and we will end it with an intermediate conclusion (Section 2.2.3).

Subsequently, we will continue with the analysis of 10 frameworks that represents the starting point in finding the essential characteristics of designing our own framework. The frameworks that we analyse are the following: The Context Toolkit, CASanDRA, DMS-CA, HYDRA, WoTKit, CARISMA, RecAm, COSMOS, Feel@Home and Octopus.

In addition, we will make a comparison between those 10 frameworks from the point of view of supported characteristics (Section 2.4) and we will finish with an overall conclusion (Section 2.5).

2.2 Related Ambient Systems

In this section we will review some related work that has been done in the area of ambient information. We start with the subsection *Ambient Environments* where we will be presenting some applications designed for offices and academic environments. Then, we continue with the subsection *Ambient Displays* that focuses mainly on digital displays and shows how the ambient information can be output in various forms. The difference between both subsections is the following: the Ambient Environments have the information flow usually spread out in many environment spaces and the system can contain numerous Ambient Displays (ADs) to produce output of digital information; an AD is constituted as a single device with local information flow and has one or many displays to show ambient media information to the user. This section ends with a *Conclusion*, where we included a comparison between the systems already considered in previous both sections based on their output modalities. Furthermore, the last section highlights the importance of using those modalities and their effect on humans.

2.2.1 Ambient Environments

2.2.1.1 ambientROOM

The ambientROOM [3] is an AIS implemented at MIT Media Laboratory as an enclosed mini-office installation with the dimension of 1.8m x 2.4m. The ambientROOM was augmented with multiple forms of ambient media such as sound, ambient light, shadow, water flow and airflow, in order to establish a new manner of interface between human and digital information. The main purpose of this work is to show the transition of the user centred-attention, so between *foreground* (user that is concentrated at computer to accomplish their tasks) and *background* (ambient displays that shows subtle information to the user), represented in Figure 2.1. With other words, to design a system that communicates ambient information in such a way that cannot distract the user centred-attention from the main task [11]. The user interruption have numerous effects on user emotional state and positive social attribution [12].

The ambientROOM prototype is constituted from sound and light, that are shown in various forms of ambient media displays and controls things such as: water ripples, light patches, natural soundscapes, bottle, clock (Figure 2.2). The *water ripples* display aims to express the awareness of a hamster's activity

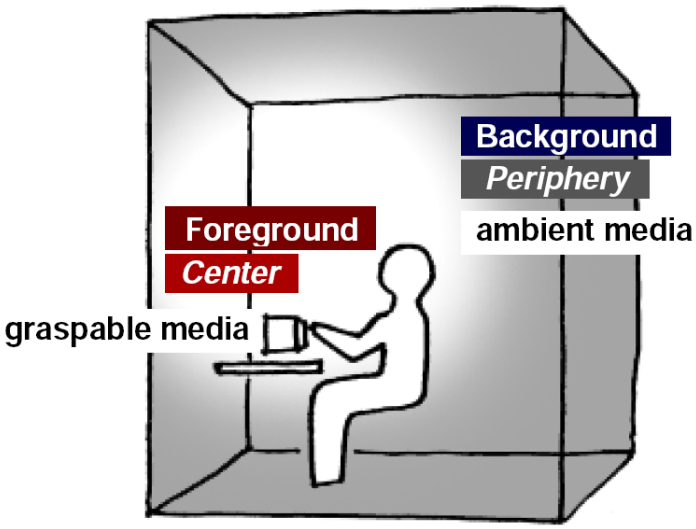

Figure 2.1: Representation of foreground and background media information in physical space — *Source: [13]*

in the laboratory. Originally it was build as a "phicon" (physical icon) that is a physical object representing a hamster. By using multiple inputs such as wheel motion, cage temperature and light level, the hamster activity is shown via a phicon vibrating and as well on a web page. The phicon is equipped with a mini-engine and vibrates in correlation with the hamster activity that is remote from the ambientROOM. However, the user can redirect the hamster activity from the phicon to the water ripples by "pointing" to the ceiling of the room. The output composed of water ripples was created with a solenoid vibrating into a water tank and a lamp that reflects the water's shadow light on the ceiling.

Another ambient media display is represented by *light patches* projected on the wall of ambientROOM. The authors have used sensors collecting information about human presence and movement. An increased number of light patches take place when the number of people in laboratory grows. The third kind of ambient display is called *natural soundscapes*. It uses soundtracks of birds and rainfall in order to communicate ambient information via sounds. The volume and frequency of the sounds is in correlation with the level of the light in the room. Instead of light intensity, the input can be replaced with the number of unread messages or a stock portfolio. The next ambient media is constituted as a control of information via a small *bottle*. By removing the cork of the bottle a soundtrack which contains sounds of vehicular traffic are released in the room. The intensity of the traffic is in conjunction

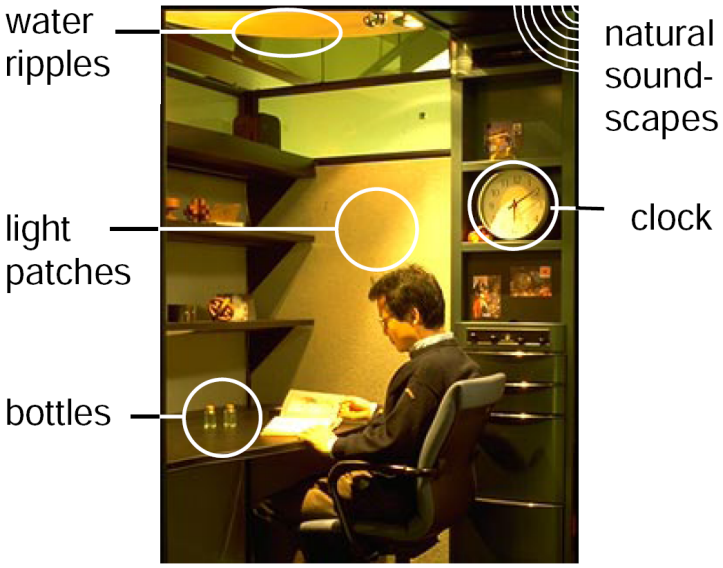

Figure 2.2: Ambient displays and controls in ambientROOM — *Source: [3]*

with the computer traffic network. By adding the cork back to the bottle the vehicular traffic is stopped. The last ambient media element from the ambientROOM is represented by the *clock*. In comparison with the classical clock, this type of clock can be easily manipulated with the hands and has the feature of navigating between time events that happened in the past or will be anticipated in the future. With a manual rotation of the clock in a specific moment of time, the ambient system will shift to that state. In this way users can review the activity display of the moment when they may be absent or anticipate the next events.

To conclude, we found ambientROOM as being an ambient system which reflects the concepts of non-distracting user's attention from current working tasks. The authors have clearly described a number of 3 ambient displays (water ripples, light patches and natural soundscapes) and 2 controls (bottle and clock) in order to handle the ambient information. We have seen how the activity of a remote/external environment, such as hamster activity and human presence, can be converted to ambient information and outputted to ambient displays [14]. We got an overview on how an ambient system can respond to the current environment characteristics (the sound volume and frequency in correlation with the luminosity level or computer traffic network), as shown in Table 2.1. In addition, we examined of how the ambient controls are mapped to physical objects (the bottle that is used as a switch or the clock as a variable between time events).

Input	Output	Data type
Hamster activity	Water ripples (projected on the ceiling)	Integer
Hamster activity	Phicon (vibrating)	Integer
Human presence	Light patches (projected on the wall)	Integer
Intensity of light room	Volume of a natural soundscapes (soundtracks of birds and rainfall)	Integer
Computer traffic network	Frequency of sounds (soundtracks of vehicular traffic)	Integer
Open/close the bottle	Turn on/off the sounds (soundtracks of vehicular traffic)	Boolean
Rotating of the clock	Display past or future events	Time

Table 2.1: The correspondence of information between input and output in ambientROOM system

2.2.1.2 Meeting Pot

The Meeting Pot [15] is an AIS that uses olfactory modality (the information is acquired by people through smell), to transmit ambient information to humans into a multi-room environment. The goal of the Meeting Pot is to inform people in an office that a coffee break takes place in an open space. The system consists of one coffee machine and five coffee aroma generators, as shown in Figure 2.3.

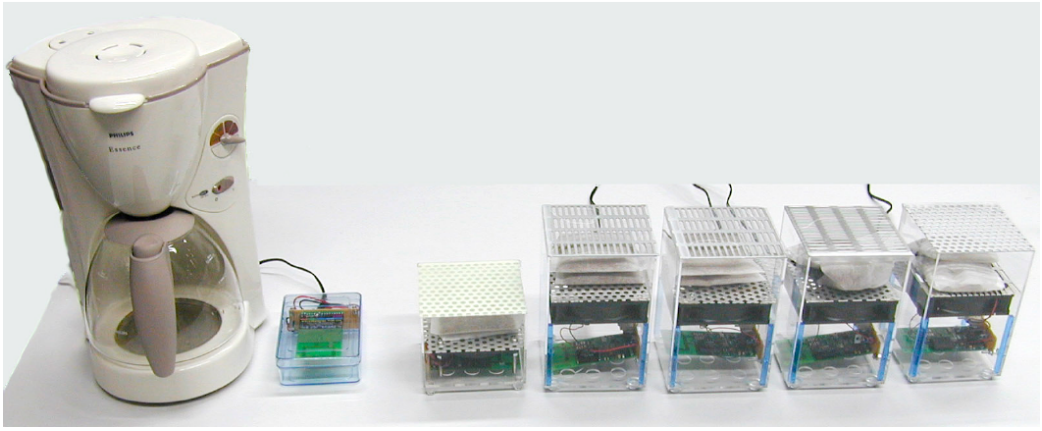

Figure 2.3: The Meeting Pot system, formed from coffee machine and aroma generators — *Source: [15]*

The aroma generators are placed in several rooms and are connected wirelessly (using radio frequency signals) with the emitter of the coffee machine. When the coffee machine is turned on to prepare a coffee, the emitter will send a signal to all receivers of aroma generators. The aroma generators that are equipped with a fan and freeze-dried instant coffee powder, will become active and will produce a smell of coffee in the room. The effect of the smell will subtly inform people that a possible meeting is taking place. The ambient information generated by the aroma generators, have a lesser priority than the main tasks of people. Therefore, busy people can just ignore the smell and not be too much distracted by the ambient notification of a possible break meeting.

The system has been evaluated in the offices of the faculty by a number of 10 participants and during a period of 16 weeks. The authors have compared the system using as output the ambient notification generated by the coffee aroma generator and the email notification. At the end, when interviewing the test participants, the result was that most of them preferred the coffee aroma generator and only a few liked to be notified by email. Those who answered the second variant (the email notification), answered that way because they did not want their behaviour to become visible to the rest of the colleagues in the room.

A positive aspect of this ambient system is represented by the distributed environment and multi-user collaboration. As we have seen the ambient devices are spread into many of the university offices to delivery context awareness. The system uses the human capacity of perceiving and the parallel processing of information in which people can continue with their current tasks if they are unable to join the meeting coffee break.

2.2.1.3 AwareNews

The AwareNews [16] is another AIS designed for a group of people working at an organisation, in order to enhance the awareness concerning news, related topics, events and activities. The reason for this is that in an organisation information is updated or changed frequently, and the classical way of providing access to it like mailing, newsletters, web portals or news feeds, are time consuming. Employers and researchers have to explicitly request or constantly check the sources of information they need. AwareNews was created to provide an easier way to access information without any extra action or interruption from the current user tasks [11].

The AwareNews system consists of peripheral displays that are located in working environments such as offices, laboratories, meeting rooms, printer rooms, hallways, cafeterias or other places. As you can see in Figure 2.4, the AwareNews display is placed on the wall on one of the meeting rooms and is visible from all of the room.

Figure 2.4: AwareNews integrated in a meeting room — *Source: [16]*

On the display filtered information based on the user's context is shown and is refreshed every 30 seconds. The information is loaded from various sources such as web pages, database, RSS news feeds, calendars, and are encapsulated into separate objects. The objects have added additional metadata (category, language, source, author and date) and combined into a single view. AwareNews system include the following objects: *News* objects (consist of a headline and a news summary), *Event* objects (contain information related to meetings, presentations, conferences, lectures, deadlines or other events), *Activity* objects (include information about people, like their changes in documents or their updates in database) and *Memo* objects (contain information that do not fit in previous objects, for instance information concerning announcements, reminders, links to website or publications).

The AwareNews application displays one information item at a time on the screen, that is loaded from the database objects and rendered through HTML via a web browser. The update between the slides is constituted by a smooth

transition with fade effects that has the particularity not to perturb the people working in the proximity of the screen. Moreover, the system contains numerous sensors and a manager of information context allowing the following characteristic: the contrast and brightness of the font is dependent by the importance of the object. The screen background is black and the recent news or the coming soon events are displayed in clear grey compared to the old news that are shown in dark grey. In this way people are directly informed about the importance of a message.

An experiment has been deployed and evaluated in order to test the capability of the AwareNews system. A number of 12 displays with different diagonal sizes (LCD monitors and Plasma screens) were installed in 12 separate rooms of their research offices. The system has been evaluated by a group of 11 participants (1 professor, 7 researchers, 1 secretary and 2 master students), during a period of time of 10 days. The evaluation results have proved that most of participants liked the layout format in which the information is displayed (9 people) and others would like to use the AwareNews system in the near future (9 people), despite of the fact that the continuous display of information can be disturbing sometimes. While some of the people appreciated the characteristic of non-interactivity display, the others preferred to have extra features added to the system such as a *back button* (to go back to the previous slide) or *details button* (to see details about the current slide from the screen). A few participants have mentioned that the screen was not well positioned and the information shown on the screen was repeated or not relevant to the people present in the room.

We conclude that the AwareNews is an essential ambient system which combines multiple sources of information and outputs them into a single view/screen. The system is outstanding by the fact that it is aware about the users present in the office rooms and relies on their user profile preferences by showing them useful-related information. Moreover, the ambient display adjusts the brightness and contrast based on the importance of the information source.

2.2.2 Ambient Displays

2.2.2.1 Hello.Wall

The Hello.Wall [17] is an ambient display [14] created for smart environments, with the goal to increase awareness and collaboration about smart artefacts [18], in local or distributed groups of people. The display has a dimension of 1.8m x 2m and uses an array of 124 LED light cells to output a large range of patterns (Figure 2.5 *(a)*, *(b)*). It has sensing capacity of people presence and ability to collect information about the location of the people in front of the display. Hello.Wall uses the concept of *calm technology*, that it "engages both the centre and the periphery of our attention, and in fact moves back and forth between the two" [11].

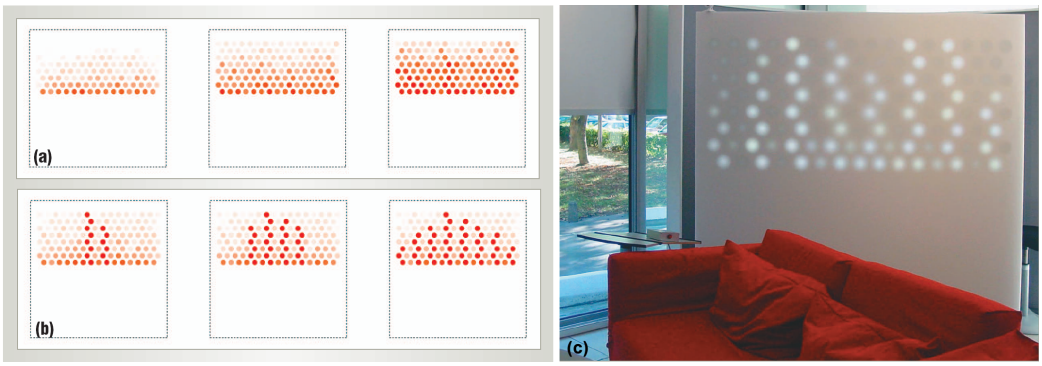

Figure 2.5: Hello.Wall patterns: levels of mood *(a)*, levels of presence *(b)*, Hello.Wall demo running in lounge area *(c)* — *Source: [18]*

People can watch the Hello.Wall display and see various abstract patterns. The patterns can include personal private information (the meaning of a pattern is known only by a person or a specific group of people), which makes it possible to expose the display even in an open public environment. In this way the visitors or people outside of the organisation would consider the display a simple decorative element.

According to the distance, the Hello.Wall includes three distinct communication zones: *ambient zone*, *notification zone* and *interaction zone*, as shown in Figure 2.6. The system senses the presence of people in the ambient zone and continuously shows general patterns such as: *presence* (Figure 2.5 *(b)*) and *mood* (Figure 2.5 *(a)*). The notification zone is designed for personal patterns that are attributed to a specific person from the group. The interaction zone is reserved for the people who directly interact with the display,

via a hand device called *ViewPort*. A ViewPort is composed of two pieces: a reader module and stick with RFID properties. Each stick contains a unique pattern and it can be sent to the display by moving the ViewPort near to the display (up 10 cm).

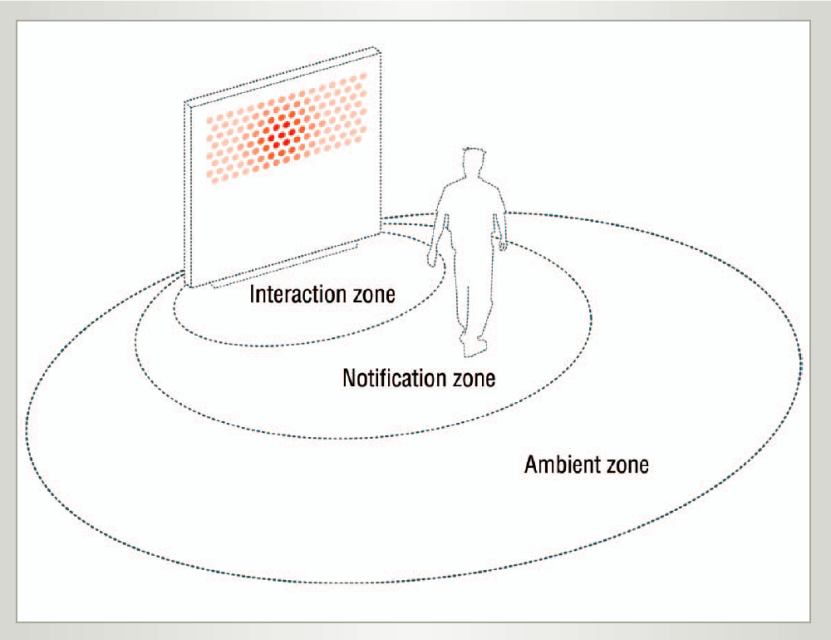

Figure 2.6: Communication zones in Hello.Wall (ambient, notification and interaction) — *Source: [18]*

In order to test the usefulness of Hello.Wall beyond local usage, the authors have built and evaluated a system of two displays in separate locations. They installed and connected the displays in two lounge spaces: one at EDF-LDC in Paris, France and the another one at Fraunhofer IPSI in Darmstadt, Germany. The objective of the experiment was to ensure an informal communication between the remote teams when engaging in a video conference. The three zones (Figure 2.6) have been mapped in both lounges, in order to make people from opposite sides aware of light patterns. When a person enters the ambient zone, the display from another place shows just presence patterns. While the person advances into the notification zone, the system obtains personal information about that person, and sends a specific personal pattern on the remote display. At this time, the remote team is notified about a possible encounter. While the person passes into the interaction zone, they can trigger the system by pushing a button from ViewPort and

state that it is time for a video conference. People from another place will see on the display a specific pattern and have the possibility to accept/reject the invitation (pattern that is shown on the opposite screen). The results of their evaluation was positive, the participants had a good experience in using the system and mentioned that since the system's introduction more video conferences took place.

To conclude, we discovered the Hello.Wall display as a great way of transmitting information between remote people via the ambient lights. Similar to the ambient systems shown above, this type of display does not distract people while they are concentrated on their tasks. Instead of calling or sending an email to notify about a video conference, this system is great because it provides a quick mode of showing the information on the display.

2.2.2.2 AuraOrb

The AuraOrb [19] is an AD that provides ambient light notification, created with the objective of reducing the user distraction from the primary user tasks [11,12]. As visible in Figure 2.7, it is constituted from a spherical modified device (Olympia Infoglobe OL3000[1]), augmented with sensors in order to detect the level of user interest (from a distance of up to 1 meter).

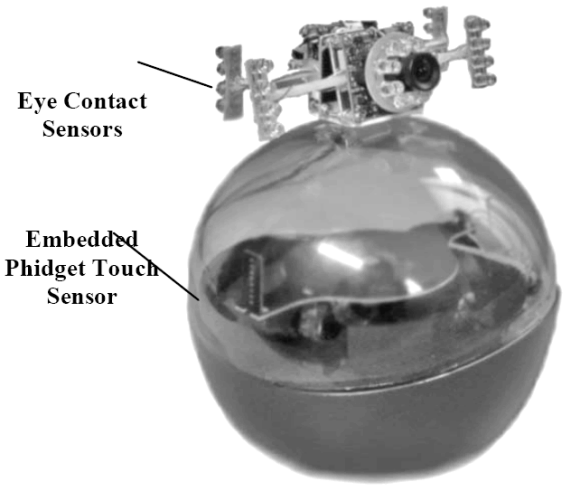

Figure 2.7: AuraOrb and its sensors (contact sensors and touch sensor) — *Source: [19]*

[1] Olympia Infoglobe OL3000: http://goo.gl/6o7Dq9

The upper part includes *eye contact sensors* (cameras with IR LEDs) to recognise if the user attention is focusing on the device and at the bottom contains *capacitive touch sensor* (Phidgets touch sensor[2]) for transferring the notification to another display, for instance to a computer screen. AuraOrb owns two output modalities: an incandescent reflective light and a circular text display (36-character display) that is visible from all angles.

The efficiency of AuraOrb display, has been demonstrated by authors, via an experiment. AuraOrb was configured to inform the user when new email messages arrive. While the user is working at the computer (for instance writing articles in Word[3]), the ambient display stands on the desktop of the user and is ready to notify them. Once a new email arrives into the user inbox, AuraOrb lights up. The user sees the light notification in the background, but they can ignore it for a moment and still keeping the focus in the foreground. Once the user looks at the display, AuraOrb instantly switches from incandescent light and displays the subject and sender of the received email on the circular display as shown in Figure 2.8.

Figure 2.8: AuraOrb displaying the email sender of the message — *Source: [19]*

[2]Phidgets touch sensor: http://goo.gl/FYVcHc
[3]Microsoft Word: https://products.office.com/en-us/word

The user has two options: one is to directly open and read the email on the computer screen (by tapping two times the AuraOrb) or the second is to move their look back to foreground and continue their primary task (by tapping one time the AuraOrb). Once the focus is lost the AuraOrb changes back to its idle state. When multiple emails are received, the display starts with the total number of unread messages.

To conclude, we observed that AuraOrb is an ambient display that provides double input and output modalities for showing the ambient information via light. It is a device that is created to avoid the user distraction by coming events and at the same time offers the possibility to select and access only the important notification (by reading first the header of the message). Moreover, the user can interact with AuraOra using a simple interface, just by tapping the device with the hand and reading the email message on the computer screen.

2.2.2.3 Ambient Timer

The Ambient Timer [20] is an AD designed for users working at the computer with the objective of reminding them of forthcoming events and appointments. It displays notifications via ambient light in a discreet way, without interrupting users from the primary working tasks [11,12]. The Ambient Timer is created as an extension frame with controlled LEDs and it is attached to the backside of a computer monitor as shown in Figure 2.9 *(a)*. It uses progressive change of light colour such as green to red as illustrated in Figure 2.9 *(b)* or green to orange, that illuminates the periphery of the monitor corresponding to the remaining time until the next event.

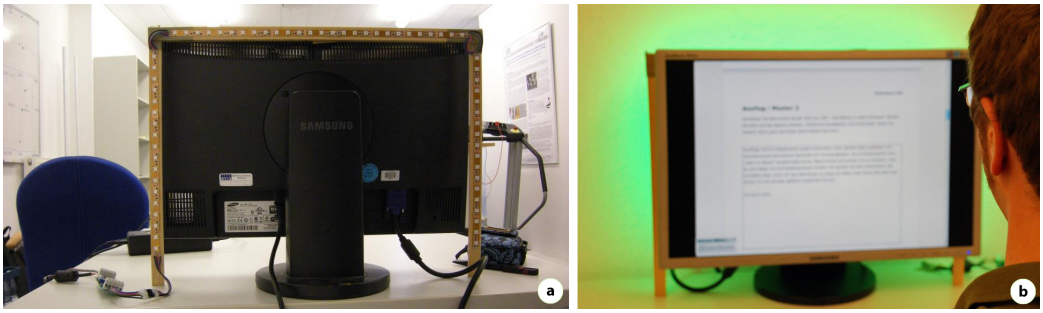

Figure 2.9: The frame with LEDs attached on the backside of the computer monitor *(a)*, Ambient Timer that shows a future event *(b)* — *Source: [20]*

Ambient Timer has been demonstrated and evaluated in an experiment. The experiment investigates the impact of the user interruption, by comparing Ambient Timer (configured in two distinctive ways: transition from green to red and transition from green to orange) with a classical digital clock (combined with a popup shown 2 minutes before the end). The experiment purpose is to correct mistakes in a text document that requires great concentration. The participants (8 male, 4 female and no colour blindness) were asked to complete the correction task in less than 10 minutes, without exceed the time limit. During the experiment they were aware of the remaining time via the Ambient Timer or clock, and a key logger script has been measuring variables such as: *interruptions* (pause between keystrokes), *keystrokes* (number of pressed keys), *keystroke time* (time between 2 keystrokes), *corrected mistakes* (grammar and spelling mistakes), *new introduced mistakes* and *overshoots* (a Boolean value that shows if the participant did not finish the task in time).

The results of the evaluation showed that the Ambient Timer with the progressive green-red colour is the preferred modality by all of the participants because it showed the least interruption from the main task. When using the clock technique, the participants needed to frequently switch their attention between the main task (correcting the document) and the clock in order to check the time left. However, the evaluation has proved that the Ambient Timer provides a discrete way to show forthcoming events using ambient notification on a display very near to the central point of interest.

2.2.2.4 Flower-shaped Ambient Avatar

The Flower-shaped Avatar [21] is an AD, constituted by a physical object created in the form of a flower. The goal of the *flower* is to give notifications and make the user aware of their current seating position in front of a monitor, that can contribute to the user's health and therefore improves their working performance. It uses the principle of calm technology [11], that shows the information in a non-intrusive mode and gives users the freedom to focus on their main tasks.

The Flower-shaped Ambient Avatar tracks the user's position via three sensors: *GYRO sensor* that determines the user's back posture, *FSR sensor* that tracks both the user proximity and the time he remains seated and *PROXIMITY sensor* that detects the user's distance to the monitor as shown in Figure 2.10 *(a)*.

The flower avatar is constructed by the following elements:

a) *flower stem* - created from a bendable silicon tube (20 cm long), that contains a nylon wire on the inside and 13 RGB LEDs

b) *servomotor* - used to pull the nylon wire to bend the flower

c) *speaker* - to output sound notification

d) *micro-controller* - an Arduino Uno[4] used to control the servomotor, the LEDs and the speaker

e) *USB cable* - to connect the micro-controller to PC

f) *casing* - a pot to enclose the micro-controller, the servomotor and the speaker

g) *petals* - only as a decorative element

The information collected from all sensors is sent to a PC (via Bluetooth communication), where a software processes the received information and sends it further to the micro-control of the flower. The position of the flower avatar is adjusted by the servomotor and includes five degrees of freedom in total (Figure 2.10 *(b)*).

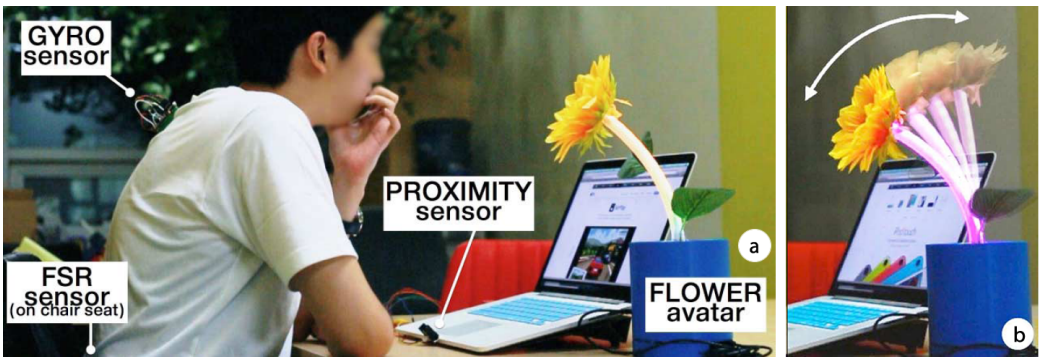

Figure 2.10: Overview of the system *(a)*, Flower avatar with multiple degrees of freedom *(b)* — *Source: [21]*

The flower is able to track the user's *back position*, *sitting time*, *proximity to the screen* and *posture changes*. The back position of the user is calculated and transmitted to the controller of the flower. The flower is bended by the servomotor to a state that is proportional to the user's back. Moreover, the colour of the flower stem is changed in accordance to the bending state: starting with green (showing the correct position of a user, with the analogy that the flower is healthy), yellow (the flower becomes unhealthy) and red (incorrect user position showed by the flower that dries out).

[4]Arduino Uno: https://www.arduino.cc/en/Main/arduinoBoardUno

The sitting time is computed and shown by the flower via the 13 LEDs. Gradually, at every 5 minutes one LED becomes blue, and that symbolises that the flower becomes a "static object". Once all LEDs became blue, the speaker outputs a sound notification in order to notify the user that is time to take a short break. The proximity to the screen is due to the fact that the user is too close to the monitor to read texts with small fonts. The proximity sensor can sense 3 proximity states (*normal*, *near* and *very near*) and sends feedback to the user via the LED lights that start blinking and a sound playing. The system monitors the posture changes at each minute and in case of a higher rate, it notifies the user via a purple LED animation which shows that the user should do some back exercises.

Therefore, this AD shows a new form of physical object that subtly notifies the user. It has multiple output modalities constituted of: light (colour change of the stem), sounds and form of flower (stem bending). It is a decorative ambient media that motivates the user to have a better posture, with the goal to improve work progress and health in an non-distracting mode.

2.2.3 Conclusion

In this section we discuss our findings with regards to the seven ambient systems that we reviewed previously.

Firstly, we noticed some common aspects that appear in the majority of AIS:

a) the ambient systems are designed to work independently without a continuous action from the users side, after they are set up and configured; that means that the hardware and software are not directly exposed to the users, they can run in the background and only display selective information after processing to users.

b) the aim of AIS is to operate without producing any perturbation or distracting the users of their main tasks; moreover, the AIS are made to support and automate their tasks.

c) AIS can interact with users via input modalities that are composed from sensors, smart devices or web services (facts); in order to process the information acquired and provide an interoperability between facts, the systems need to have at least an engine and a set of rules.

Secondly, we noticed that the AIS can have various output modalities. The same kind of digital information can be processed and presented in various forms to the user. Moreover, it is even possible to redirect the information flow to tactile displays [22] as it was seen in ambientROOM. In Table 2.2 below you can see the output modalities for all ambient systems that were reviewed in the current section. It is interesting to see that AIS use additional modalities (such as: physical objects, aroma generators or light) beyond the traditional ones such as screen and sound that are found in all classical computers.

Ambient Sys.	**Type**	**Output modalities**				
		Light	**Sound**	**Aroma**	**Physical Object**	**Screen**
ambientROOM	AIS	✓	✓		✓	
Meeting Pot	AIS			✓		
AwareNews	AIS					✓
Hello.Wall	AD	✓				
AuraOrb	AD	✓				
Ambient Timer	AD	✓				
Flower-shaped Ambient Avatar	AD	✓	✓		✓	
		visual	**auditory**	**olfactory**	**tactile**	**visual**

Table 2.2: The output modalities of the ambient systems

Thirdly, we have seen that every output modality has a specific state in any moment of time and can be controlled in correlation with one or many input parameters. The output state is controlled by a given value that can have one of the following types: binary variable or multi variable. For instance in case of ambientROOM, there are three forms of output: light, sound and physical object. These three modalities can be controlled by the intensity (multi variable), volume (multi variable) and respectively on/off state for physical object (binary variable).

2.3 Related Frameworks

In this section we will further discuss some frameworks that allow us to build systems such as the ones described earlier in Section 2.2.

2.3.1 The Context Toolkit

The Context Toolkit [23,24] is a framework built in Java, designed to quickly develop context-aware applications [25] such as AIS. The framework uses an architecture that provides *"separation of concerns between context sensing and application semantics"* [23], and is composed of the following main components: *Widgets*, *Interpreters* and *Aggregators* (Figure 2.11).

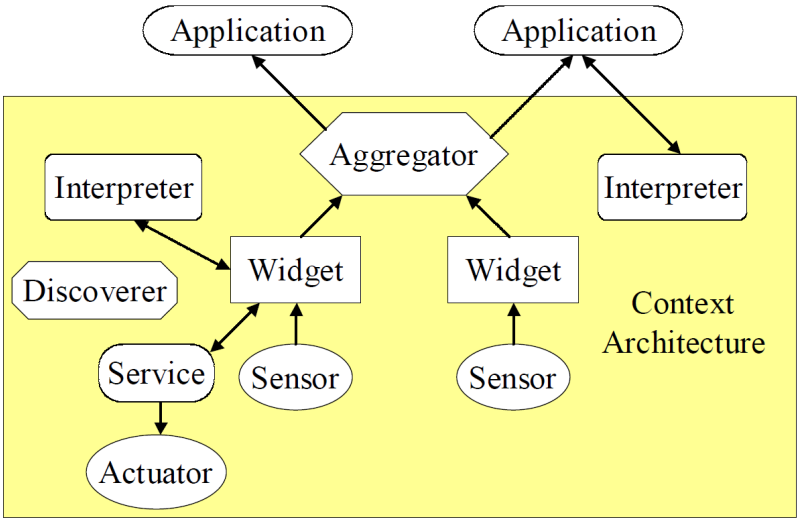

Figure 2.11: The Context Toolkit - Example of configuration architecture — *Source: [25]*

Similar to the GUI Widgets (GTK+[5], Swing[6], jQuery UI[7]) that provide an abstract level between application and presentation, the *context widgets* allow a separation between application and sensors data acquisition. The interpreters are used to translate the type of data from low to high level (for instance it translates the user ID in the first and last name of the user). The aggregators collect the information from the many widgets and store it in a database server for retrieval later.

[5]The GTK+ Project: http://www.gtk.org
[6]Java Swing - GUI: https://docs.oracle.com/javase/tutorial/uiswing/
[7]jQuery UI: https://jqueryui.com

The Context Toolkit has been built with the followings widgets: *IdentityPresence* widget (gives information about people that are detected and triggers callbacks when a person arrives or leaves), *Activity* widget (provides information about the user activity and triggers a callback when the level of noises is changed), *NamePresence* widget (returns the name of the user detected), *PhoneUse* widget (offers information about length of use of the phone), *MachineUse* widget (gives information about the user that is logged in to the computer and the length of the user session) and *GroupURLPresence* widget (returns a unique URL that is associated for each user).

The usage of The Context Toolkit has been demonstrated by the authors via three applications: *In/Out Board*, *Information Display* and *DUMMBO Meeting Board*. *In/Out Board* being similar with a chat status, it provides a list with information about the people that are currently in the office building and the time when they were last time sensed going in or out. *Information Display "displays information relevant to the user's location and identity on a display adjacent to the user. It activates itself as someone approaches it, and the information it displays changes to match the user, her research group, and location"* [23]. The *DUMMBO Meeting Board* application records audio/video people activity when two or more people are in front of the board and provides support for retrieving it later.

The negative aspect is that the tool does not provide resource discovery mechanism to automatically adapt to the changes of the environment. Moreover, the tool does not offer the possibility to design applications with defining custom rules.

2.3.2 CASanDRA

CASanDRA [26] is a framework based on Service Oriented Architecture (SOA) that facilitates the development of AmI applications. CASanDRA is capable of context information acquisition about the user and the environment from the sensors and personal mobile devices. A third-party application can subscribe to the framework in order to receive context information updates or notification calls when events take place.

The framework consists of a *network-based* architecture as illustrated in Figure 2.12 and it is composed of two main components: *CASanDRA Server* (a Java application that serves as host and runs on the remote Linux server) and CASanDRA Lite (a C++ application that is used as client and runs on

Windows Mobile). *CASanDRA* Lite provides support for collecting information from the mobile device sensors (*radio signal strength* measurement, accelerometer, Radio-frequency identification (RFID) and ZigBee SD cards) and making them accessible to *CASanDRA Server* via semantic technology (XML).

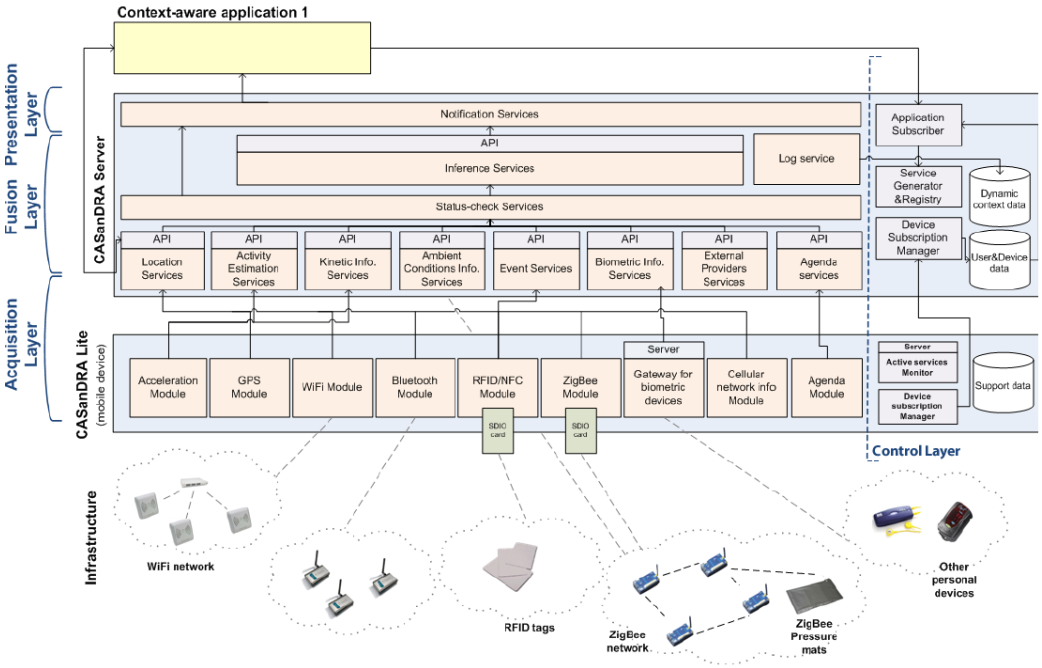

Figure 2.12: CASanDRA architecture — *Source: [26]*

The CASanDRA stack is structured in four layers: *acquisition* (provides direct access to the data sensors), *fusion* (extracts the raw data from the sensors and makes it available for the upper layer), *presentation* (provides notifications base on the sensors data) and *control* (manages the subscribers, the sensors access and handles the events). Once a third-party application subscribes to CASanDRA framework, the control layer will provide access to a XML interface that contains information about the sensors description, type, ports, retrieval methods. The fusion layer generates the corresponding web services associated to the sensors, when the defined condition in the configuration is achieved (for instance it triggers a notification when a specific user moves into a given room).

The positive point of CASanDRA framework is that it provides an interface with the most popular technologies such as: WiFi, Bluetooth and ZigBee.

Moreover, it has built-in algorithms for the following services: indoor *location estimation* that returns the physical coordinates, *proximity detection* based on RFID tags, *kinetic information* relying on the mobile device sensors, *activity inference* that detects the state of a person (standing, sitting, walking), *RFID reading detection* that manages the tags data, *ambient information* that deals with "multihop routing" in a multiple number of WiFi nodes, *biometric data access* that provides heart beat rate and oxymetry information, *access to virtual sensors* constituted as a API to access external sensors and *calendar service* that reads user information from a web calendar.

With all of these, the framework is still limited to the system notification and simple triggers based on conditions defined in the configuration. It does not provide support for rules defined between multiple sensors and events, and remains to the third-party application to implement this operation.

2.3.3 DMS-CA

The Data Management System-Context Architecture or DMS-CA [27] provides a framework architecture designed to develop context-aware applications for Smart Building Environments such as Home or Office automation. The framework is built to collect information from different kinds of sensors (temperature, humidity, light level detection) or contextual information (current user profile, user location, user schedule and current time) and triggers events based on contextual rules (Figure 2.13 *(a)*). Such kind of rule can be for instance when a specific user enters into a meeting room in a defined period of time, that user will receive a copy of a digital document on their personal mobile device.

The architecture of DMS-CA (Figure 2.13 *(b)*) is structured in four layers which contains a separation of the type of information. Each layer is only concerned about the layers nearby and does not need to know how to handle information from the other layers. For instance "Mobile devices do not need to know how to control the rule engine nor how to invoke services, and the server does not need to know how often the rule engine will check the context" [27]. The top layer called *Mobile Devices* is the client layer that is composed of all devices that interacts with the system and directly exchanges information with the server called *Mobile Device Server* layer. As a first step the user defines their own rules using a user interface via a physical mobile device (smartphone, tablet) or a XML rule file. The rules defined into *Mobile Devices* layer are pushed into the server and then injected into the *Rule Engine* layer. The rule engine is checking the defined rules and

once a condition is met, the server triggers commands to the rule owner's mobile device. According to these rules, the system will take the necessary actions in order to deliver to the user the expected services and events. One example might be *"a rule that is triggered when the temperature in an office falls below 22°C on a weekday between 8.30am and 5pm, and this results in a message being sent to the appropriate activator to heat the office"* [27].

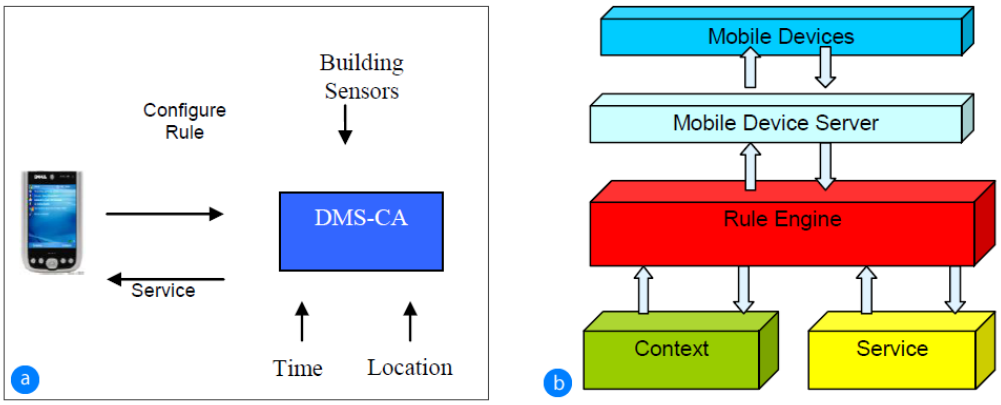

Figure 2.13: Overview of DMS-CA system *(a)*, DMS-CA system architecture *(b)* — *Source: [27]*

The positive point of this framework is that it uses a technique called "Event Driven Rule Checking" in which each rule is associated to the relevant contextual information. Doing this technique means avoiding the regularly checking of the rules at each context change and executing them only when the events are fired. Moreover, the framework contains a mechanism called "time task queue" that minimises the checking of the rules. Each rule is inserted in a queue, sorting and extracting the one with the nearest upcoming time. Using these principles the memory of DMS-CA is reduced considerably and a higher number of rules can be handled in a short time.

The negative point is that the interface for setting the rules is limited to the context that is defined beforehand by the administrator in a XML configuration file. Another aspect is that the service will be returned to the same mobile device in which were defined the rules. The research paper did not describe how the framework can send a service to a different mobile device. Another disadvantage is that the framework is written in Java and designed for PDA. The framework will not work on the latest smartphones or devices that does not support Java.

2.3.4 HYDRA

HYDRA [28] is a IoT middleware designed for developing context-aware applications and services [25], with the goal to integrate sensors and devices into intelligent environment systems. It provides the capabilities of reasoning via ontologies and semantic processing using key-value approach. HYDRA is based on Context Awareness Framework (CAF) that provides the support to deploy a system over the existent wireless and wired networks. CAF is composed of two main components: *Data Acquisition Component* and *Context Manager (CM)*. They ensure a bridge between the data retrieved from the sensors via *Context Providers* component and *Context Consumers* which is modelling the retrieved contextual data (Figure 2.14).

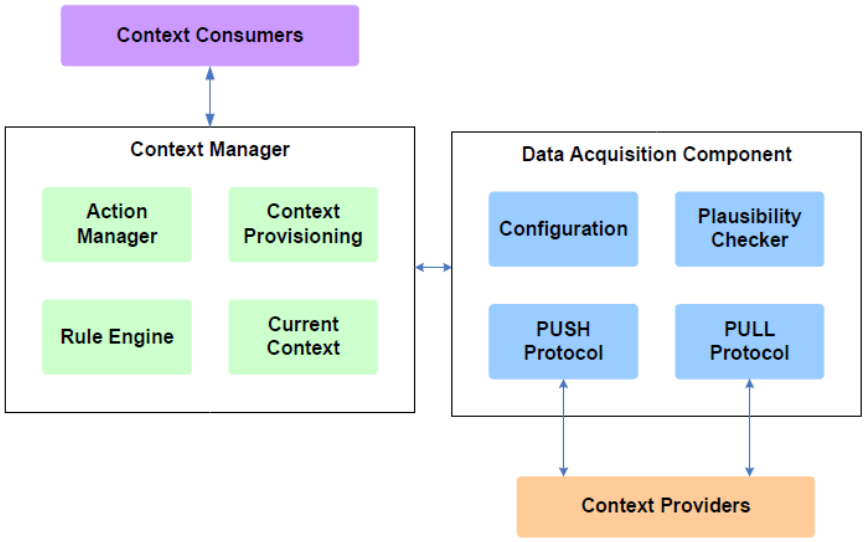

Figure 2.14: Components of the Context Awareness Framework — *Source: modified from [28]*

The component from the right is responsible for run-time handling of identified data received from the *Context Providers* component via the two protocols: *PUSH Protocol* and *PULL Protocol*. The difference between these two protocols is that the PUSH Protocol handles the data pushed by a source, while the PULL Protocol collects the data provided by a source at a custom interval of time. The CM component deals with Context Management, Context Awareness and Context Interpretation. The CM provides the possibility of querying the data and returns it in key-value pair format. Moreover, the core of CM contains a rule engine based on the Drools platform [29], that performs the context reasoning and interpretation relying on the defined rules.

Three distinct types of contexts are modelled in the CAF of the HYDRA project as following: *Device Contexts* (provides low-level contexts for storing and interpreting the data received from the device), *Semantic Contexts* (provides high-level contexts that define the semantic for three core types such as: Location, Environment and Entity) and *Application Contexts* (provides the highest level of contexts where the reasoning is performed, can define the rules and triggers actions when changes in the context take place).

The essential point of HYDRA is that the CM offers the possibility to create systems that can reason and interpret the data received from the source based on the pluggable rules configured in the Drools platform. The content of a Drools rule is constituted of a set of WHEN conditions (LHS - left hand side) and a set of THEN actions (RHS - right hand side) that are fired when the set of WHEN conditions are satisfied. *Context Consumers* is the end component that uses the data outputted by the CM component and makes important changes into context. The *Context Consumers* receives the contextual information via three modalities: *Context Querying* (uses a set of arguments to query the CM component to return a set of data), *Context Sensitive Actions* (*Context Consumers* is asynchronously informed about any change in context) and *Context Storage* (provides access to the XML serialized format of status of the context that has been saved previously and stored remotely).

The main advantage of using HYDRA is that it provides an adaptive context-aware framework which is reasoning over an advanced rule engine and directly interprets the data retrieved from a device. It uses ontologies to store the type and meta-information about devices and semantic description of data using a key-value approach to support the interoperability among devices [30]. The HYDRA prototype has been validated in real end-user scenarios including the domain of building automation [31].

2.3.5 WoTKit

The WoTKit [32] is a lightweight Web of Things (WoT) toolkit that provides a core for rapid developing of WoT applications [33] in order to simplify the way of how the end-user can find, control, visualise and share the data among devices. WoTKit was developed in order to satisfy the following set of requirements: *Integration* (the integration of applications with WoTKit is realised via a unique web server which is made public the access to all developers), *Visualisation* (WoTKit offers advanced tools based on *Google*

chart tools[8] to draw and represent the data), *Processing* (simple to create mashups and configurable processing components using a visual programming interface), *Sharing* (publishing and allowing others to access the things over the web) and *Advanced Application Support* (WoTKit provides a suitable API that allows the developers to take over the data and create their own applications).

WoTKit is built in Java on top of a popular framework called *Spring*[9] and uses the *Apache ActiveMQ*[10] message broker to deliver the data collected from the sensors to the application consumer having low latency. The architecture of WoTKit is shown in the Figure 2.15 *(a)*. WoTKit provides RESTful API services, in which the external application can control things, get historical information and register new things to the system by posting in JSON the sensor data in JSON format (sensor name, location, accessibility, type, display name and units) to a custom URL, similar to the way of how it is done in Node-RED[11] [34]. Moreover, it contains a component called *Process Engine*, that processes the data received from the sensors by using a multithreaded schedule (Figure 2.15 *(b)*). The *Process Engine* is listening for any message that arrives and it is added into the queue. The scheduler looks for the next message present in the queue and it delivers it to the next available pipe to be processed.

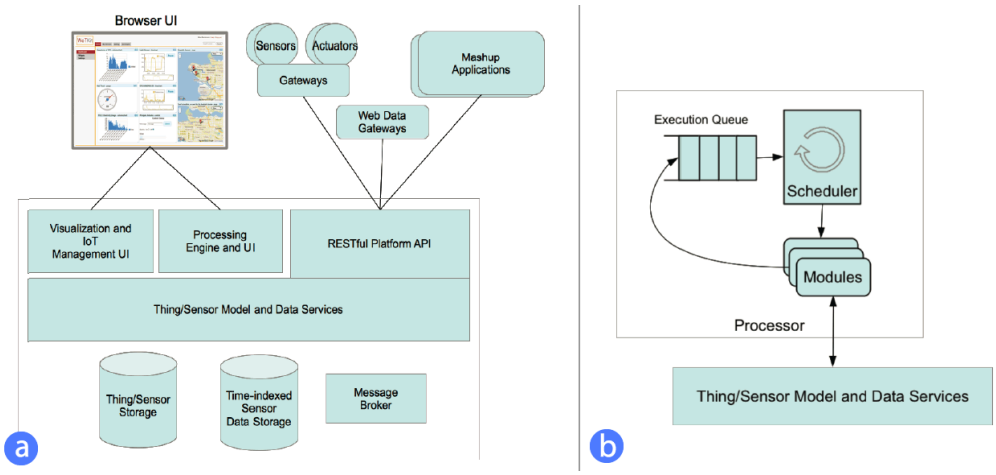

Figure 2.15: WoTKit system architecture *(a)*, WoTKit processing engine *(b)* — *Source: [32]*

[8]Google chart tools: https://developers.google.com/chart/
[9]Spring Framework: http://spring.io
[10]Apache ActiveMQ: http://activemq.apache.org
[11]Node-RED tool: http://nodered.org

The main advantage of WoTKit is that it provides common integration of the sensors via REST architecture and offers the possibility to draw the flows between components similar to Yahoo Pipes. Moreover, it offers an easy way to create visualisation of the data received from the sensors. As disadvantage, WoTKit does not offer the possibility to create rules between devices using sensor's data and counts on the external applications to do this job.

2.3.6 Other Frameworks

CARISMA

CARISMA or Context-Aware Reflective middleware System for Mobile Applications [35] is a mobile computing middleware that is dynamically adapted to the changes of the environment context. The changes of context are handled by the framework using policies and relying on the user profile preferences. When different policies take place in the same context to deliver a service, conflicts may occur. CARISMA's architecture is based on a microeconomic mechanism for conflict resolution, that is similar with a sealed-bid auction. In this way the application decides which of the policies should be applied in order to deliver the service regarding to the user expectations.

RecAm

RecAm [36] is a context-aware framework for multimedia recommendation that collects the context information from AmI environments and constructs a user profile for each individual or group of individuals. The framework uses an algorithm that relies on the user choices from the past in order to personalise the recommendations and to predict the user preferences. Then, the initial algorithm is extended with the PLSA algorithm — Probabilistic Latent Semantic Analysis [37] that has a positive impact on RecAm's performance in predicting accuracy between items and users.

COSMOS

COSMOS [38] is a framework used for processing context information and provides isolation of data over 3 layers: *context collector layer* (obtaining the raw data about the environment via hardware devices and user preferences), *context processor layer* (filtering and computing of raw data) and *decision making* (identifying and taking actions based on the processed data). COSMOS introduces the concept of *context nodes* that represents each part of the context information and is used to assemble the *context management policies*. The context management policies offer adaptation of the application based on the environment.

Feel@Home

Feel@Home [39] is a cross-domain context management framework that allows to build applications that interact between multiple environments. The paper approaches the context interaction between three different environments such as *smart home* (manages the status of the house, user location and user activity), *smart office* (manages the personal location and activity of a user or a group of users) and *outdoor* (manage the user geo-location, user movement type and network status). The framework uses semantic web language — OWL[12], is composed of a *global routing scheme* and supports two context producer-consumer patterns of interaction such as *intra-domain* and *cross-domain*. The global routing scheme includes *remote queries from applications*, *global administration server* and *domain context managers*. The *global administration server* delivers the query to the corresponding domain context and obtains back the result. The *domain context managers* includes wrappers for collecting the information from sensors or web services, a context aggregator that triggers the actions, a semantic reasoner and access to the context already stored.

Octopus

Octopus [40] is a dynamically extensible middleware framework that allows users to build applications for home or office environments for connecting new physical objects over the existing network infrastructure (network printers and network controllable power supplies). The framework provides abstraction layers that isolate the hardware of the sensor networks and offer an interface for fetching and subscribing to data through a semantic URL scheme. Octopus uses *solvers*, that consist of API for adding/removing modules to the system in order to perform data processing and analysis over the new sensors introduced. Multiple solvers can be combined to perform advanced operations that bridges the gap between internet of data and the IoT.

2.4 Comparison between Current Frameworks

In this section we present the analysis of the frameworks that have been described in Section 2.3 as related work. Based on our research we found a set of characteristics that are defined below and with the aim to be part of the optimal framework used in ambient environments. A couple of framework characteristics are related to the environment context while others are focusing on the framework architecture.

[12]Web Ontology Language: http://www.w3.org/2001/sw/wiki/OWL

Firstly, we define the set of characteristics that have been identified. Secondly, we show an overview that contains a comparison table (Table 2.3) with all the frameworks have been already described and the characteristics they have. Then, we present our conclusions by discussing the frameworks and their characteristics.

2.4.1 Framework Characteristics

Context Personalisation is one characteristic of the framework in which the system relies on the user's preferences. The system can detect and identify each user who is present in an ambient environment and dynamically adjusts the settings in order to satisfy the users expectation.

Context Adaptation is a framework characteristic that allows multiple systems to identify the same user, load their rules and adapt the context of the environment based on their preferences.

Context Reasoning refers to the fact that the system owns knowledge about the current context, can take decisions relying on the user rules and triggers system events.

Context Acquisition represents the component of framework that collects contextual data from the devices or sensors in raw format.

Context Discovery denotes that the framework auto-discovers the sensors and devices attached to the system without any additional software development [41].

Context History is a framework feature that offers the possibility to store the contextual data and retrieve it later for processing purposes.

Modularity is a property of the framework in which the architecture is split in multi-layers and provides a separation of concerns between layers or modules.

API is a set of specifications, routines, and tools that provides an interface for a third-party application to handle the sensor data provided by the core framework or events obtained by reasoning.

WOA is an architectural approach of SOA in which the framework provides access to resources via URL and manipulates the resources using REST web services [8].

Visualisation is a feature of the framework that allows the visualisation of contextual data in a graphical format such as charts, plots, and graphs.

2.4.2 Overview

By considering the characteristics defined in the previous section (Section 2.4.1) we notice that the context and the architecture of the frameworks varies. In Table 2.3 is represented a comparison between all the frameworks and the symbol (✓) is used where a specific characteristic is part of the framework.

Framework	Context						Modularity	API	WOA	Visualisation
	Personalisation	Adaptation	Reasoning	Acquisition	Discovery	History				
The Context Toolkit	✓			✓		✓	✓		✓	
CASanDRA				✓			✓	✓	✓	
DMS-CA	✓		✓	✓			✓			
HYDRA	✓		✓	✓		✓	✓			
WoTKit						✓	✓	✓	✓	✓
CARISMA	✓		✓		✓			✓		
RecAm	✓		✓			✓				
COSMOS	✓		✓	✓			✓			
Feel@Home	✓	✓	✓	✓		✓	✓	✓		
Octopus				✓			✓	✓	✓	

Table 2.3: Comparison between different frameworks
API - Application Programming Interface; **WOA** - Web-Oriented Architecture;

2.4.3 Discussion regarding Current Frameworks

The first characteristic (*Context Personalisation*) is missing in three of ten frameworks (CASanDRA, WoTKit and Octopus). These three frameworks do not have built-in any module in order to sense and identify the users that are present in the near proximity of the system. This characteristic becomes indispensable for an ambient system that adjusts the context settings based on the user preferences.

The second characteristic (*Context Adaptation*) is missing from most of the frameworks. The only framework that contains this characteristic is *Feel@Home*. The authors have been focused to develop a framework that can sense the presence of any user in multiple environments such as the office, home and outdoor. In their framework the user location is known at any moment and environment, and the context can be adjusted to the user preferences. Because our framework can be deployed in multiple laboratories this characteristic is required for our model.

Context Reasoning is not marked for some of the frameworks. Four out of the ten frameworks that we have considered as related work do not have the feature of reasoning. They do not provide a way to set rules or possibility to define relations between devices, and remains at the developer responsibility to program these functions. In our framework model we focus on the characteristic of *Context Reasoning* and we want to offer an easy way to define rules between devices for users that have no programming experience.

Seven of ten frameworks have a context acquisition mechanism that collects the data from the sensors. *WoTKit* and *CARISMA* use the data that is available from web services or external applications via the API.

The characteristic of *Context Discovery* is only present at one of the ten frameworks. *CARISMA* provides the function of context discovery that allows the information regarding sensors and devices to be attached, modelled and stored. We use this characteristic as a requirement for our model.

Just half of our list with related frameworks use the concept of *Context History*. They store all the information that are passing the framework and add timestamps for retrieving it later.

The *Modularity* characteristic is present in eight of the frameworks. Most of the frameworks use an architecture that is divided into modules in order to simplify the development and maintenance of the software.

Half of the frameworks provide access via API and four of them offers the WOA capabilities. The only framework that offers the visualisation of the data is *WoTKit*.

2.5 Conclusion

In this chapter we have seen a set of applications with regards to the use of AIS. As well as, we figured out how they are sensing and collecting ambient information being aware of the current environment and therefore triggering external facts by displaying the information via an output modality (Section 2.2). Furthermore, we have reviewed a series of frameworks and their characteristics which help to build ambient systems (Section 2.3).

Nevertheless, the actual frameworks provide just a limited set of the features that we need, in order to be able to set up an ambient system in collaborative laboratories. Even most of the frameworks support the *modularity* feature, they do not provide "extensibility". As it was described earlier, the *modularity* feature refers to the framework architecture or how the framework is structured. The "extensibility" is a software design principle that allows to add new functionality to a software. In most of the frameworks, the "extensibility" is missing and this makes it difficult to extend a framework and further develop it, which usually ends up with stopping at the prototype phase. For instance an "extensibility" issue they have is that they are not compatible with the latest updates of a version, new programming languages or future hardware [42].

By taking into consideration all of these cons, we decided to develop our own framework from scratch and called it "BeAware". The reason of starting our framework from scratch is that the actual frameworks do not provide the flexibility needed. Firstly, we would like to build a lightweight and accessible framework. Secondly, we would like to create our own design for the graphical user interface of our framework, in order to manage the data (rules, facts, users and permissions). Thirdly, to develop a framework that supports "extensibility" and the use of the latest web technologies such as Node.js[13], jQuery[14], PHP7[15] & CodeIgniter[16], MongoDB[17], and many others that will be described in Chapter 5. Besides, we would like our framework to include the following essential characteristics: Context Personalisation, Context Adaptation, Context Reasoning, Context Acquisition, Context Discovery, Context History, Modularity, API and WOA.

[13]Node.js: https://nodejs.org
[14]jQuery: https://jquery.com
[15]PHP7: http://php.net
[16]CodeIgniter: https://www.codeigniter.com
[17]MongoDB: https://www.mongodb.com

3

System Requirements Analysis & User Classes

Having the literature study and the characteristics of different frameworks identified in the previous chapter (Chapter 2), we establish the requirements for our system model in this chapter. We start by defining the classes of users, which provide information about the users who are going to use our framework. Then, we continue with describing the requirements of our system that give information about the tasks and functionalities, and how the users will use these functionalities of the framework.

3.1 Mission Statement

Purpose

The purpose of the project is to develop an extensible framework that permits to quickly deploy customised AIS. The model must allow the users to easily connect smart devices and web services into a network, and define rules between them via a graphical user interface.

Subject

The framework is designed to be used for creating collaborative ambient environments in academic institutions such as laboratories, research rooms and meeting rooms. The first place where the system will be deployed is the WISE research laboratory within the Vrije Universiteit Brussel, a university in Belgium.

Target audiences

The audience of our framework is composed of the following categories of users: teachers (academic staff who have high level of access — professors, researchers, PhDs), students (people who have low level of access — undergraduate students, postgraduate students), system administrators and university visitors.

3.2 User Classes and User Characteristics

3.2.1 User Classes

As we have seen in the Section 3.1 — Target audiences, we have a number of 4 different categories of users. From those categories we will start working with only 2 main classes of users that will interact with the interface of the framework: teachers and students (Figure 3.1). The visitors are interacting just with the ambient system and not with the framework interface, so in this case it is not necessary to consider them in one of our user classes. Both classes, the teacher and student can become system administrators, therefore we are excluding the category of system administrators when describing their user characteristics. Further, we will present the audience characteristics of both selected user classes.

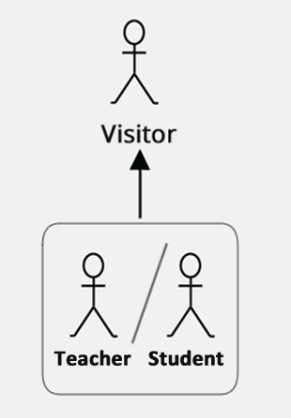

Figure 3.1: Audience class hierarchy of BeAware freamework

3.2.2 User Characteristics

***Teacher* Class**

- Languages issues: are able to communicate in English
- Age: 30-65 years old
- Gender: no limitation
- Web experience: have reasonable experience with websites
- Frequency of use: often
- Type of user: primary
- Task knowledge: Have extent of task knowledge

***Student* Class**

- Languages issues: are able to communicate in English
- Age: 20-30 years old
- Gender: no limitation
- Web experience: have good experience with websites
- Frequency of use: rarely
- Type of user: indirect user
- Task knowledge: No extent of task knowledge

3.3 Requirements Analysis

3.3.1 Functional Requirements

The functional requirements describe all the activities or tasks that could be done by the users via the GUI. We start to describe the tasks of the *Student* class that has the lesser permission into application, continue with the *Teacher* class, and finishing with *Administrator* class who has no restriction at all in the web application.

3.3.1.1 Student Class

A user who belongs to *Student* class can do the following tasks:

1. *Log* into the web application using a username and a password. Can *log out*
2. Request to reset their user password
3. Update their profile information:
    - Username

- Email address
- First name
- Last name
- MAC address (the MAC address of the user's smartphone that allows the system to track the user)

4. Change their account password
5. Handle and manage *Rules*:
    - View the list of rules they created previously, having the following attributes:
        - Rule ID
        - Rule status indicator (*enabled*, *not active* - enabled but user absent, *disabled*)
        - Rule name
        - User presence (*yes* or *no*, depending if the rules is always enabled or only when user is present)
        - Datetime when rule is added
        - Datetime when rule is updated
        - Rule result (*true* or *false* based on rule result)
    - Sort and search the list of rules
    - Enable/Disable the rules (one by one/multiple selection in the same time)
    - Create new rules and update the rules created previous, by asking the field questions:
        - Rule name
        - When to treat this rule (always or only the user is present)
        - Rule status (enabled or disabled)
        - Rule conditions - WHEN (a block where the user can define a set of rules composed from logic "OR" and "AND" variable conditions)
        - Rule consequence - THEN (a block where the user can define the result consequence of the rule)
    - Delete the rules created
    - Share one or multiple rules with other users using the following methods:
        - Share rules by searching for a user account
        - Share rules by filling in an email address
        - Share rules by using a web address
        - Share rules on social media (Facebook and Twitter)

6. The user is notified when they receives new rules that are shared by other users and has the possibility to review and accept/reject

3.3.1.2 Teacher Class

The users who belong to the *Teacher* class can do **all the tasks that the users from the *Student* class can do**, plus the following tasks:

1. Select between the laboratories in which they gained access
2. Handle and manange *Facts* (Smart devices/Web services):
    - View the list of facts that they created previous, having the attributes:
        - Fact ID
        - Fact status indicator (*enabled* or *disabled*)
        - Fact name
        - Fact description
        - Fact category (with possibility of click-to-filter)
        - Fact type (*input* or *output*)
        - Value (the fact value that is fetched)
        - Acquisition time
        - Interval (fetching or retaining interval time)
    - Sort and search the list of facts
    - Enable/Disable the facts (one by one/multiple selection in the same time)
    - Create new facts and edit the facts created previous, by asking the field questions:
        - Fact category (using autocomplete[1] or create new category if it is not found in the list)
        - Fact name
        - Fact description
        - Fact type (*input* or *output*)
        - Rule unit
        - Channel type (*separate* or *common*)
        - List with channels (labs), where user has gained access (which contains the user can define URL source and retrieving interval/retaining interval for each channel)

[1] e.g. jQuery UI autocomplete: https://jqueryui.com/autocomplete/

3. Handle the devices found through the network:
    - View, search and sort the list of devices auto-discovered
    - Select and import the devices in the list with facts
    - Clear and discover new devices through the network

3.3.1.3 Administrator Class

As explained previously, any user that belong to one of both classes (*Student* or *Teacher*) can become part of this class if any existing administrator gives permission. In this case we will describe as well the tasks of the *Administrator* class. The users who belong to the *Administrator* class can do **all the tasks that the users from the *Teacher* class can do**, plus the following tasks:

1. Handle and manage *Users*:
    - View the list of all users, having the attributes:
        - Email
        - User status indicator (*online* or *offline*, shows if the user is present in the lab)
        - User name
        - First name
        - Last name
        - Super admin (*yes* or *no*, indicates if the user is admin)
        - Active (*yes* or *no*, indicates if the account of user is active)
    - Sort and search the list of users
    - Enable/Disable the users
    - Add new users and update update the user profiles of the existing users
    - Delete the existing users
    - Change the user permissions
2. Has access to manage the permission for all users regarding lab permission (read and edit)
3. Login over other users and access their account
4. Update the ambient system configuration
5. View the list of rules of all the users (having information regarding who added/modified)
6. Has the possibility to select and manage all the laboratories
7. Change the group to where the users belong

3.3.2 Core Functionality Requirements

The core functionality requirements indicate the main components that should be supported by the framework. According to Chemuturi [43, Chapter 2], *the core functionality requirements are those functionalities of the product without which, the product is not useful for the users* and *the main purpose of the software development is to fulfil this core functionality.* Therefore, this section will contain the core requirements of our framework.

The framework needs to work in real-time and must perform the following tasks/functions:

1. Detect the User Presence:
   - The framework must detect any user presence based on their smartphone connected to the WiFi router
   - The framework must identify the laboratory where the user is situated
2. Execute a Rule Engine:
   - Shall continuously check all the rules to figure out if any condition is fulfilled, and trigger the consequence event defined in the GUI
   - The rules which depend on user presence, must be considered by framework only in a specific lab and only when the user is detected to be present in that lab
   - The changes regarding the rules accomplished through GUI, need to be immediately reflected in the core
3. Do Data Acquisition:
   - The framework must be able to retrieve in parallel, the information provided by the facts (smart devices or web services)
4. Do Device Discovery:
   - The framework must scan the local network regularly, find new smart devices that are attached to the network and make them available to the GUI
5. Control Multimedia Presentations:
   - The framework must be able to load multimedia presentations (video, images, text) and show them on digital displays
6. Use the existing Network Infrastructure:
   - The smart devices must be connected using the current network connection interface: WiFi or wired Ethernet

3.3.3 Data Requirements

Data Type

- All the information that include real-time processing (e.g. rules, facts, environments, devices) must be stored in a NoSQL database. The motivation of choosing NoSQL database is that it provides schema-less data model (no strict structure definition), low latency, high performance and highly scalable.
- The *rule condition* and *rule consequence* must be stored in the database in a JSON format. The motivation of choosing JSON is because of the followings: easy to query, takes less data space, easy accessible in many programming languages and more readable.
- The information used for GUI shall be stored in a NoSQL database or relational database
- The information retrieved from facts (smart devices/web services) are offered in JSON format.
- The configuration of devices should be made in YAML format. YAML has the main advantage that is human-readable.
- The framework shall send information to facts using REST services.
- The control of multimedia presentations must be realised using web sockets.

3.3.4 Environmental Requirements

The environment requirements describe the factors that can indirectly interact with our ambient system and it is important to take them into account since the design phase.

Physical Requirements

- Movement
  - The users can frequently go inside/outside of the lab for short intervals of time. In this case the user presence flag can toggle between being present or absent. To avoid this from happening, the system must retain the value of being present for an interval of time.
  - In order to detect the presence of all of the users in the lab, the lab must be fully covered with a WiFi signal from the router. Otherwise, the users that move out of the Wi-fi coverage will not be detected by the system.

  - When the user is moving between labs, the system must instantly recognise their presence in the last lab and revoke their identity in the previous labs.
  - The movement is also an important factor when we use devices based on the motion sensor or image recognition

- Noise
  - In the case that devices audio-based (microphones and speakers) will be connected to the system, it is necessary to take into account that in the lab may be noises coming from other people that are in the lab.

3.3.5 Usability Requirements

The *usability requirements* indicate a measure for making the system easily to use and be appropriate to users needs and requirements. Conform to ISO-9241-11 [44], the usability is defined as *the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use*. Hence, for our system we identified the following main usability requirements:

1. Defining a rule/fact should be possible after a minimal training
   - Motivation: Since this is not the main activity of the members of the lab, the new users need to figure out in an easier way regarding system functionalities. University members do not have time to provide long training.
   - Measuring Concept: Learnability
   - Measuring Method: Task scenario after a short training
     - Result: Percentage of errors
   - Criteria for judging (percentage of errors that is assumed for completing the task):
     - Current level: n.a.
     - Worst level: 20 %
     - Planned level: 10 %
     - Best level: 5 %
2. When adding a rule/fact, the system must inform the user in case that any required value is not filled in or is invalid
   - Motivation: The system must provide feedback to user about their interaction with the system

- Measuring Concept: Learnability (Feedback)
- Measuring Method: Different task scenarios
  - Result: Percentage of tasks successfully completed
- Criteria for judging (percentage of errors that is assumed for completing the task):
  - Current level: n.a.
  - Worst level: 90 %
  - Planned level: 100 %
  - Best level: 100 %

3. Creating a new rule must be possible in less than 3 minutes
   - Motivation: The users must be able to quickly add new rules without taking time from other lab tasks
   - Measuring Concept: Quality of task performance
   - Measuring Method: Task scenario
     - Result: Time to complete the task
   - Criteria for judging (time in minutes that is assumed for completing the task):
     - Current level: n.a.
     - Worst level: 5 minutes
     - Planned level: 3 minutes
     - Best level: 2 minutes
4. Adding a new rule must be possible in less than 2 minutes
   - Motivation: The users must be able to quickly add new facts without taking time from other lab tasks
   - Measuring Concept: Quality of task performance
   - Measuring Method: Task scenario
     - Result: Time to complete the task
   - Criteria for judging (time in minutes that is assumed for completing the task):
     - Current level: n.a.
     - Worst level: 3 minutes
     - Planned level: 2 minutes
     - Best level: 1 minute
5. Allowing the users to quickly find the information already saved (rules or facts)
   - Motivation: The users must be able to quickly find all the information that they saved previously. They do not need to go

through dozens of pages to find a rule or a fact.

- Measuring Concept: Quality of task performance
- Measuring Method: Task scenario
  - Result: Time to complete the task
- Criteria for judging (time in seconds that is assumed for completing the task):
  - Current level: n.a.
  - Worst level: 1 minute
  - Planned level: 20 seconds
  - Best level: 10 seconds

4

System Modelling and Design

In this chapter we discuss the overall design of the proposed framework BeAware. Based on the system requirements identified in the previous chapter, we define the set of user tasks and user object models. The user object models contain the ORM conceptual diagram which shows the relations between objects, and UML class diagram that provides a visualisation of the system classes.

4.1 User Tasks Analysis

This section presents the relation between tasks, the way how the users perform the tasks and identifies the relevant tasks for our framework. We model the user tasks using CTT notation (Concurrent Task Trees) with a tool called CTTE (ConcurTaskTrees Environment) that provides support for developing and analysing of task models in order to improve the design of the software applications [45].

4.1.1 Task Models using CTT

As we have seen already in the Section 3.2, we have 2 main classes of users: Student and Teacher. The users from both classes can become Administrator, therefore we discuss this class apart.

4.1.1.1 Tasks modelling of Student Class

1. User Login

The task model "User Login" (Figure 4.1), is the task that allows the users to log into application and have access to website functionality associated with their account.

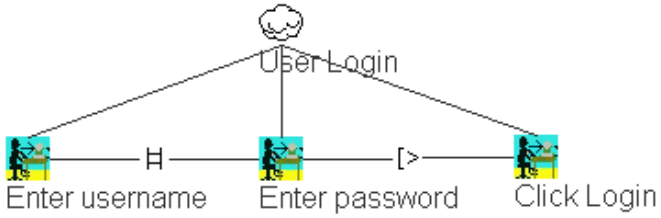

Figure 4.1: Task model for *User Login*

2. Reset Password

The task model "Reset Password" (Figure 4.2), allows users to reset the password in case that they forgot it. The task contains the following steps: the users fill in the email address, they receive an email that contains a URL to confirm, they click the URL and fill in the new password.

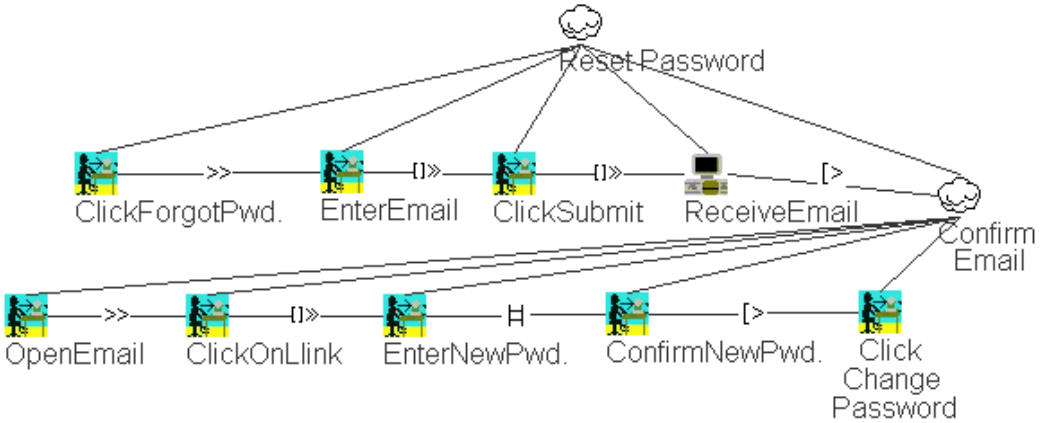

Figure 4.2: Task model for *Reset Password*

3. Update User Profile

The task model "Update User Profile" (Figure 4.3), is the task that allows users to change the user information regarding their account profiles.

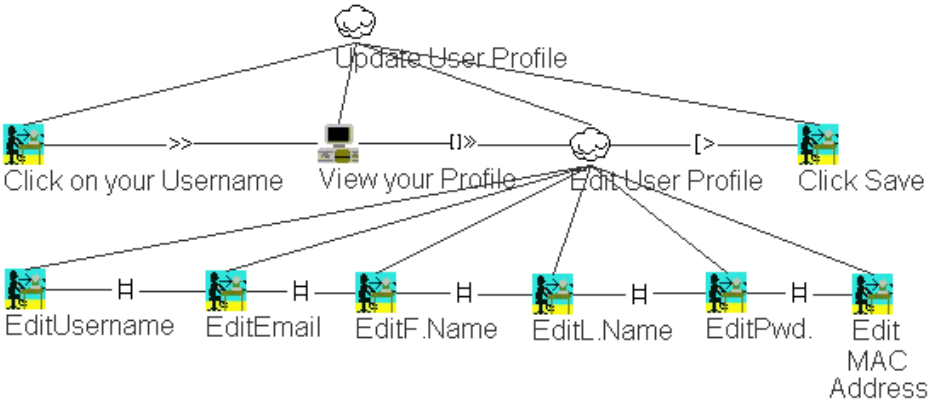

Figure 4.3: Task model for *Update User Profile*

4. Manage Rules

The abstract task "Manage Rules" (Figure 4.4), is the task that allows users to find, select and add rules. The task is decomposed in 4 subtasks which will be described in the next paragraphs.

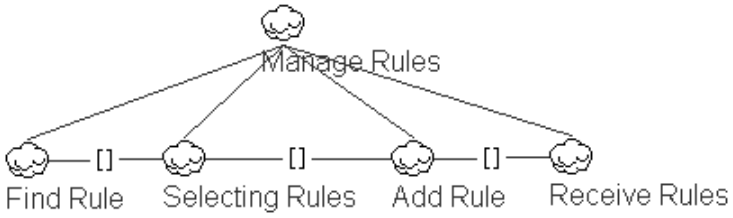

Figure 4.4: Task model for *Manage Rules*

5. Find Rule

The task model "Find Rule" (Figure 4.5) allows users to filter the rules based on a search key that is entered into the filed "Search".

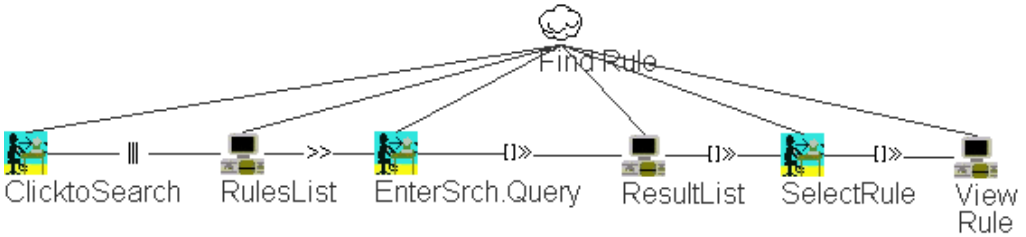

Figure 4.5: Task model for *Find Rule*

6. Selecting Rules

The task model "Selecting Rules" (Figure 4.6), is the task that allows users to select rules for specific operations such as: Edit, Delete, Enable/Disable or Share. These abstract subtasks are modelled later in the next paragraphs.

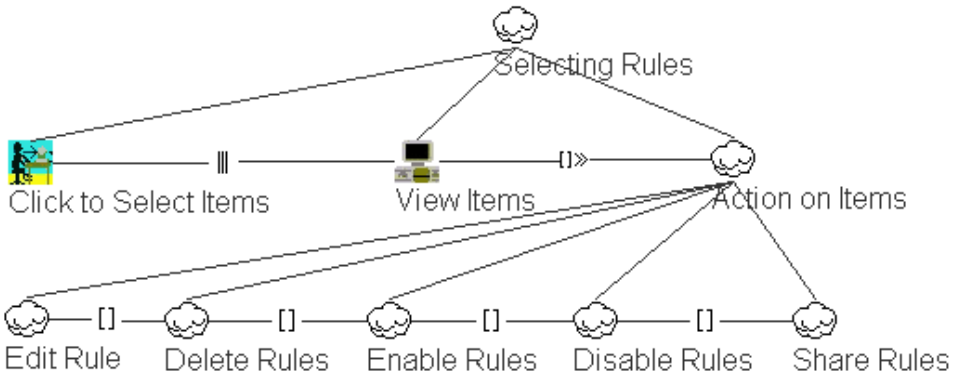

Figure 4.6: Task model for *Selecting Rules*

7. Add Rule

The task model "Add Rule" (Figure 4.7), is the task that allows users to create new rules into the system. The page form contains 3 sections: general rule information, rule conditions and rule consequences.

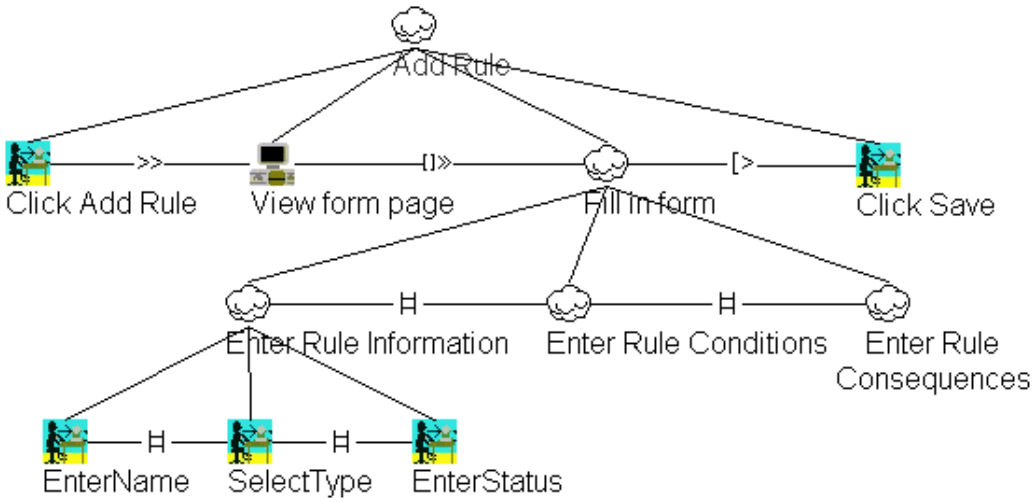

Figure 4.7: Task model for *Add Rule*

8. Enter Rule Condition

The task model "Enter Rule Conditions" (Figure 4.8), is a subtask of the task "Add Rule" in which are specified the fields for creating logical conditions. It permits to add unlimited groups of conditions composed of "OR" and "AND" symbols.

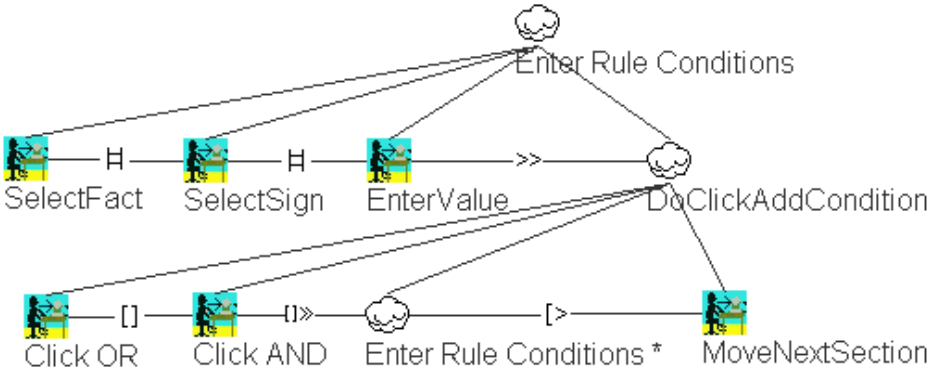

Figure 4.8: Task model for *Enter Rule Conditions*

9. Enter Rule Consequences

The task model "Enter Rule Consequences" (Figure 4.9), is a subtask of the task "Add Rule" in which are specified a list of consequences to trigger in case that the evaluation of rule is *true*.

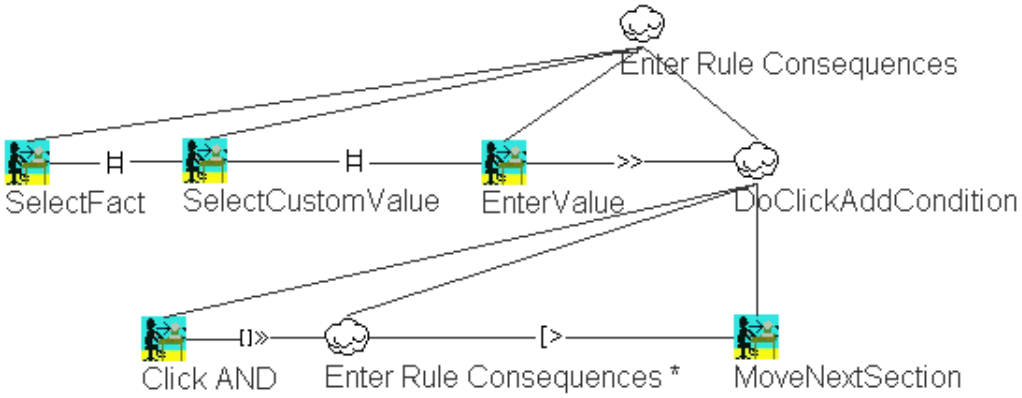

Figure 4.9: Task model for *Enter Rule Consequences*

10. Edit Rule

The task model "Edit Rule" (Figure 4.10), is the task that allows users to edit the rules already created into system. The abstract subtasks "Enter Rule Conditions" and "Enter Rule Consequences" are identical to abstract subtasks from "Add Rule" (Figure 4.7), with the difference that the fields already contain the previously defined value.

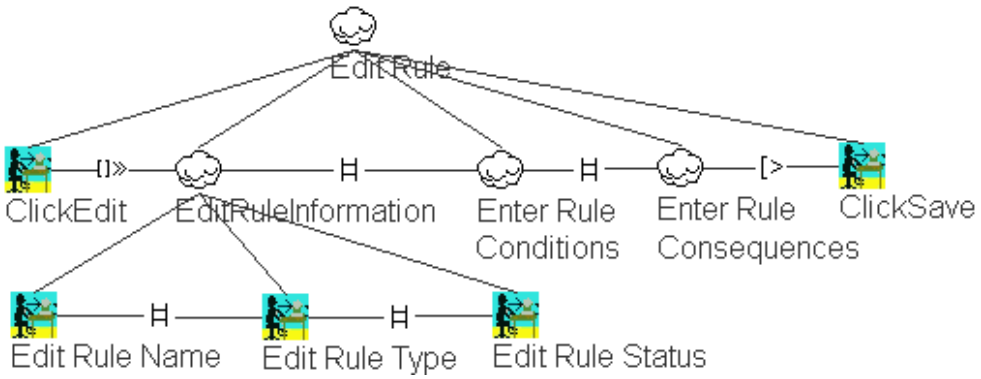

Figure 4.10: Task model for *Edit Rule*

11. Delete Rules

The task model "Delete Rules" (Figure 4.11), is the task that allows users to delete one or multiple rules added previously.

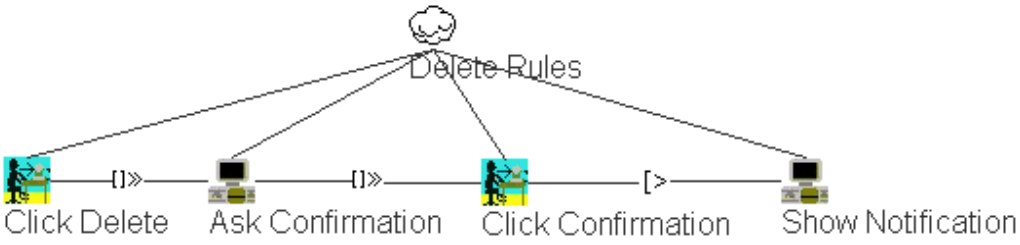

Figure 4.11: Task model for *Delete Rules*

12. Enable/Disable Rules

The task models "Enable Rules" and "Disable Rules" (Figure 4.12), are the tasks that allow users to enable/disable one or multiple rules. The disabled rules are not loaded or considered by the system.

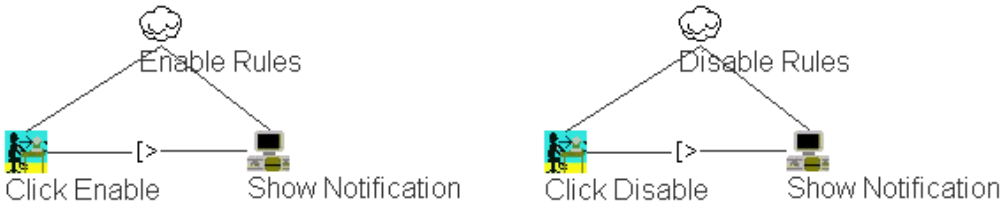

Figure 4.12: Task model for *Enable/Disable Rules*

13. Share Rules

The task model "Share Rules" (Figure 4.13) allows users to share the selected rule. Users can share one or multiple rules simultaneous and has the possibility to specify which sharing method to use it.

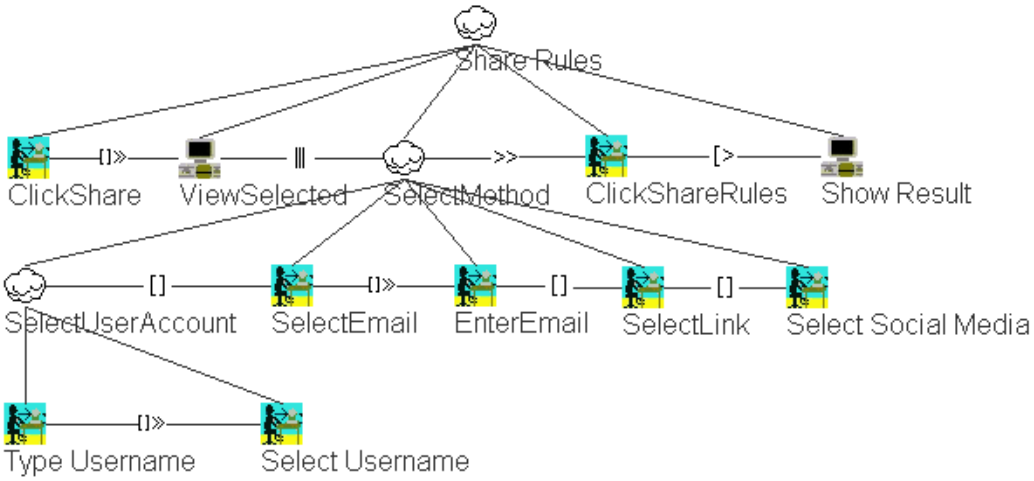

Figure 4.13: Task model for *Share Rules*

14. Receive Rules

The task model "Receive Rules" (Figure 4.14) allows users to review the rules shared by other users. Users can accept or reject the received rules.

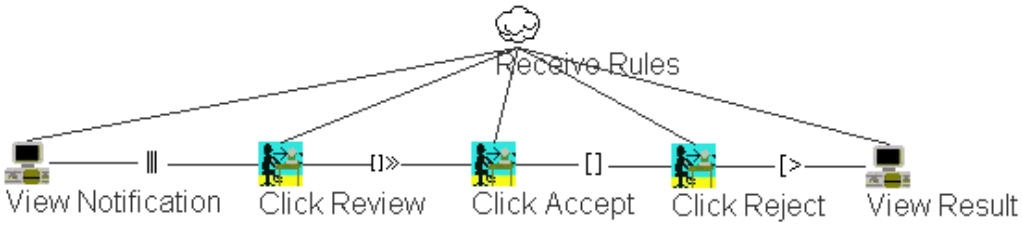

Figure 4.14: Task model for *Receive Rules*

4.1.1.2 Tasks modelling of Teacher Class

The *Teacher Class* contains all user tasks that belong to *Student Class* (Section 4.1.1.1), plus the user tasks defined in this subsection.

1. Manage Facts

"Manage Facts" task model (Figure 4.15) is used to handle the smart devices/web services via the GUI (find, select, add or change lab).

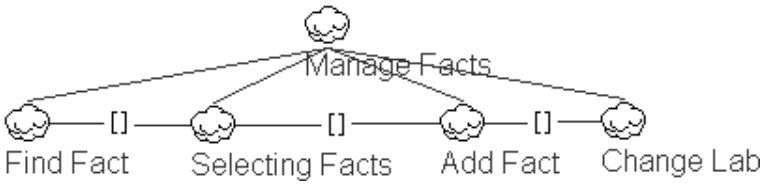

Figure 4.15: Task model for *Manage Facts*

2. Find Fact

The task model "Find Fact" (Figure 4.16) allows the users to find the facts that match the key entered into the "Search" field.

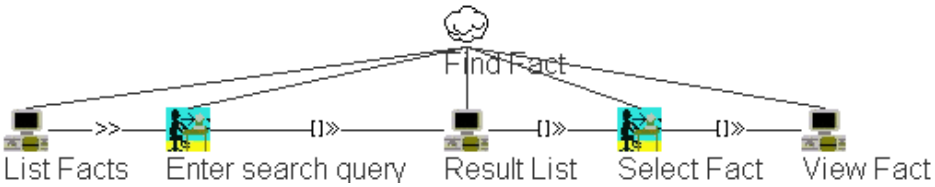

Figure 4.16: Task model for *Find Fact*

3. Selecting Facts

The task model "Selecting Facts" (Figure 4.17), is the task that allows users to select facts for specific operations such as: edit, enable or disable. These abstract subtasks will be modelled later in this chapter.

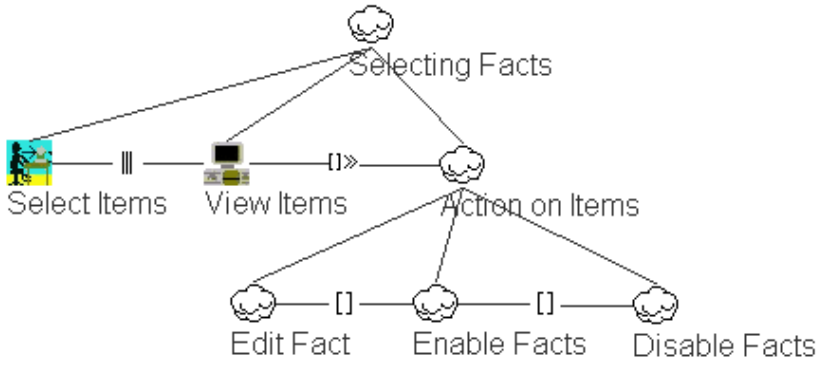

Figure 4.17: Task model for *Selecting Facts*

4. Add Fact

The task model "Add Fact" (Figure 4.18), is the task that allows users to add new facts into the system. The page form is structured in 2 sections: general fact information and fact channels (labs).

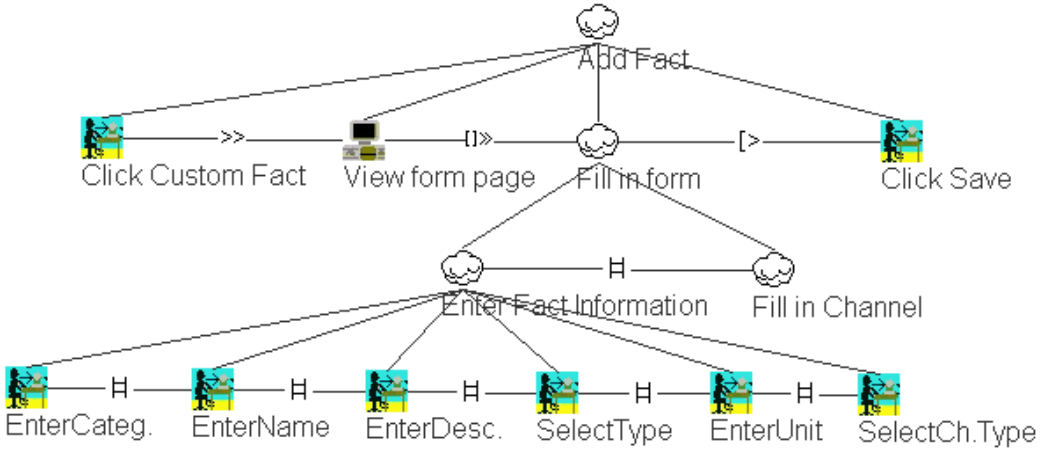

Figure 4.18: Task model for *Add Fact*

5. Fill in Channel

The abstract task "Fill in Channel" (Figure 4.19) is a subtask of "Add Fact" 4.18) that allows users to set for each laboratory (channel) a list of parameters such as: URL source, path, interval and status.

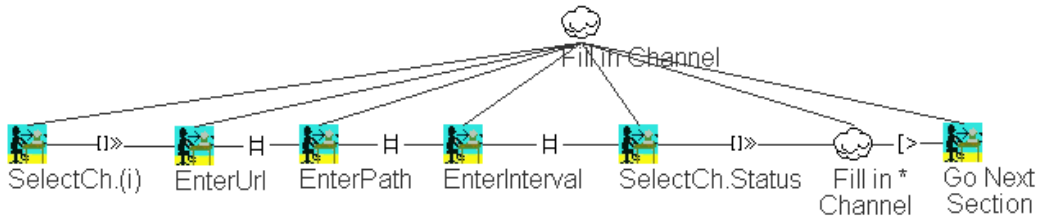

Figure 4.19: Task model for *Fill in Channel*

6. Edit Fact

The task model "Edit Fact" (Figure 4.20) is similar with the task "Add Fact" (Figure 4.18). The only difference is that the form fields are already filled in and the user has the possibility to edit these fields.

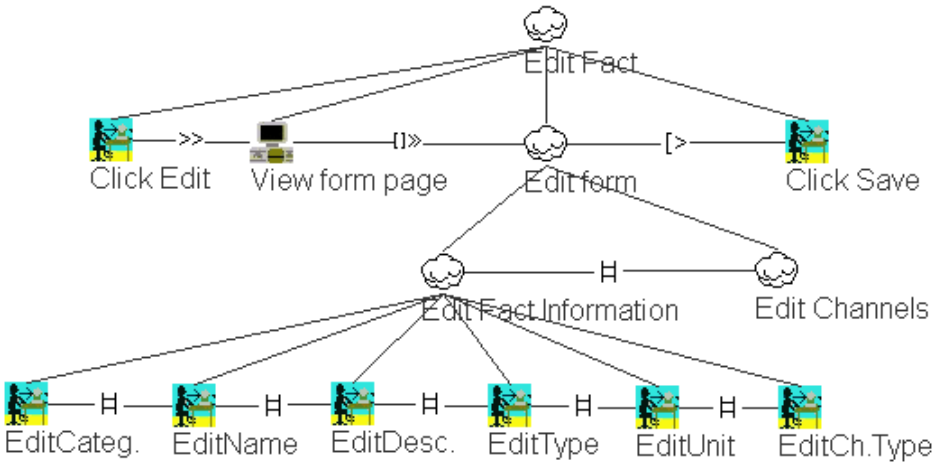

Figure 4.20: Task model for *Edit Facts*

7. Change Lab

The task model "Change Lab" (Figure 4.21) allows users to select a lab from the list that contains all the labs to which the user has access.

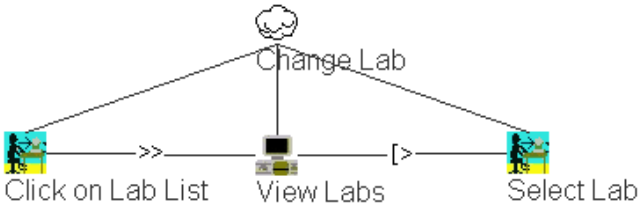

Figure 4.21: Task model for *Change Lab*

8. Edit Channels

The task model "Edit Channels" (Figure 4.22) allows users to edit the parameters of each channels.

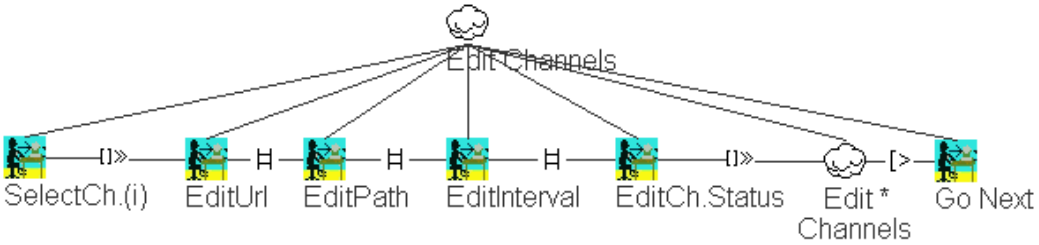

Figure 4.22: Task model for *Edit Channels*

9. Enable/Disable Facts

The task models "Enable Facts" and "Disable Facts" (Figure 4.23), are the tasks that allow users to enable/disable one or multiple facts. The disabled facts are not loaded or considered by the system.

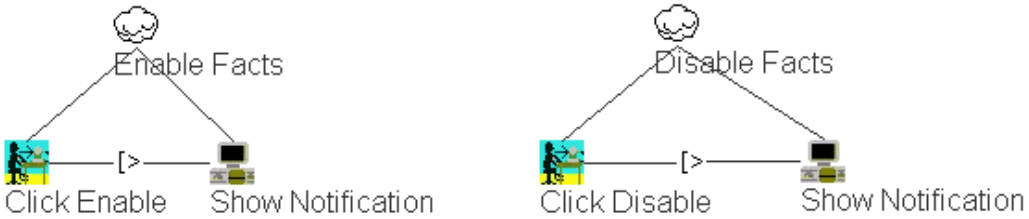

Figure 4.23: Task model for *Enable/Disable Facts*

10. Approve Devices

The task model "Approve Devices" (Figure 4.24) allows users to approve and add new devices that are already discovered through the network.

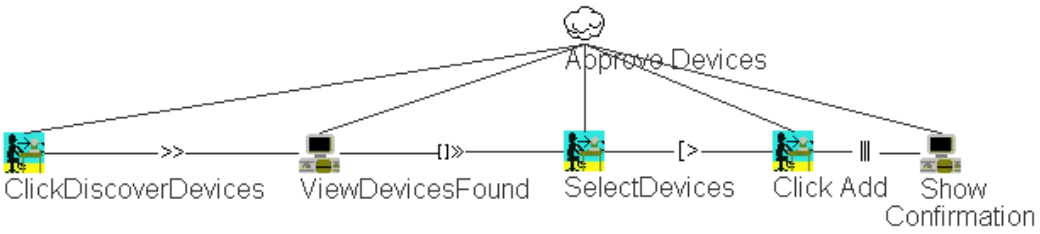

Figure 4.24: Task model for *Approve Devices*

4.1.1.3 Tasks modelling of Administrator Class

The *Administrator Class* contains all the user tasks that belong to *Teacher Class* (Section 4.1.1.2), adding the following additional user tasks:

1. Manage Users

The task model "Manage Users" (Figure 4.25) allows to handle the users of the application. It permits to update the user details and the lab permission.

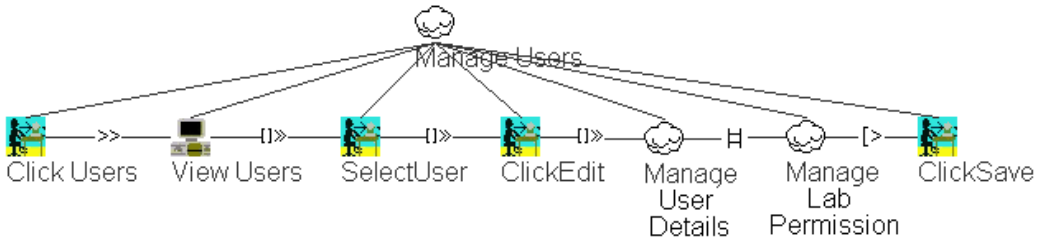

Figure 4.25: Task model for *Manage Users*

2. Manage User Details

As part of the task "Manage Users" (Figure 4.25), the subtask "Manage Users Details" (Figure 4.26) allows to update the user account details.

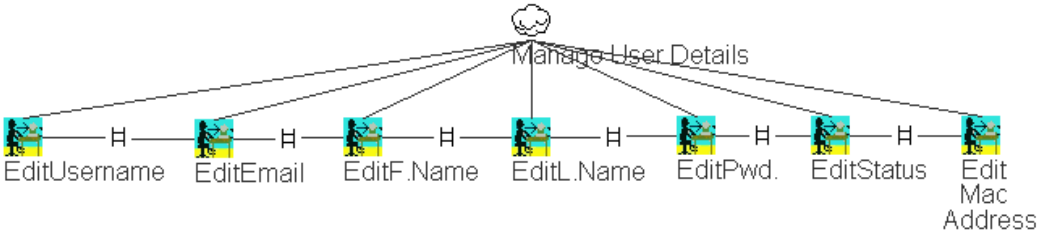

Figure 4.26: Task model for *Manage User Details*

3. Manage Lab Permission

As part of the task "Manage Users" (Figure 4.25), the subtask "Manage Lab Permission" (Figure 4.27) permits to update the lab permission of a selected user.

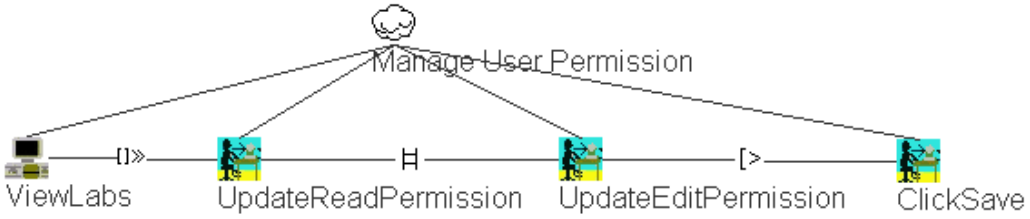

Figure 4.27: Task model for *Manage Lab Permission*

4. Enable Users

The task model "Enable Users" (Figure 4.28), allows to enable the selected user account.

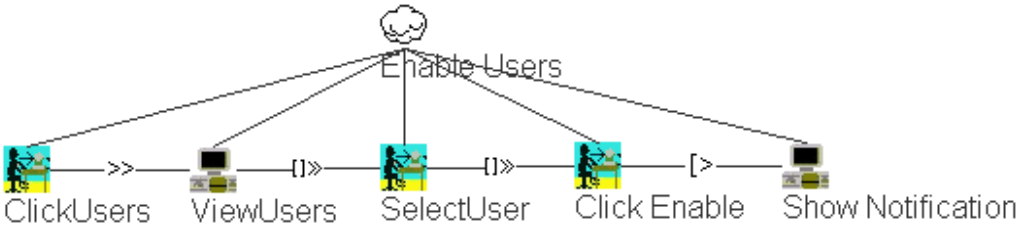

Figure 4.28: Task model for *Enable Users*

5. Disable Users

The task model "Disable Users" (Figure 4.29), allows to disable the selected user account. Users who are disabled can not log in into application.

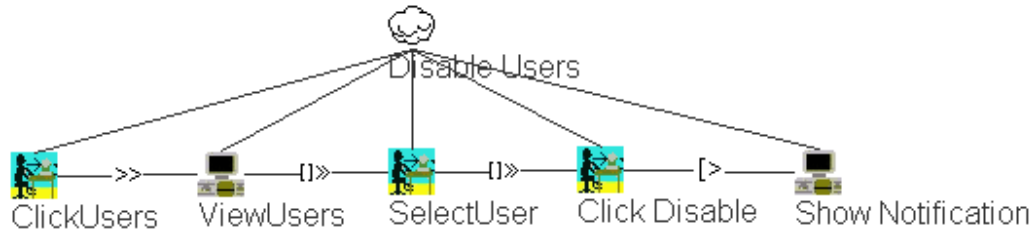

Figure 4.29: Task model for *Disable Users*

6. Delete User

The abstract task "Delete User" (Figure 4.30) permits to delete one or multiple user accounts.

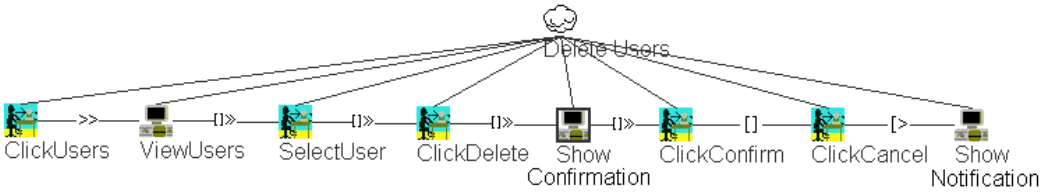

Figure 4.30: Task model for *Delete User*

7. Login as Another User

The task model "Login as Another User" (Figure 4.31) allows the current logged in user to switch to another user without knowing the password.

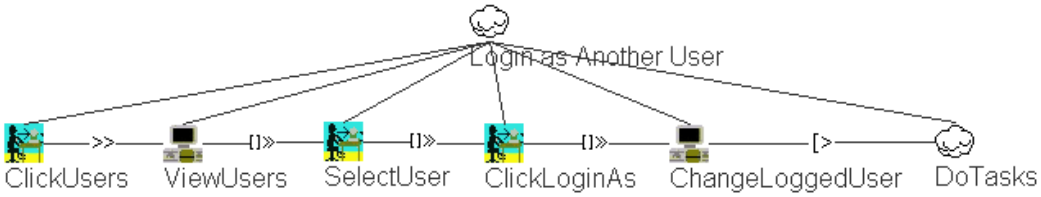

Figure 4.31: Task model for *Login as Another User*

8. Restore to Original User

The abstract task "Restore to Original User" (Figure 4.32) allows users to leave the current account and restore the account of the original user that was logged in.

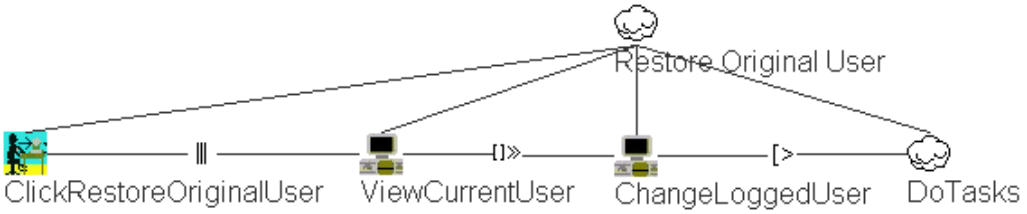

Figure 4.32: Task model for *Restore to Original User*

9. Update Configuration

The task model "Update Configuration" (Figure 4.33), allows users to edit the configuration of the system by specifying the field values for a list of parameters.

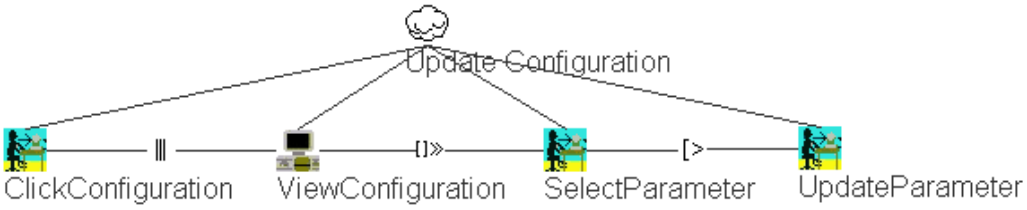

Figure 4.33: Task model for *Update Configuration*

10. Manage Groups Access

The task model "Manage Groups Access" (Figure 4.34), allows to distribute registered users to a specific group of users (*Student*, *Teacher* or *Administrator*). Moreover, it permits to assign the pages permission for every group.

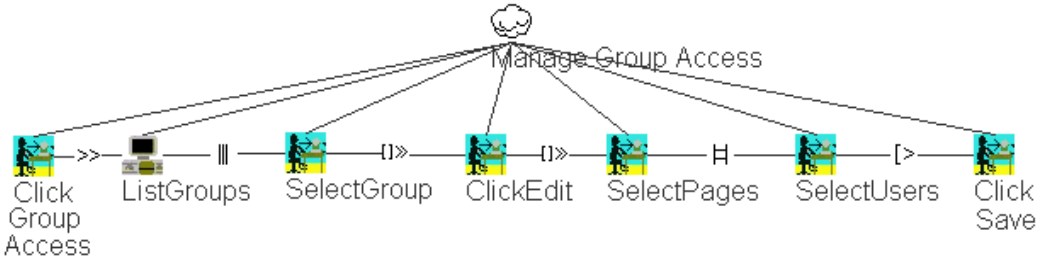

Figure 4.34: Task model for *Manage Groups Access*

11. Do Tasks

The task model "Do Tasks" (Figure 4.35) is the main task of the application that allows the logged in users to do the following activities: update user profile, manage rules, manage facts, approve devices, manage users, login as another user, restore original user and update configuration.

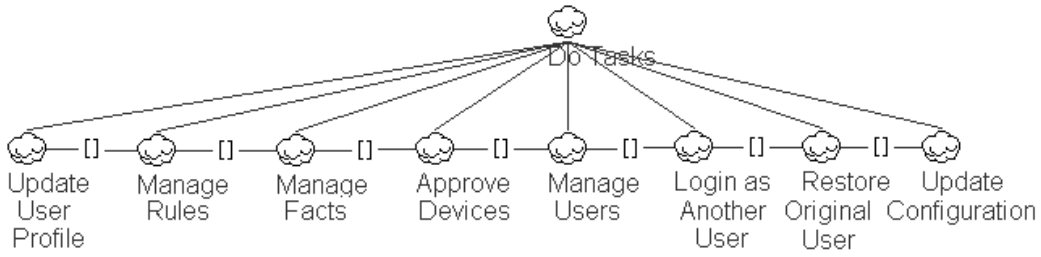

Figure 4.35: Task model for *Do Tasks*

4.2 User Object Models

4.2.1 ORM Modelling

The Object-Relational Mapping (ORM) is a modelling technique for representing in a graphical language the conceptual data requirements of a software system. The ORM conceptual schema diagram consists of a collection of objects connected with roles which include a series of data constraints (mandatory role, uniqueness, exclusion, equality, subset and occurrence frequency).

Figure 4.36 represents the ORM conceptual schema diagram for our framework. The core of the diagram is constituted by the object *User* which is connected with 6 type-entities (*Fact*, *Environment*, *Rule*, *Rule_Share*, *Presentation* and *Group*) and contains 8 type-values *(User ID*, *Username*, *Password*, *First Name*, *Last Name*, *Email Address*, *MAC Address* and *User Enabled*). The *Rule* is connected with *Rule_Share* which indicates that a rule that is shared with other users is part of the *Rule* object.

We can also see that the *Rule* has a certain type-entity called *Rule Type* which contains the symbol "exclusive OR constraint" between *Always* and *With User Presence*. It means that the rule is considered by the system only

when the user is detected or is always considered. In the same way the symbol "exclusive OR" applies to the *Fact Type* object. Another symbol used in the diagram is "external uniqueness constraints", that appears at the objects *Time* and *Date*, and highlights the date to be unique. In the diagram are also shown the "internal uniqueness constraints" (with a line on top of role) and "mandatory role" (represented by a dot beside the object type end or the role end).

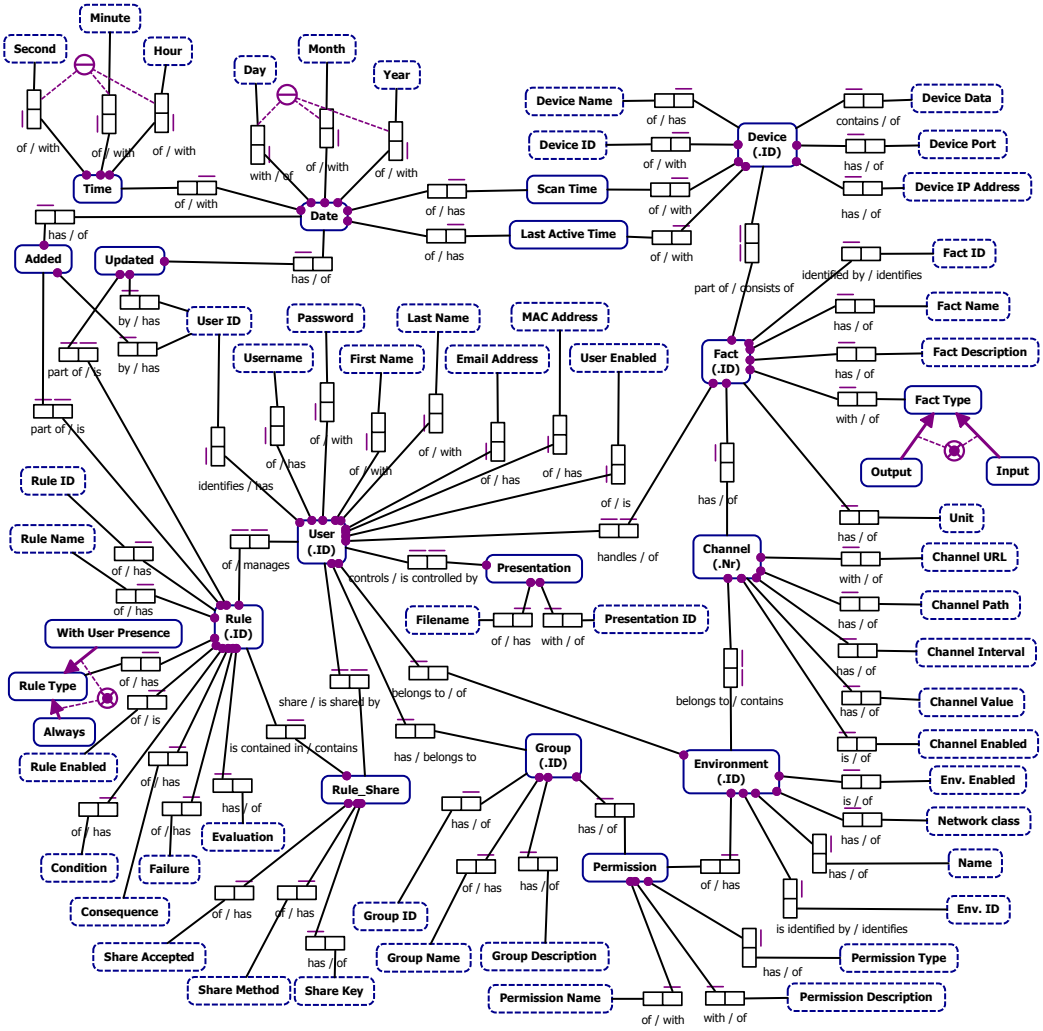

Figure 4.36: The ORM conceptual schema of the BeAware framework

4.2.2 UML Class Diagram

The *Unified Modelling Language* (UML) provides an overall visualisation of the system design. The UML class diagram shows the classes of the system, the relations between the classes, the operations and attributes of every class.

Figure 4.37 shows the UML class diagram of our framework that contains 13 classes highlighted in 4 different colours. The colours show the classes that belong to the same group: blue (classes related to *Rules*), grey (classes related to *Users*), red (classes related to *Facts*) and white (*Others*). In the blue colour are two classes called: *Rule* and *Rule_Share*. We can see that the subclass *Rule_Share* inherits from the superclass *Rule*, which it means that all attributes (superior part of the rectangle) and operations (inferior part of the rectangle) from superclass are available in the subclass.

The arrow between *User* and *Rule* represents the relation of association in which any user can create rules and share the rules with other users. From the superclass *User* inherit two classes: *Student* and *Teacher*. As we can see, the users from the class *Student* are not allowed to add or manage facts. From the superclass *Teacher* inherits the class *Administrator*. Users belonging to class *Administrator* have associations with other classes (shown with arrows) that permit additional functionalities such as: managing the users, managing the system configuration, controlling the group access or handling the permission. In the red colour we can see that the class *Fact* is a superclass for the class *Devices*. The class *Fact* is used for adding and managing the facts, while the *Device* subclass performs operations on the devices found through the network. Each fact contains a number of channels that is equal with the number of environments (laboratories).

As it is shown in the class diagram in Figure 4.37, the classes *Channel* and *Permission* have a dotted line pointing to the class *Environment*. The arrows with dotted lines represent the relationship of dependency to the class *Environment*, that means that both classes use the targeted class. In the context of our ambient system, *Environment* is the class that deals with laboratories. The class *Permission* uses the class *Environment* to control the permission of each laboratory whereas the class *Channel* maps the fact/device values to a corresponding laboratory.

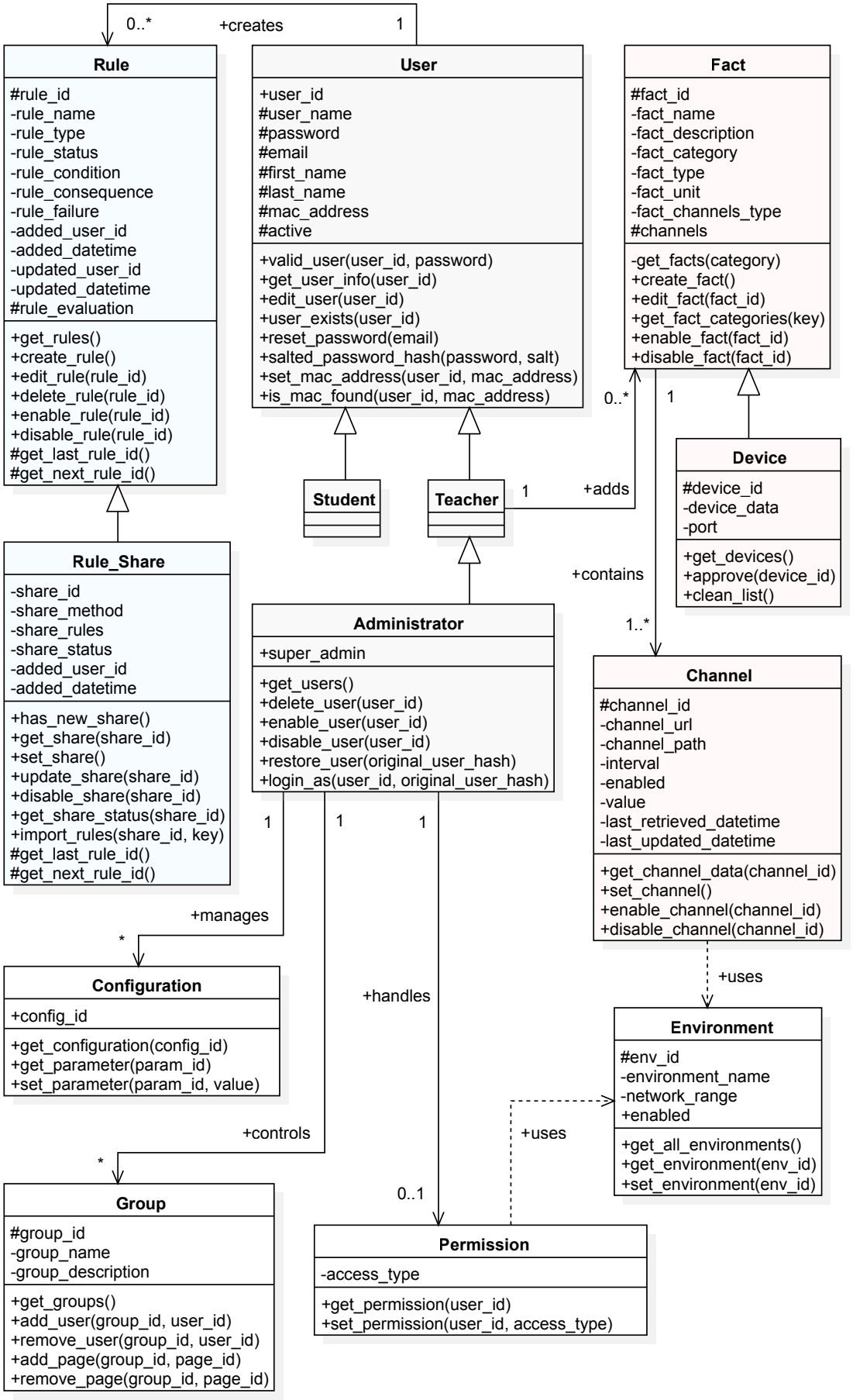

Figure 4.37: The UML class diagram of BeAware framework

5 Implementation

In the previous chapter we have modelled and designed our proposed framework by defining the user tasks and user object models. Based on that, we will show in this chapter how the framework was physically implemented. Firstly, we will introduce the framework architecture and discuss its components. Secondly, we will describe the system core and its main modules. Following that, we will present the database design and continue with the graphical user interface of the framework.

5.1 Architecture

In this section we will describe the architecture of our framework. We start by introducing the framework architecture diagram show in Figure 5.1, that is based on the existing client-server network communication model extended to support inter-server communication as well. In summary, the client-server model is composed of at least one server that serves with information one or multiple clients. The inter-server model is similar to the client-server model, with the difference that the clients become themselves servers and can exchange information with primary server/servers. This results in two types of network communication models: client-server (C2S) or server-server (S2S).

The architecture diagram of our framework contains a set of three components in the upper part: the core of the framework, the user interface and the database. That set of components constitute the primary server in both C2S and S2S network communication models. In the lower part of the architecture diagram a number of connected environments are shown, in our case, a number of academic laboratories. Each laboratory contains an individual Local Area Network (LAN) and communicates with the framework core via a communication interface: RESTfull API or Web Sockets. The LAN is composed of a main network device called *router* that forwards the packets between networks and ensures the connection of our *facts* for ambient purpose.

The *facts* can be classified in two categories: *Devices/Sensors* and *Web Services*). The distinction between both categories is that the electronic *Devices/Sensors* are physical located in the environment and connected to LAN via cable or wireless, while the source content of *Web Services* is situated on separate machines and connected via the World Wide Web (WWW). Moreover, *Devices* and *Sensors* must have their own web server that can serve sensor data or permit to receive data from our framework. By the type of communication flow the framework can have two types: *input* (facts who release content to the framework core) and *output* (facts who permit to be triggered by the framework core). In this way regarding network communication model, each fact can be a *client* when a C2S connection is used, or a *server* when S2S is used.

As shown in the lower part of Figure 5.1, each environment contains an *Ambient Interaction Zone* in which the users interact with the framework. The *Ambient Interaction Zone* covers the whole laboratory or only a specific portion of the laboratory which depends on the integration type and the length of sensors and displays that are used. In our architectural model we consider that each user has a *Smartphone* with an open WiFi network connection to the *Wifi Router*. The *Database* is used to store and retrieve facts, rules, users, configurations and permissions. In our framework we are using two distinct databases: MySQL[1] (relational database used to serve the *User Interface* components) and MongoDB (non-relational database used to store user rules and the ambient information). The *User Interface* provides to users the ability to manage the data in the database by having the possibility to execute CRUD operations such as: create, read, update and delete. The core is the most important part of the framework and consists of following components: *Data Acquisition*, *Rule Engine*, *Event Trigger*, and *Extensions*.

[1]MySQL: https://www.mysql.com

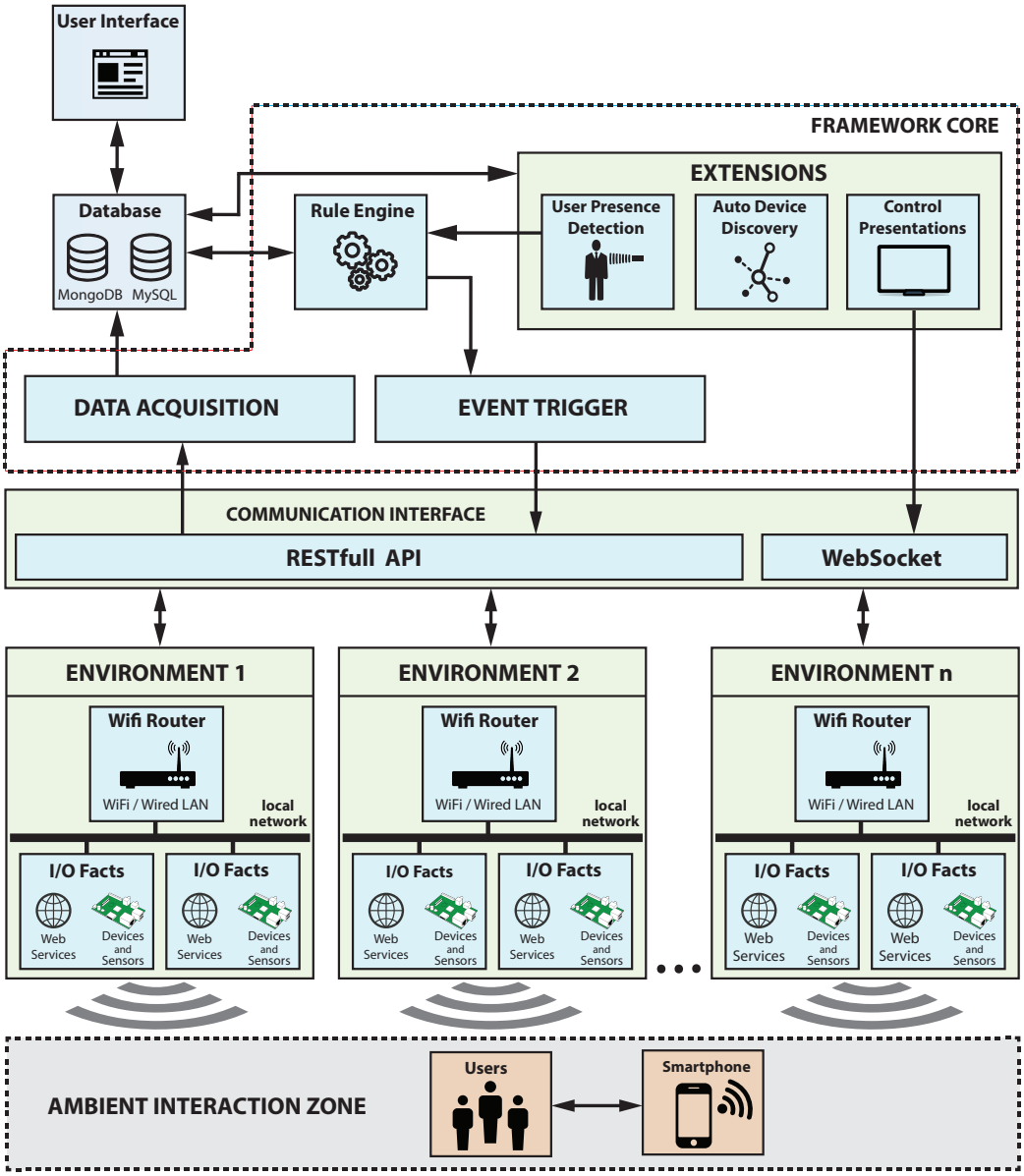

Figure 5.1: Architecture of the BeAware framework

Data Acquisition

The *Data Acquisition* is the component of the framework core that collects information in real-time from *Devices/Sensors* and *Web Services*, and stores it in the database. The information that is offered in JSON format, is fetched by the framework using URL endpoints, thanks to the RESTfull interface over the protocol Hypertext Transfer Protocol (HTTP).

Rule Engine

The *Rule Engine* component loads and processes in real-time the rules that are defined in the *User Interface* and saved in *Database*. Each rule condition called "WHEN" is composed from the value of one or multiple facts. When any of the rule condition is satisfied, the *Rule Engine* sends a signal notification to the component *Event Trigger*.

Event Trigger

The *Event Trigger* component receives the signal sent by the *Rule Engine* and triggers an event to the corresponding fact that is defined in the rule body consequence called "THEN". The "event trigger" means that the component uses the popular HTTP method called "POST", to send via the RESTfull API a payload containing a custom value.

Extensions

The framework is extensible and allows the developers to easily implement new components. For the moment three extensions have been developed:

A) The **User Presence Detection** component has two roles: one is to detect the user presence by identifying if any user is currently inside of a lab, and second to identify the location of any user by knowing in which lab the user is located. One variant of using this extension is the following: when a user is detected as being present in one of the labs, their rules are considered by the *Rule Engine* and enabled only in that specific environment. If the user moves out of the lab their rules will be cancelled.

B) The **Auto Device Discovery** component allows the user to find new devices that are connected to the network and show them in the *User Interface*. Subsequently, the user has the possibility to review and quickly add them to the system.

C) The **Control Presentations** extension has been implemented in order to control the multimedia presentations that can be loaded on the screens or displays. The framework extension communicates via a *WebSocket*[2] interface and outputs multimedia content on the screen of devices that are placed in the ambient environments.

[2]WebSocket: https://goo.gl/reluii

5.2 Database Design

This section presents the database model that was designed for our framework. A database model determines the structure of the database, how the data is organised and how it can be manipulated. The BeAware framework uses two different databases: `MongoDB` and `MySQL`. `MongoDB` is used to store, handle and process in real-time the data of the framework core (facts, rules, devices, environments and configurations). `MySQL` database is used to serve the framework's components of the user interface.

Comparing both databases, `MongoDB` is a non-relational or noSQL database that holds a series of advantages such as: high scalability, dynamic schema, faster data access, improved query language, no complex joins and no object mapping needed. `MongoDB` is a document-based database that stores the data in JSON format. The schema of both databases are quite different but have some common aspects. In the Table 5.1 an analogy between the structure components of both databases is shown. The schema of `MySQL` database is structured in *tables* which contain data stored in *rows* and *columns*. The `MongoDB` database is formed from *collections* of *documents* that contain one or multiple *fields* where the data is located.

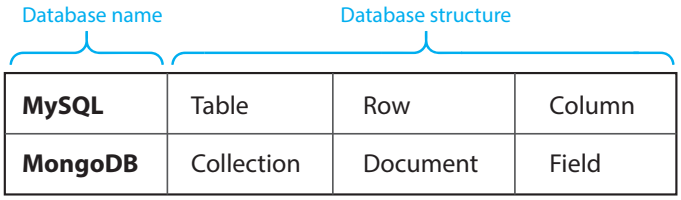

Table 5.1: Analogy between the schema structure of MongoDB and MySQL database

The `MongoDB` database schema of our framework shown in Figure 5.2 contains 14 collections as following: *facts, devices, device_ports, device_ data, rules, rules_ share, rule_ share_ status, rules_ ownerships, environments, data_ acquisition, operators, configuration, presentation_ control and user_ data*. Next, we will describe the application of those collections:

- *facts* is used to store facts defined in the GUI and processed by the core; each document contains a field object-type called *channel*, having the number of channels equal with the number of environments;
- *devices* collection contains documents with devices discovered through the network and provides information about devices such as: name, IP address, MAC address, status or last time when the device was active;

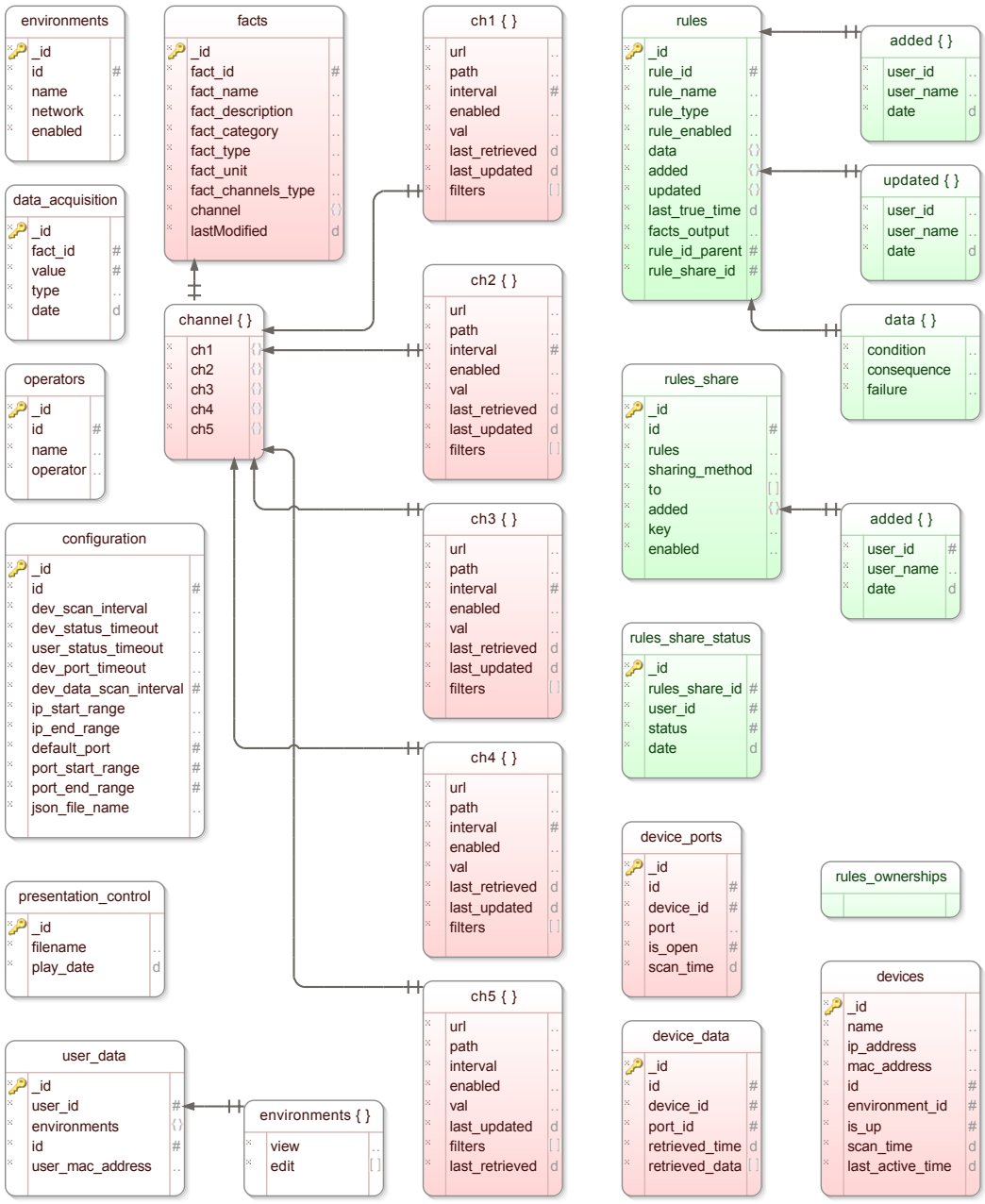

Figure 5.2: MongoDB database schema of BeAware framework

- *device_ports* contains the open ports of devices found in the network;
- *device_ data* includes the configuration of devices found in the network;
- *rules* contains the rules defined by users in GUI; the field *data* includes the rule conditions and the rule effect (*consequence* and *failure*);

- *rules_ share* contains rules that are shared with other users;
- *rule_ share_ status* is a collection used to identify rules that are already approved/rejected;
- *rules_ ownerships* is used to avoid the conflict between rules, by storing information about the rule and assigned fact;
- *environments* includes the list with environments (laboratories);
- *data_ acquisition* collection logs all values that are received by the *Data Acquisition* component or sent by the *Event Trigger* component;
- *operators* collection includes the list with logic operators used in the rule conditions (e.g. equals, greater than, at least);
- *configuration* contains the custom configuration used by the framework;
- *presentation_ control* collection contains the presentation file name that needs to be loaded and played by the system;
- *user_ data* contains the user permission allocated for each environment;

In Figure 5.3 is shown the `MySQL` database schema of our framework, in which the tables that are not relevant to our system were excluded. The schema contains the main table *fuel_ users* and relations to other tables that are used for managing user groups and page permissions in the GUI. The table *fuel_ logs* is used to log all the actions done by users in the GUI and *fuel_ ci_ sessions* is used for handling the user login session.

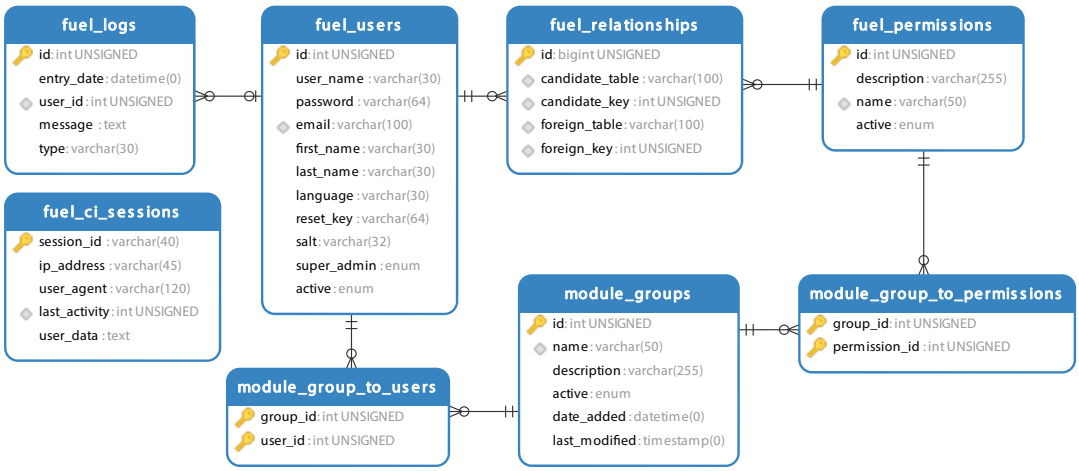

Figure 5.3: MySQL database schema of the BeAware framework

5.3 System Core

5.3.1 Technologies and Programming Languages

We developed our framework from scratch by taking into consideration a main factor: to create an extensible framework that allows future development. In this way we have selected the most recent technologies. We implemented the framework core on top of an asynchronous event-driven architecture called `Node.js` (Figure 5.4). We used noSQL database called `MongoDB` for data storage and `JavaScript`[3] as main programming language, since Node.js is a server-side engine that interprets JavaScript code. We chosen to execute the Node.js engine on a Linux platform and used the operating system called `Ubuntu`[4] that is running in a `VMware Workstation`[5] virtual machine.

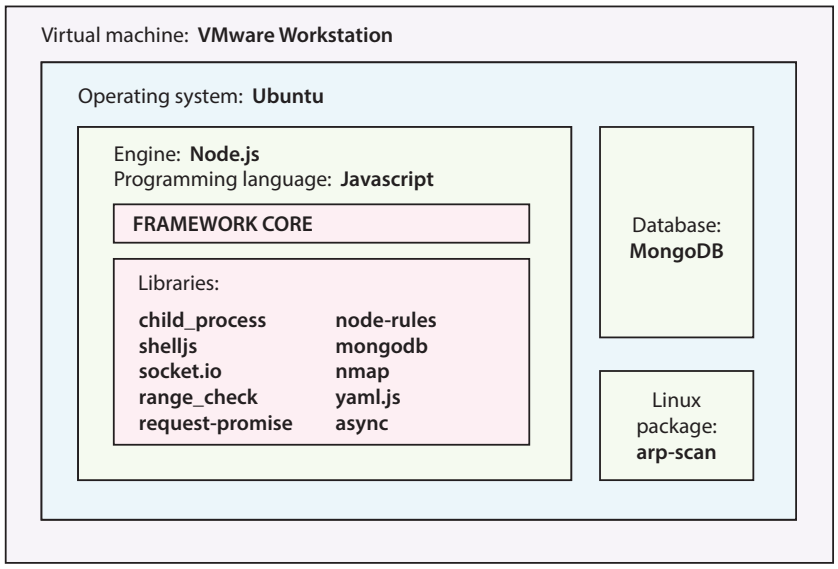

Figure 5.4: The layered diagram of technologies, programming language and libraries used to build the framework core

Some Node.js packages and libraries have been also used to construct the framework core. This includes the followings:

- `child_process`[6] provides the ability to execute multiple Node.js script files by spawning in child processes;

[3] JavaScript: https://developer.mozilla.org/en-US/docs/Web/JavaScript
[4] Ubuntu: http://www.ubuntu.com
[5] VMware Workstation: https://www.vmware.com/be/products/workstation
[6] child_process: https://nodejs.org/api/child_process.html

- `node-rules`[7] is a light weight forward chaining rule engine that was used as basis and adapted to create our *Rule Engine* component;
- `shelljs`[8] permits to run Unix shell commands from Node.js script;
- `mongodb`[9] is the native MongoDB driver that allows to connect and access the MongoDB database from Node.js;
- `socket.io`[10] is an API framework that allows to create real-time application in Node.js using as communication interface Web Sockets;
- `reveal.js`[11] is a framework that allows to create presentation in HTML providing a broad set of tools and configuration;
- `arp-scan`[12] is a linux package that allows to scan the local network and check the list of devices that are connected. It is used in our framework by two extensions (*User Presence Detection* and *Auto Device Discovery*) and runs under the `shelljs` library;
- `nmap`[13] is a network discovery utility that was used in our framework to create the extension *Auto Device Discovery*. `nmap` is used in relation with `arp-scan` and scans the open ports from the network and return them in Node.js script;
- `range_check`[14] is used for two functions: IP address validation and checks if a specific IP address is in a specific class or range of IP addresses;
- `yaml.js`[15] is a parser library that allows to perform conversion between the formats JSON and YAML (YAML Ain't Markup Language);
- `request-promise`[16] is a HTTP client that perform requests over REST-full API and is used in two component of our framework: *Data Acquisition* and *Event Trigger*;
- `async`[17] provides additional function for working with asynchronous JavaScript;

[7] node-rules: https://github.com/mithunsatheesh/node-rules
[8] shelljs: https://github.com/shelljs/shelljs
[9] mongodb: https://github.com/mongodb/node-mongodb-native
[10] socket.io: https://github.com/socketio/socket.io
[11] reveal.js: https://github.com/hakimel/reveal.js
[12] arp-scan: http://linux.die.net/man/1/arp-scan
[13] nmap: https://github.com/nmap/nmap
[14] range_check: https://github.com/keverw/range_check
[15] yaml.js: https://github.com/jeremyfa/yaml.js
[16] request-promise: https://github.com/request/request-promise
[17] async: https://github.com/caolan/async

5.3.2 Description of Main Modules

This section will present the way of how the main modules of the framework core were implemented. As it was shown previously in the framework architecture diagram (Figure 5.1), the framework core is composed of four components: *Data Acquisition*, *Rule Engine*, *Event Trigger*, and *Extensions*. Next, we will review each component and describe the implementation steps of the algorithm.

5.3.2.1 Data Acquisition

The *Data Acquisition* module is created inside of a main `Node.js` function that is executed periodically, in our case we have chosen to execute every 1000 ms. In the first step, the function contains the code lines that extract the list with environments (laboratories) from the `MongoDB` database. Then, the program iterates over environments and extracts from database at each iteration the list with facts that belong to the corresponding environment which are enabled and have defined the database field *fact_ type* as input.

In the next step, the program selects the channel that belongs to the environment of the current iteration and checks the condition if the interval of time defined in the field *interval* has been passed. In case that the condition is satisfied, the program makes a call request to the URL defined in the field *url* of the channel. The call response will be a JSON object containing a collection of information.

In the last step, the program is parsing the JSON object received as response and selects the essential value based on the field *path* defined in the channel. The field *path* determines the path in the JSON tree where the desired value is located. Then, the program saves into database, the value in the field *val* and current time in the fields: *last_ retrieved* and *last_ updated*.

As we have seen in Figure 5.2, each channel *ch** consists of the following fields: *url*, *path*, *interval*, *enabled*, *val*, *last_ retrieved*, *last_ updated* and *filters*. An example of object channel *ch1* with corresponding values is represented in Figure 5.5. The module will fetch at every 3 seconds the url http://www.beaware.tld:8000/api/count_persons.json and based on the path "counter", it extracts the number of people that are present in the corresponding environment, in this case environment 1. The number extracted will be saved in database, in the field *val*.

```
{
  ▾ ch1 : {
        url : http://www.beaware.tld:8000/api/count_persons.json    ,
        path : "counter" ,
        interval : 3,
        enabled : "1",
        val : "0",
        last_retrieved : "ISODate(\"2016-07-10T19:54:26.238+02:00\")"    ,
        last_updated : "ISODate(\"2016-07-10T19:54:26.238+02:00\")"
    }
}
```


Figure 5.5: The structure of object channel stored in the MondoDB database

5.3.2.2 Rule Engine

The starting point for implementing the *Rule Engine* component, was a Node.js library called `node-rules`, which was extended and adapted to the requirements of the BeAware framework. The `node-rules` library is quite limited. It only allows to define a list of rules and list of facts in JSON format, iterates over facts, checks if any of the rule conditions is satisfied and execute a portion of code based on the rule result. Our framework provides the necessary functionality, to define the rules and facts in a graphical user interface (see Section 5.4.2), store them in `MongoDB` database and executes a series of data processing operations which will be further described.

The *Rule Engine* component is called every 2000 ms. In the first step, the program creates 2 arrays: one for the rule and another one for the facts. It retrieves from database the list with rules that are enabled and iterates over them. Each rule will contain all fields that exist in the document called *rules*, as it was shown in the database schema of `MongoDB` in Figure 5.2, including the object field *data* that has three sub-fields: *condition* (contains the JavaScript function that return the result of the rule), *consequence* (JavaScript function that is executed when the rule result is true) and *failure* (JavaScript function that is run when the rule result is false). Two types of rules can be found in the database: rules that are active only when the user is present or rules that are permanently active. The program filters rules that are active at the moment and add them into the first array.

In the second step, after all the rules are added into an array, the program continues to populate the second array with facts. The program queries the database and filters facts that are enabled and have the type "input" ($fact\_type$ = *"in"*). Then, iterates over the facts that belong to an active environment and pushes them into another array.

In the final step, the arrays are sent to a function that processes the data of both arrays. The function iterates over the array with rules and checks the result of each rule condition. The rule condition is composed from the value of one fact or a logic between the values of multiple facts. Based on the rule result, it will execute one of the functions contained in the object field *data* under the sub-field: *consequence* (when result is true) or *failure* (when result is false). An example of a rule that is retrieved from the database, is represented in Figure 5.6. The condition of the rule contains a function that has an input parameter "R" and the following code in the body:
R.when((this && this.fact_id == '2') && (((R.fact(2).val >= '1'))));


```
{
    _id: "ObjectId(\"5718c979639a94193e5d83d2\")"    ,
    rule_id : 8,
    rule_name : "Turn TV on when at least 1 person in the lab"    ,
    rule_type : "1",
    rule_enabled : "1",
  ▾ data : {
        condition : "function(R) {R.when((this && this.fact_id == '2') &&
        (((R.fact(2).val >= '1'))));}"   ,
        consequence : "function(R) {if(this.fact_id ==
        '2'){custom.trigger.rule_set_consequence('1', R.fact('2').channel_id,
        '1', R.info);} R.next();}"   ,
        failure : "function(R) {if(this.fact_id ==
        '2'){custom.trigger.rule_set_failure('1', R.fact('2').channel_id, '-1',
        R.info);} }"
    },
  ▾ added : {
        user_id : "1",
        user_name : "admin" ,
        date : "ISODate(\"2016-04-21T14:37:13.000+02:00\")"
    },
  ▾ updated : {
        user_id : "1",
        user_name : "admin" ,
        date : "ISODate(\"2016-06-08T03:43:52.000+02:00\")"
    },
    last_true_time : "ISODate(\"2016-06-08T03:47:46.737+02:00\")"    ,
  ▾ facts_output : [
        "1"
    ]
}
```


Figure 5.6: The structure of a rule stored in the MondoDB database

The "R" parameter contains an object with all of the facts, while "this" is an object that references to the fact of current iteration. In the body of the function it will be checked if *this* is defined, *fact_ id* is the fact identifier of the current iteration and the value of fact with id 2 is greater or equal than 1.

5.3.2.3 Event Trigger

The *Event Trigger* component works in correlation with the *Rule Engine* component. As it was already mentioned, according to the result of the rule's condition, the function from *consequence* field or *failure* field will be executed. The body of both functions, contains a check to determine if the fact belongs to the current iteration and a call to an external function *custom.trigger.rule_set_consequence()*, respectively *custom.trigger.rule_set_failure()*. Both external functions have four input parameters:

1) fact_id (the identifier of the fact that will be triggered)
2) channel_id (environment id where the fact is located)
3) a custom value that is sent
4) an object with information about the rule

A series of operations will take place, inside the external functions. Firstly, the program retrieves from the database the fact identified by *fact_id* and checks if the fact is already used by another rule. Once a rule takes over the control of a fact, the other rules will be blocked to take control of the same fact, until the rule that has the control is not active anymore. This is a conflict resolution operation that will avoid a fact to oscillate between 2 values. Secondly, the program selects the channel based on the input parameter *channel_id* of the functions and it checks two conditions: the current trigger value is different than the value sent previously and elapsed time since last trigger is higher than the trigger time defined in the configuration. Finally, if both conditions are satisfied, the program executes a POST request using the following information:

1) the parameter *url* from the selected channel will be used as target address of the POST request
2) the parameter *path* from the selected channel will represent the name of a custom variable that will be send in the POST request
3) a payload value that represents the value of the custom variable which will be send in the POST request

Moreover, the field *val* (that belongs to the selected channel), will be updated in `MongoDB` database with the payload value together with current timestamp.

5.3.2.4 Extensions

A) **User Presence Detection**

The *User Presence Detection* extension is based on the actual computer network configuration and uses the smartphones of the users as device to detect their location and presence. Each smartphone contains a MAC

address that uniquely identifies the device. When a smartphone connects to the WiFi router, the router assigns an IP address to that device. As it was shown in the framework architecture (Figure 5.1), each environment contains an individual WiFi router that includes a customisable IP network range. Some examples of network ranges are the followings: 10.0.0.0 - 10.0.0.255, 10.0.1.0 - 10.0.1.255 and 10.0.2.0 - 10.0.2.255. To make this extension work properly, each router should have a different range of IP addresses.

Therefore, each environment is assigned an IP range. The list with IP ranges are saved into the field *network* in the collection called *environments* from `MongoDB` database. The MAC address of the smartphone is saved into the field *user_mac_address* in the collection called *user_data* from `MongoDB` database. By having these information, the program scans the networks using the `arp-scan` library and checks if any device is connected to the network. In case that any of the MAC address that belong to a user is found through the network, the program determines the IP address of that device and checks in which range of IP addresses it belongs. In this way, it can detect the presence of a user, it can determine when a user was last time active and it can determine a user location in function of environment (laboratory).

B) **Auto Device Discovery**
The *Auto Device Discovery* extension works on a similar principle like *User Presence Detection* extension. It uses the actual LAN computer network and allows to find devices through the network with a minimum configuration. In order to connect a new device and make it available in our framework, it is necessary to only provide an YAML file in which are defined few variables, as they are shown in Figure 5.7.

For the first three variables (*fact_name*, *fact_description* and *fact_category*) a freely chosen name is required to be defined that allows the users to identify the device in the GUI. The fourth variable *fact_type* represents the type of information flow and can have one of these values: “in” or “out”. The value “in” is used when the framework retrieves data from device, and value “out” when the framework pushes data to device. The next variable *fact_unit* needs to be filled in with the measure unit of the device. In our example the “p.” represents the number of the people. The last three variables under the *channel* group, depend on the *fact_type* and represent the followings: *url* is the end-

point used for data acquisition (when *fact_ type*: "in") or data trigger (when *fact_ type*: "out"), *path* is the variable name of the JSON tree that selects the place where the relevant value is located (when *fact_ type*: "in") and the name of the variable that will be send in the POST using the RESTfull interface (when *fact_ type*: "out"), *interval* is the period of time to fetch again the value from device (when *fact_ type*: "in") and the period of time to retain a rule in the "true" state after the rule condition becomes "false" (when *fact_ type*: "out").

```
---
  -
    # The custom name of the device
    fact_name: "Counter people"
    # The custom description of the device
    fact_description: "Total people in the lab"
    # Device category
    fact_category: "Kinect"
    # Type: "in"   = framework receives data from device
    #       "out"  = framework sends data to device
    fact_type: "in"
    # The custom unit of the measured value
    fact_unit: "p."
    channel:
      # URL/Endpoint (from/to where to retrieve/send the data),
      url: "http://www.beaware.tld:8000/count_persons.json"
      # path (in = json path / out = variable)
      path: "counter"
      # Interval to refetch the data (fact_type = "in")
      # Interval to retain the rule in true state (fact_type = "out")
      interval: 5 # seconds
```

Figure 5.7: The configuration of a YAML file for making the device discoverable

Next, it will be described how the program works. Firstly, the program scans the computer network with `arp-scan` library, using the ranges of IP addresses defined in the collection *environments* from `MongoDB` database (see Figure 5.2). The IP address of all devices that are detected online, are saved into a database collection called *devices*, together with the following fields: MAC address, *environment_ id* (the identifier of the environment where the device is connected), *is_ up* (flag that represents the status of the device: "1" - connected, "0" - disconnected), *scan_ time* (the timestamp when the device was scanned last time) and *last_ active_ time* (the timestamp when the device was detected online last time).

Secondly, the program uses the `nmap` library to scan if any of IP addresses has open ports and if any are detected it will be saved in the database into *device_ ports* collection. Then, the program executes cURL requests for each port that is found open and checks if the YAML file exists. In case that the file is found, the program retrieves the YAML file, converts in to the JSON format using the `yaml.js` library and stores it in the database into collection *device_ data*. This way, the device will auto-appear in the GUI and the user can approve it or reject it. A device that is approved, appears in the list with facts and the user can define rules on it.

C) **Control Presentations**

The *Control Presentations* is an extension that uses the publish-subscribe architecture model to control multimedia presentations that are created with `reveal.js` framework and shown on digital displays (TV or projection screens). The extension provides API access to set the presentation file name that is desired to be loaded on the screens. For instance by executing a POST request to the following URL address http://www.beaware.tld:8000/ambient/api/set_presentation including a parameter *filename=index1.html*, it will load on the screen the `reveal.js` presentation with the filename "index1.html".

Firstly, the file name that is sent via an API interface, is inserted together with a timestamp into the *presentation_ control* collection from `MongoDB` database. Then, the program opens a *WebSocket* with `socket.io` library and tracks all changes that appear in the *presentation_ control* collection. When a new file name is inserted into the database, the program collects the message and publishes it to subscribers through *WebSocket*. The subscribers who receives the message, orders the `reveal.js` framework to change the current presentation with the file name that is received.

This extension has been developed especially to allow our framework to control multimedia presentations. Mainly, the users can define facts relying on this API and then, create rules in the system which depend and use these facts. In this way the framework will execute POST requests to the API, and therefore, control the presentations that are displayed on the screens.

5.4 Graphical User Interface

In this chapter we will further discuss the implementation of framework's user interface. It starts by describing the methods, technologies and programming languages that have been used. Then, it continues with presenting the components of the user interface for some of framework's main pages.

5.4.1 Technologies and Programming Languages

The user interface of the framework is built on top of a Model-View-Controller (MVC) framework called `FuelCMS`[18]. The MVC is an architectural pattern allowing to create user interfaces by splitting the software in three parts (model, view, controller). `FuelCMS` is a CMS-based framework that is developed in a lightweight and rapid development web framework called `CodeIgniter` and programmed in a server-side scripting language called `PHP`. The PHP code is executed on a HTTP server called `Apache`[19] running on the Linux platform `Ubuntu`.

`FuelCMS` operates with both databases `MySQL` and `MongoDB`, databases that were already discussed in Section 5.2. The driver for `MySQL` is already included in the source of `FuelCMS`, while the `MongoDB` driver was developed in order to add and update the information concerning facts, rules, devices, environments and configurations.

In addition to `FuelCMS` framework, the following programming and markup languages have been incorporated in our framework: `Javascript` and `jQuery` were used to create dynamic interaction between user and graphical interface, `HTML` was used to describe the web documents, `CSS` used to describe the visual style of the web documents and `AJAX` that allows to asynchronous exchange data with the server. Beside those, the following libraries have also been used: `jQuery UI`, `Semantic UI`[20] and `Datatables`[21].

5.4.2 The Main Webpages of the Framework

We have seen that based on the system requirements defined earlier in Section 3.3, have been resulted the user task models presented into Section 4.1.1. Those user tasks, helped us to deeply analyse the system,

[18] FuelCMS: http://www.getfuelcms.com
[19] Apache: https://httpd.apache.org
[20] Semantic UI: http://semantic-ui.com
[21] Datatables: https://datatables.net

design and develop the framework's user interface. In this subsection we present some of the main pages of our framework's user interface and discuss its components in correlation with the CTT user task models.

5.4.2.1 Add Facts

The Figure 5.8 shows the user interface of the page that allows users to add new facts to the database. The page is implemented based on the task model called "Add Fact" (task 4. from Section 4.1.1.2) and includes two main sections: *Fact Information* and *Channels*.

The section *Fact Information* contains the following fields:

- *Fact belongs to* represents the category in which the fact will belong to. A list of existing categories will appear when the user starts to type in, but also the user has the possibility to fill in a new category.
- *Fact Name* is a custom text used to identify the fact.
- *Description* a small description of the fact.
- *Fact Type* is a selection field that represents the type of communication flow: "Input" when the sensor data is received by the framework or "Output" when the framework sends data to the fact.
- *Fact Unit* is the measure unit of the fact (eg. °C, %).
- *Chanel Type* is a selection field that can have the value: "Separate" or "Common". When the option "Separate" is selected the fact will have a separate channel for each environment. The option "Common" means that the channels of all environments will be merged in one channel.

The section *Channels* is used to fill in fields which belong to a specific channel, in our case each channel constitute an individual environment/laboratory. The section *Channels* is divided in two parts: "Current channel" (selected lab of the user) and "Other channels" (all the rest available laboratories). The fields of every channel is changed according to *Fact Type*.

If the field *Fact Type* will be selected as "Input", the channels will be populated with the following fields:

- *URL* represents the URL of the JSON resource from where the framework receive sensor's data.
- *Path* is the variable key in JSON resource where the data is located.

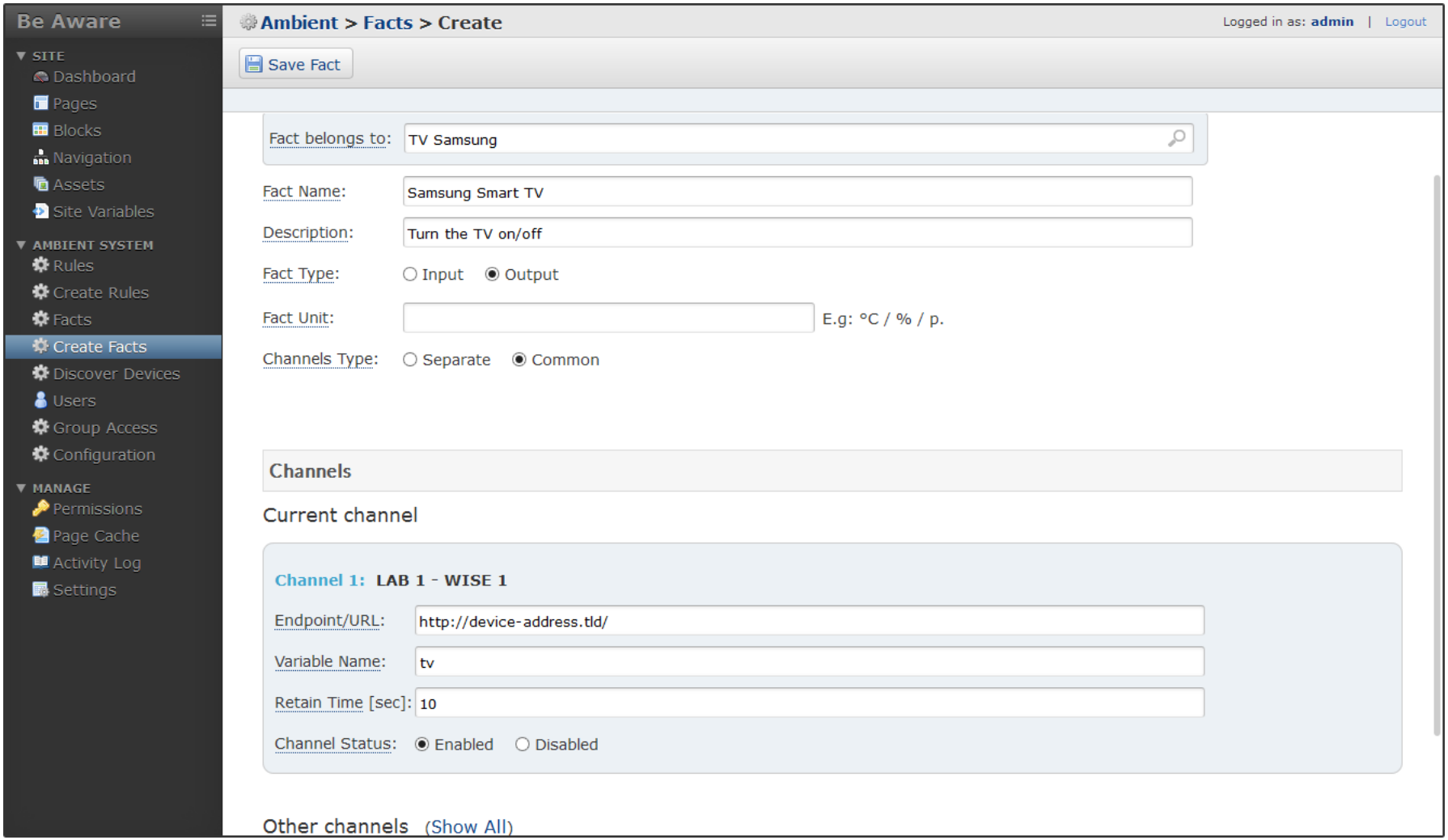

Figure 5.8: The user interface of the page that allows to add new facts

- *Interval* represents period of time in seconds for fetching again of data
- *Channel Status* is the channel status and can be: "Enabled" or "Disabled"

In case that the field *Fact Type* will be selected as "Output", the channels will be populated with the following fields:

- *Endpoint URL* represents the RESTfull resource that will be triggered by the framework.
- *Variable Name* is a custom variable name that will be posted to the resource, containing the value defined later in the rule's consequence.
- *Retain Time* represents the time in seconds to retain the actual value, even if the rule condition is not satisfied anymore.
- *Channel Status* is the channel status and can be: "Enabled" or "Disabled"

After filling in the fields and clicking on the button "Save Fact", the fact will be inserted into `MongoDB` database and listed on the page "List Facts", page that it will be discussed further.

5.4.2.2 List Facts

The Figure 5.9 shows the user interface of the page that contains the facts added by users. The page is implemented based on the abstract task called "Manage Facts" (task 1. from Section 4.1.1.2). The list with facts is retrieved from the database, rendered using the library `Datatables` and shown into a table.

A `MongoDB` driver has been especially developed for our framework, in order to access the database in `FuelCMS` framework. Moreover, a controller to provide the needed data in a predefined structure for `Datatables` was also implemented. The reason for developing this driver and controller, is that the stack of technologies *MongoDB-FuelCMS-Datatables* allows to perform quickly a series of operation (such as: sorting by columns, incremental search and dynamic pagination), only by loading a small amount of data from database with AJAX.

The table shown in the page from Figure 5.9, include some of the fields described earlier in the previous section "Add Facts" and some new fields. Further, we will describe the columns where appears new fields or symbols:

- *ID* a unique identifier of the fact.

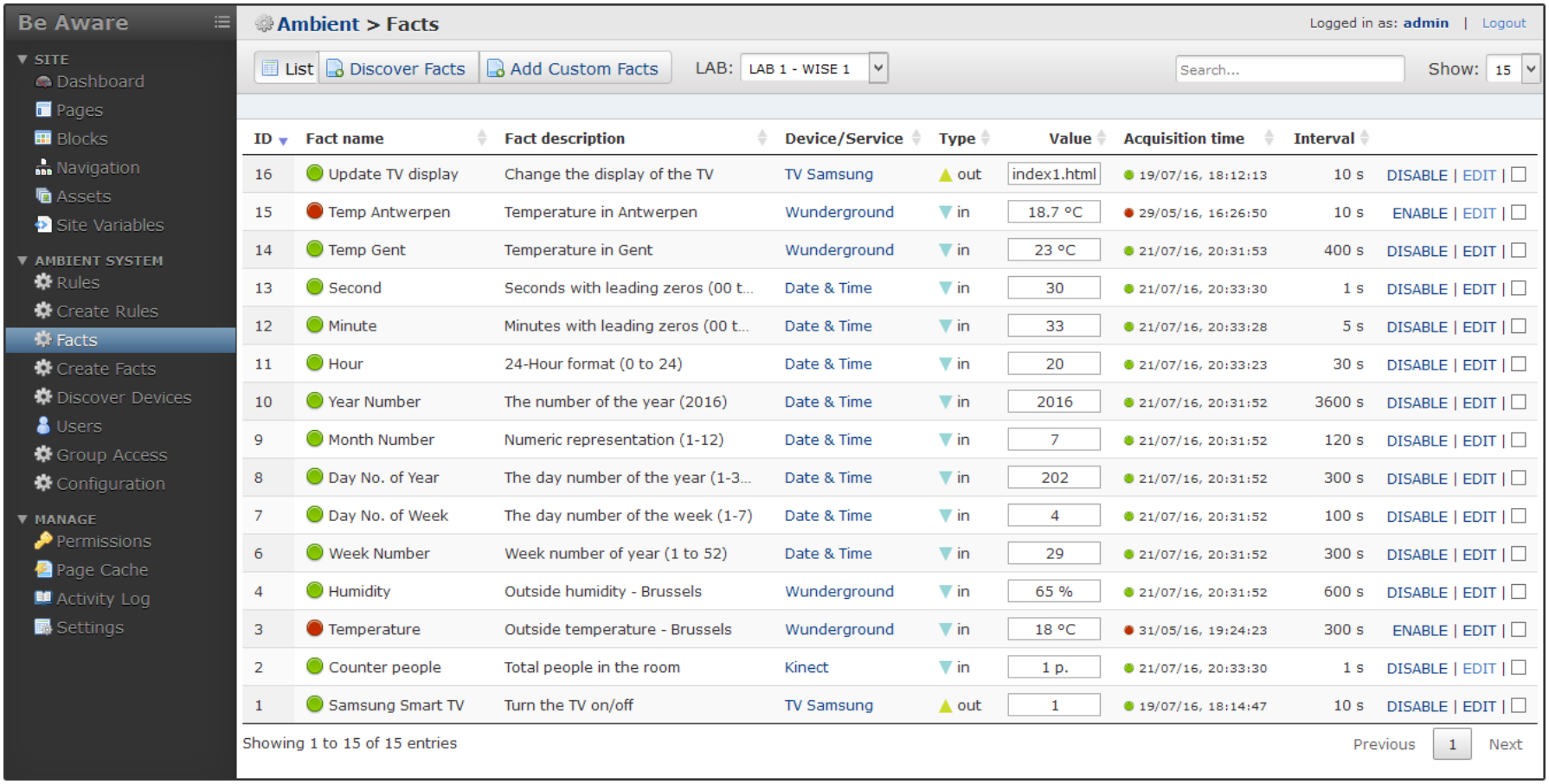

Figure 5.9: The user interface of the page that lists the facts belonging to LAB 1

- *Value* is the value sent/received by the framework.
- *Time* is the time when the value was sent/received last time.

In the upper part of the table are two select fields, one called "LAB" and another one called "Show". The field "LAB", allows to choose the channel or the environment (laboratory) for which to be listed the table with facts. The field "Show" permits to select the total number of results that are visible on the page. In addition, the results can be searched and sorted by all columns.

5.4.2.3 Create Rules

The Figure 5.10 shows the user interface of the page that allows the users to create rules relying on the facts previously added into database. The page is implemented according to the task called "Add Rule" (task 7. from Section 4.1.1.1) and includes three main sections: *Rule Information*, *Specifying Rule Condition (WHEN)* and *Specifying Rule Consequence (THEN).*

The section *Rule Information* contains the following fields:

- *Rule Name* is a custom text used to identify the rule.
- *When to treat this rule* is a selection field that allows to choose: "Always" or "Only when I am inside of a Smart Lab". If the option "Always" is chosen, the system will always consider the rule, while "Only when I am inside of a Smart Lab" is the option that in which the system will consider the rule only when the user who created the rule is present in the sensing environment.
- *Status* is the status of the rule that can be: "Enabled" or "Disabled"

The section *Specifying Rule Condition (WHEN)* permits to create a logic condition composed of multiple facts. A condition consists of a fact name (first input field), a logic operator (second field) and a value (third field). The first input field offers the possibility to type in the name of the fact and then select from the list with proposed facts of type "Input". The second field contains the following logic opperators: *Equals*, *Not Equals*, *Greater than*, *Less than*, *At least* and *At most.* A rule condition containing "n-facts" can be constructed using the operators "AND" and "OR", which are accessible by clicking on "Add OR Condition", respectively on "Add AND Condition". In Figure 5.10 is defined a rule condition consisting of a single fact: "Counter people is at least 1".

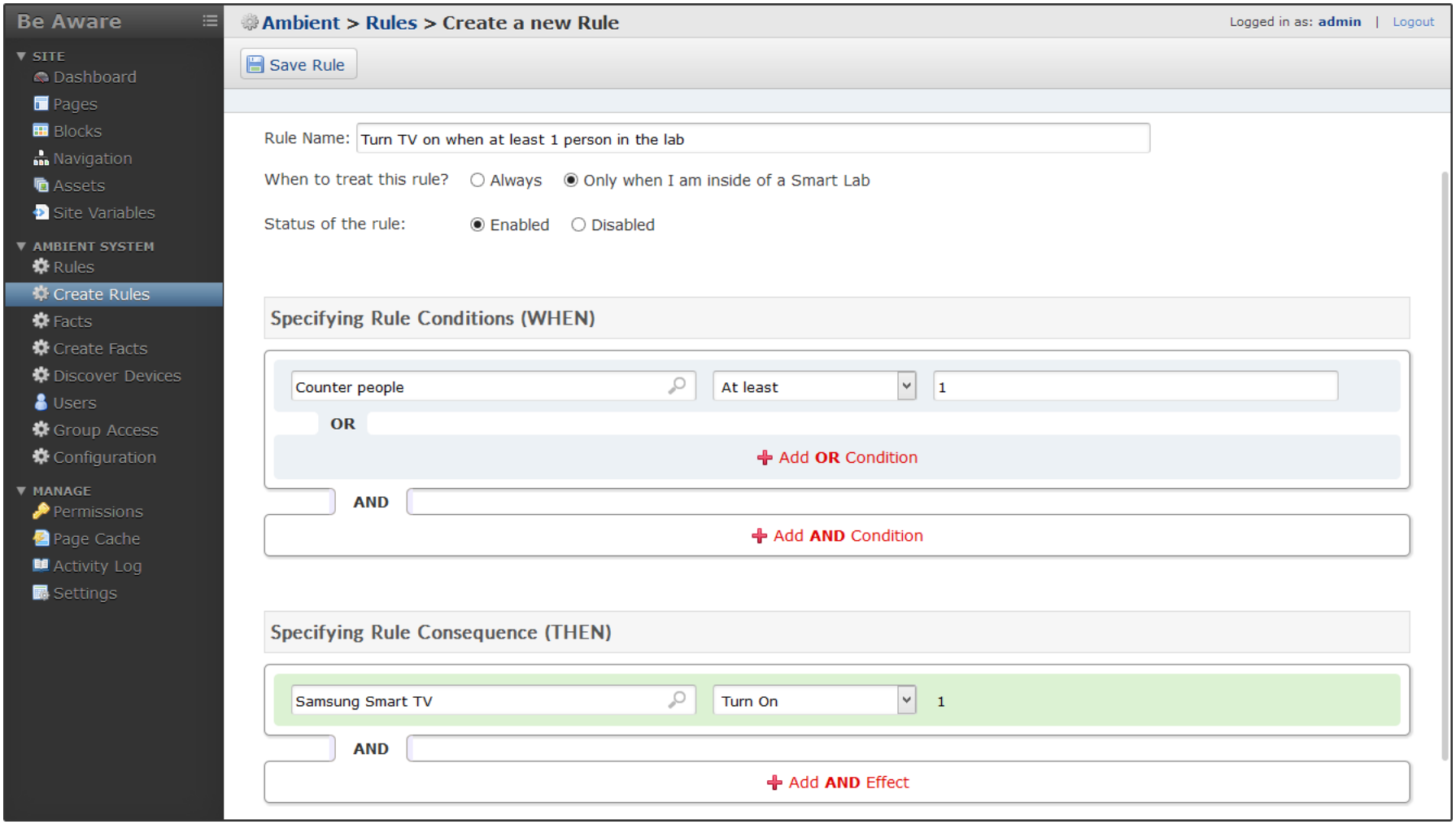

Figure 5.10: The user interface of the page that allows to create new rules

The section *Specifying Rule Consequence (THEN)* represents the effect of the rule, executed when the rule condition is satisfied. The first field constitute the fact name, in which the user starts typing in and selects from the list with proposed facts of type "Output". The second is a selection field having the following options: "Turn On", "Turn Off" and "Send a custom value". If the third option is selected, an input text field appears which allows the user to fill in a custom value. A rule consequence containing "n-facts" can be constructed, using the operator "AND", that is accessible by clicking on "Add AND Effect". Once the rule condition is satisfied, the framework's core will trigger the facts with the assigned values defined in the rule consequence. The value "-1" is used, when the rule condition is not satisfied.

After filling in the fields and clicking on the button "Save Rule", the rule will be inserted into `MongoDB` database and listed on the page "List Rules", page that it will be discussed further.

5.4.2.4 List Rules

The Figure 5.11 shows the user interface of the page that contains the rules added by users. The page is implemented based on the abstract task called "Manage Rules" (task 4. from Section 4.1.1.1). The list with rules is shown into a table, in the same way as it was described earlier in this section for "List Facts".

The table contains the following columns:

- *ID* is a unique identifier of the rule.
- *Rule name* contains the name of the rule and beside it has a bullet with the following colour significance: "green" (rule enabled and active), "orange" (rule enabled and not active), "red" (rule disabled and not active). In the second case (rule enabled and not active), the rule is not loaded by system because it depends by the user presence and the user is absent.
- *User presence* shows if the rule depends by the user presence ("Yes") or is always active ("No").
- *Added by* and *Modified by* are the users who added the rule and who modified the rule. These fields are visible because the current user is logged in as "Admin" (has a higher level of priority).
- *Added on* and *Modified on* represent the time when the rule was added, respectively modified. These fields are visible only by users with a higher level of priority.

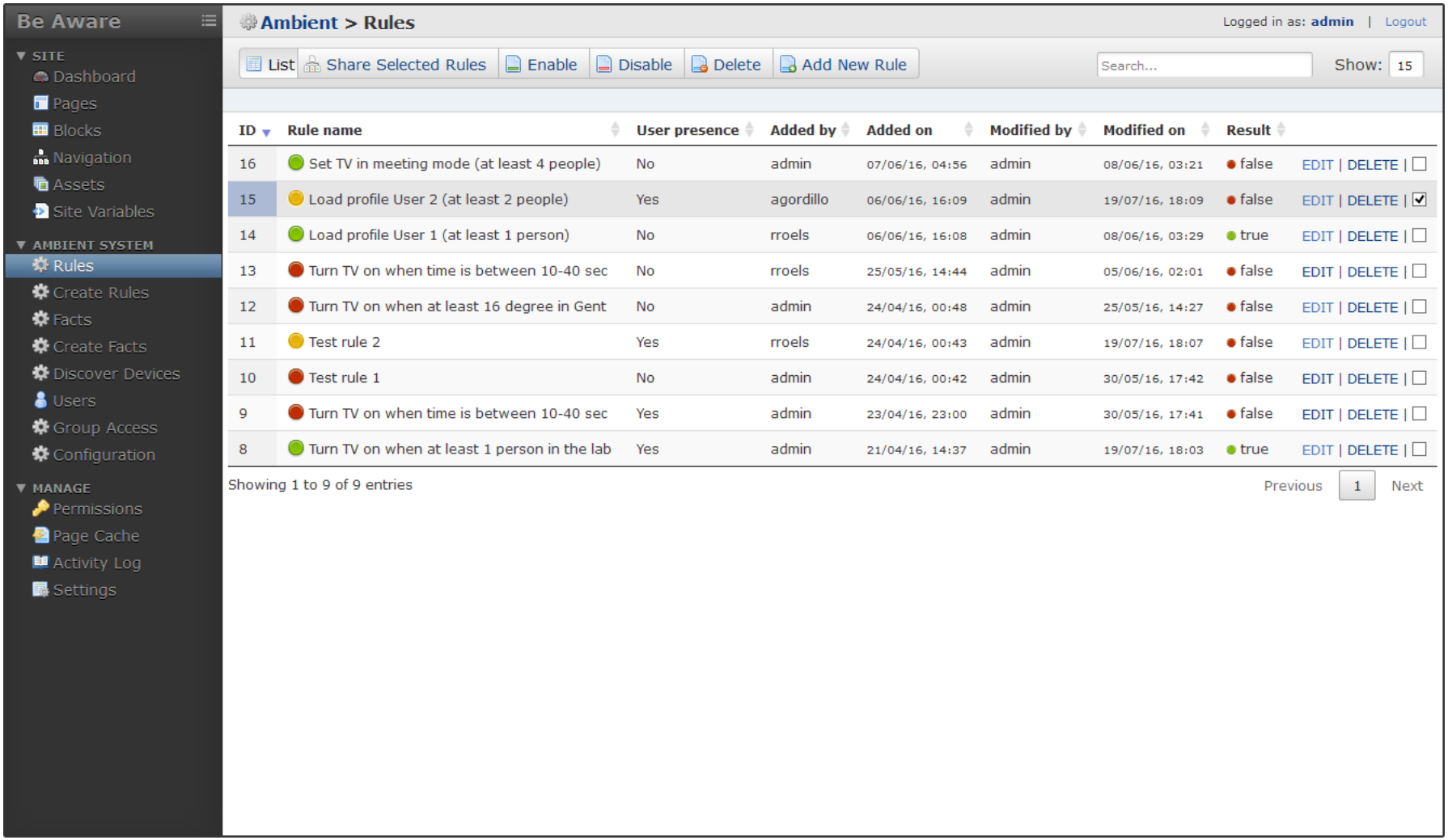

Figure 5.11: The user interface of the page that lists the user's rules

- *Result* is the result of the rule at the moment when the page is loaded.

The user are able to select rules and execute a series of operation such as: enable, disable, delete or share. The share operation will be discussed further.

5.4.2.5 Share Rules

Figure 5.12 shows the pop-up that appears on the page "List Rules" after the user has selected three rules from the list and clicked on the button "Share Selected Rules". The pop-up is implemented based on the task model called "Manage Rules" (task 13. from Section 4.1.1.1).

The pop-up contains a table showing the list of rules selected by the user. Under the table is a field called "Sharing method" that represents the user's choice of how the user wants to share the rules with other users. There is possible to choose one of the following options:

- **User account**
  The option allows users to share rules using the username of another user. When this option is selected, an input field called "User name" appears. While the user is typing in that field, a list with existing usernames is showing up and the user can select the username of who should receives the rules. The receiver of the rules, will be notified and has the possibility to review the rules before to approve or reject them.
- **Email**
  The second option permits to share the rules by using the email address of a user. When this option is selected an input field called "Email address" appears. After filling in the field with the email address of who is going to receive the rules and clicking on the button "Share Rules", an email containing a link will be sent to the email address that was filled in. The receiver of the email, follows the link, reviews the rules and then approves or rejects them.
- **Link**
  By selecting this option, the program generates in the next step a link based on the selected rules. That link can be shared with anyone.
- **Social Media**
  The option allows to share the rules through social media such as Facebook and Twitter.

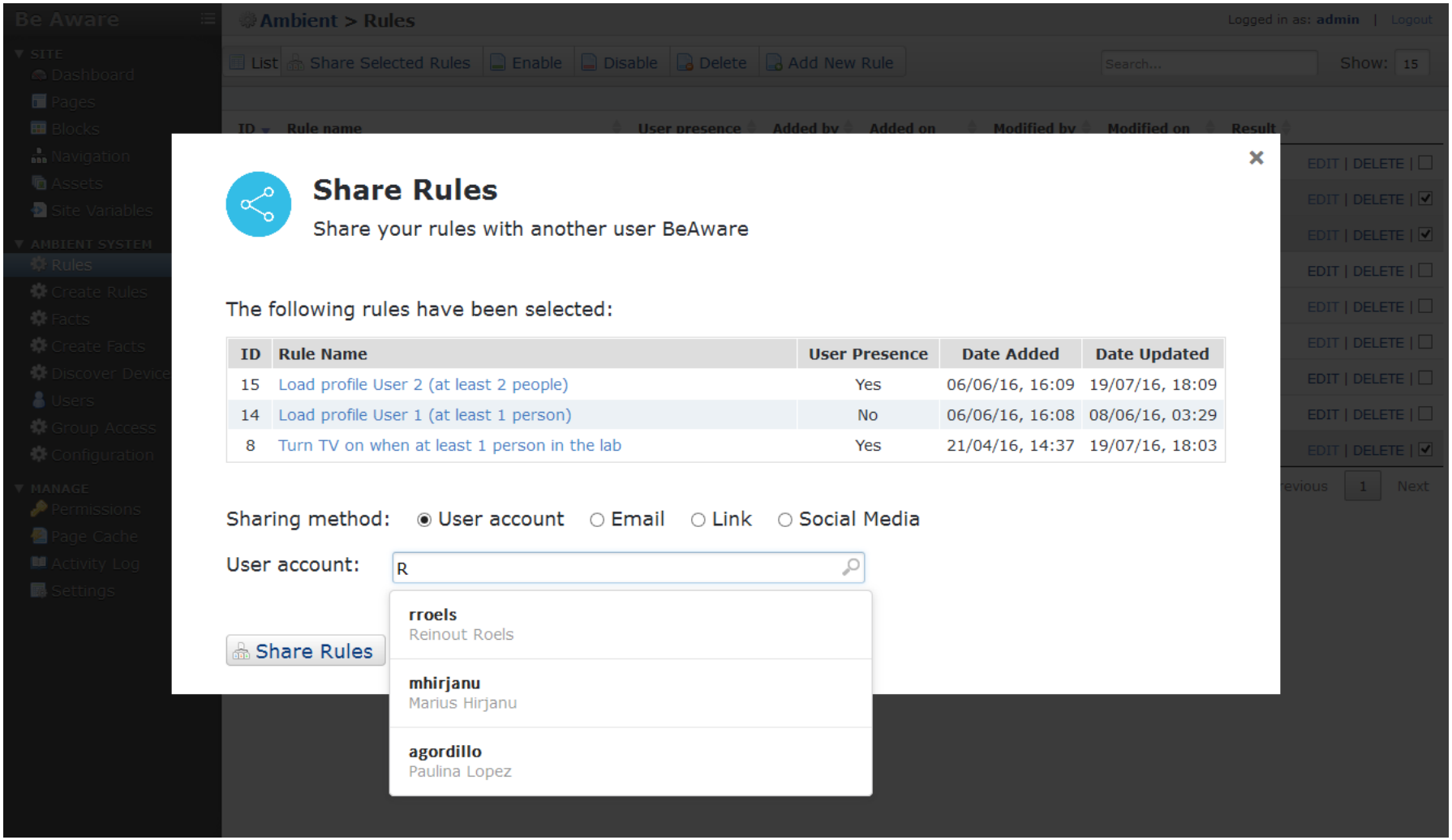

Figure 5.12: The user interface of the page that allows users to share rules with others

6 Use Cases

In this chapter we present some use cases of how our framework is used to deploy smart ambient systems. Therefore, an ambient system has been integrated in the WISE research laboratory of VUB using the BeAware framework and a number of three use cases has been demonstrated. These three use cases are the following: *Saving Energy Mode*, *Relying on the User Profiles* and *Meeting Mode*. Before describing theses use cases we would like to give an overview of the ambient system.

6.1 Overview

This section includes an overview of how an ambient system can be set up using the BeAware framework and introduces the architecture of an ambient system that has been deployed in the WISE research laboratory.

The system architecture is shown in Figure 6.1 and includes a block called "Framework" in the upper side and some devices connected to it in the lower side. The block "Framework" represents a simplified version of the initial framework, that was earlier introduced as the framework architecture in Figure 5.1. Now, the "Framework" has three main components (*User Interface*, *Database* and *System Core*) and is running on a server called "Web Server 1". The "Web Server 1" is runs on a `VMWare` virtual machine with op-

erating system Ubuntu, and is composed of the following software: `Node.js`, `Apache`, `PHP`, `MongoDB` and `MySQL`. The framework is connected to a router via a LAN connection and exchanges information using the RESTfull API and WebSocket interface, with two devices that we call "facts" in this context (`Kinect` and `Raspberry PI`[1]).

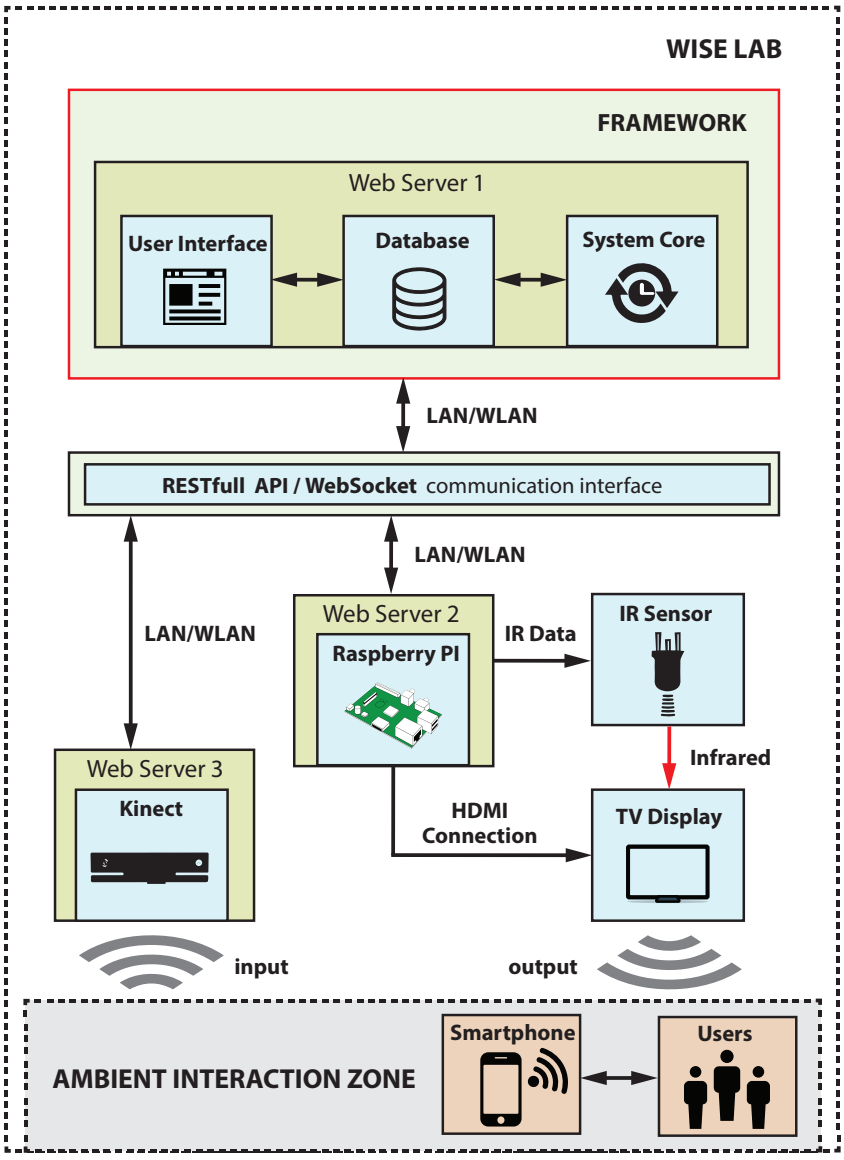

Figure 6.1: The architecture of the ambient system deployed to WISE lab using the BeAware framework

[1]Raspberry PI: https://www.raspberrypi.org/

`Kinect` is a fact of type "Input" that allows the tracking and counting of people who are present in the laboratory. A real-time counter is accessible by the framework through the REST interface. `Raspberry PI` is a fact of type "'Output' that allows the framework to fulfil two functions: to control the TV device via the IR sensor and to update the content on the TV display via the High-Definition Multimedia Interface (HDMI). Both devices, the `Kinect` and the `Raspberry PI`, are running on their own web servers (Web Server 3 and Web Server 2).

Next, having given an overview of the system, we will present three use cases that show how our framework can be used. For each of these use cases a scenario will be given and it will show how the system is set up and discuss what technically happens when a scenario is executed.

6.2 Use Case 1: Energy Saving Mode

6.2.1 Scenario

In our university research laboratory, there are a couple of electronic devices that help students and the academic staff to improve their studies, teach and research. These electronic devices (such as: TVs, multimedia projectors, displays, heating, air conditioning) must be switched off during the night or in the days when nobody is working in the lab. So even if it does not require a lot of time to daily switch *off* and *on* these devices, this is a repetitive task and sometimes it could happen that students or academic staff forget to comply with it. That means that an additional consumption of energy and short life of the devices.

To avoid this we came up with a solution that automates this task by using the ambient system already created with our BeAware framework. In this use case we will only show how a TV can be controlled via the framework, but any of the given devices can be controlled in a similar way.

6.2.2 Setting up the System

In order to achieve the task described earlier using our framework, it is required to do two actions beforehand:

1) Create two new facts using the user interface of the framework (one fact of type "Input" and another fact of type "Output"). First fact is assigned to Kinect which is called "Counter people" and retrieves the number of people

detected in the lab at any moment by Kinect. The second fact is attributed to TV, is called "Switch TV on/off" and allows the framework to switch the TV "off" or "on".

2) Define a rule based on these two facts created. The rule will be: switch the TV on only when at least one person is present in the lab and is during the day (between the hours 7:00 and 20:00). Outside these hours the TV must be switched off. The rule's condition is defined in the graphical interface of the framework and looks like in the Figure 6.2.

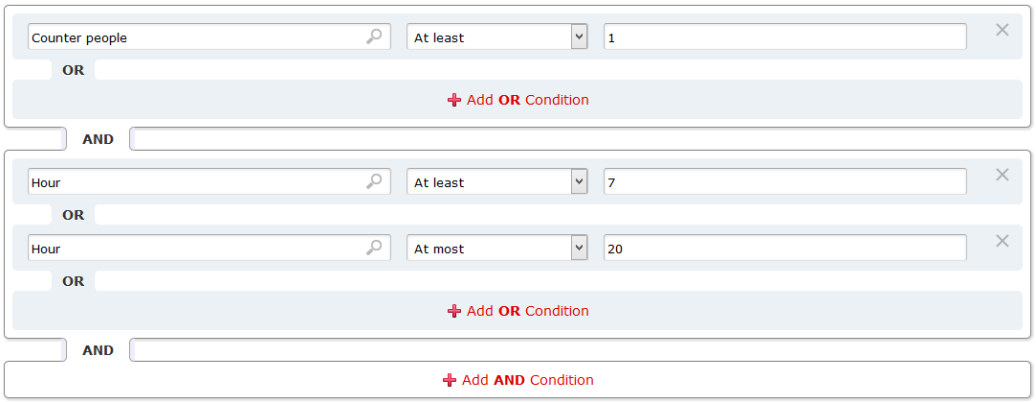

Figure 6.2: The rule's condition for archiving the first scenario

The rule's consequence contains the fact that it will be triggered when the rule's condition is satisfied and is shown in the Figure 6.3.

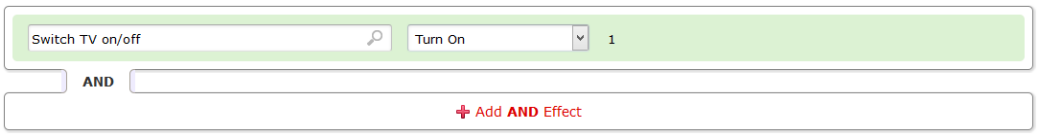

Figure 6.3: The rule's consequence of the first scenario

6.2.3 The Execution of the Scenario

By executing this scenario, a series of things will take place. Initially, the Kinect senses the users that are present in the "Ambient Interaction Zone" of the lab (Figure 6.1) and via software, it converts the visual recognition in a variable containing the total number of people. Secondly, the framework retrieves the value of that variable via the RESTfull interface and stores it in the database. Then, based on the defined rule's condition, the system

core of the framework performs reasoning and triggers the `Raspberry PI` device with a value defined in rule's consequence. Lastly, the `Raspberry PI` processes the value that is received and sends a signal code via infra-red to the *TV Display*. The signal code is similar with the signal sent by a TV remote control. The TV will be switched "on" and "off" according to signal code received. Additionally, the `Raspberry PI` can be configured to send a sequence of signal codes to the TV. Moreover, a sequence of codes was implemented in our scenario that switches on the TV but also selects the HDMI port as input source of the TV.

6.3 Use Case 2: System Relying on Profiles

6.3.1 Scenario

In the second use case, we want to keep the scenario discussed previously in the first use case (Section 6.2.1) and additionally build further on it. The new scenario consists of showing personalised content on the *TV Display*, but customised to the people that are present in the laboratory. Therefore, the ambient system will be composed of two people, which are called "User 1" and "User 2". The "User 1" would like to see on the *TV Display* a slide of images whereas the "User 2" prefers some more informative content displayed on the TV screen such as: today's menu of the university restaurant, their personal Google Calendar[2] or the weather forecast. Moreover, in case that both users will be working simultaneously in the lab, the content of the screen will be adjusted to the user who came first in the lab.

6.3.2 Setting up the System

Because this scenario is an extension of the scenario from the first use case (Section 6.2.1), it will be necessary to keep the facts and rules added earlier and more than that, create two rules and one fact of type "Output". The new fact is called "Update TV display" and allows to update the content of TV screen according to the name of a HTML document that is defined into the rule's consequence. Both rules that are necessary to be created, will contain the condition "Counter people at least 1" (similar with the first condition shown in Figure 6.2) and will have a distinct rule consequence. Practically, another document name will be defined for each user. Figure 6.4 shows the rule's consequence of the rule defined for "User 1". The consequence

[2]Google Calendar: https://calendar.google.com

of the second rule will instead contain the value "index1.html", a value of the document assigned to "User 2", respectively the value "index2.html". An important thing is that for these both rules an option will be activated to consider the rule only when the users are in the smart lab (see Section 5.4.2.3).

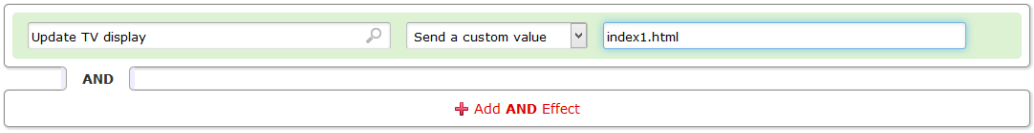

Figure 6.4: The rule's consequence of the second scenario

6.3.3 The Execution of the Scenario

In addition to the first scenario (Section 6.2.1), the system will execute a sequence of actions. First, the *System Core* of the framework (Figure 6.1), is checking regular the database for any new rules or facts added through the *User Interface*. Once that they are added in the *Database*, the *System Core* will execute reasoning with regards to the rule conditions. If one of the two users will arrive to the lab, their smartphone will notify the framework that the rules of that user must be activated and processed. Therefore, if a rule associated to that user is satisfied, the framework will trigger a local API that inserts the document name into the database. The last document name is published to subscribers via *WebSocket* interface. One of the subscribers is the `Raspberry PI` device, which plays the received document name and transmits it via HDMI interface to the *TV Display*.

6.4 Use Case 3: Meeting Mode

6.4.1 Scenario

In the third use case we will use our framework to show relevant material on the *TV Display* in a meeting room, targeted towards the people that attend the meeting. Therefore, the ambient system must identify if at least four people are present in the lab, and a video that is related to the meeting should be played automatically on the TV screen.

6.4.2 Setting up the System

In order to achieve the scenario described in this use case, it will be necessary to reuse the fact "Counter people" that was defined in the first use case (Section 6.2.1) and create a new rule using the GUI of our framework. The rule must have selected the option "When to consider this rule" to "Always" and have defined a condition and consequence as it is shown in the Figure 6.5.

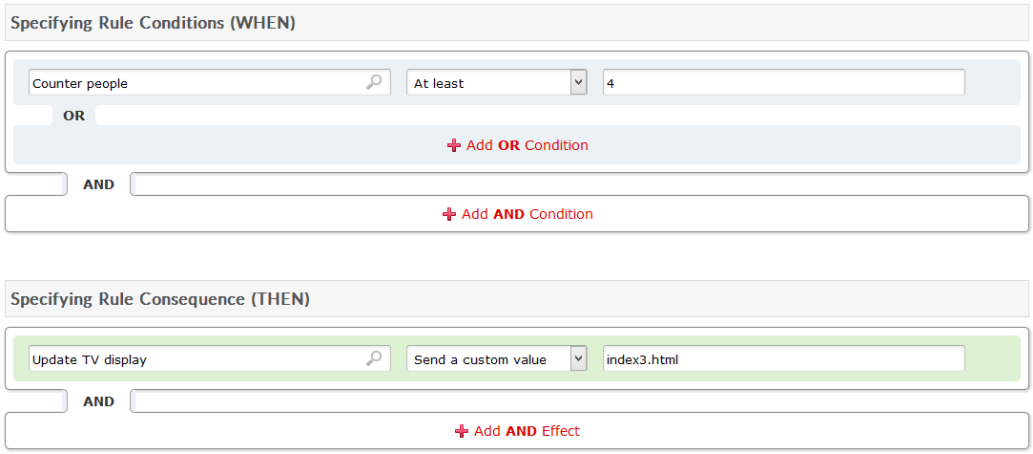

Figure 6.5: The rule's condition and consequence of the third scenario

6.4.3 The Execution of the Scenario

The ambient system will execute this scenario similar with the execution of the scenario from the previous use case (Section 6.3).

6.5 Conclusion

In this chapter we presented the use of our framework by showing three use cases. For each of these use case, the scenario was described, the way of how the system must be set up and what happens behind the scenes when the scenario is executed. We have seen how the physical devices can be linked to the framework's facts and how they can be defined in a friendly graphical interface and then, reused in multiple scenarios. We have also seen, how easy can be created rules that are relying on the facts defined earlier and deploy ambient systems, thanks to the developed framework.

7 Conclusion

This final chapter concludes the thesis by summarising our research work done for the development of the BeAware framework and its importance for easily creating smart ambient environments. It begins with a summary starting from the problem statement, then it presents our contribution and ends up with limitations and possible future work.

7.1 Summary

An ambient system refers to an application which embeds a set of electronic devices into physical environments and make them receptive and responsive to the presence of people. The objective of this research was to investigate the current Ambient Intelligence paradigm and determine an optimal context-aware framework that allows to quickly deploy ambient information systems that can be integrated in university labs. The ambient systems introduced in university laboratories may help the students and academic staff to automate their daily tasks and improve their performance in study and research.

Therefore, we came up with a set of research questions in Section 1.2 and conducted a literature study for gaining a good insight about the topic. We started our literature research by analysing some related ambient systems and digital displays, where we have identified some interesting aspects that

should be considered when designing and building ambient systems. First of all, the ambient systems are meant to work independently after they are set up and configured. They can run in the background and must operate without perturbing or distracting the users of their main tasks, which might otherwise have negative impacts on users. Second, we have seen that the ambient systems can have multiple output modalities, in which the same kind of ambient information can be outputted and redirected to numerous displays. According to the study, the *visual modality* (displaying the ambient information via *lights* and *screens*) is the most popular output modality, following by *auditory*, *tactile* and *olfactory* (see Table 2.2). Third, we noticed that for building an ambient system similar to the studied ones, a framework model is needed and not a software program developed for each system apart. The framework model should ensure the communication and interoperability between electronic devices and allows users to set up rules on them.

The main issues of actual frameworks is that they only provide a limited set of characteristics. They work only on some specific operating systems or provide support for only a restricted type of interfaces or protocols. Another issue is that the actual frameworks are not extensible. They are not compatible with the latest updates of a version, new programming languages, technologies or future hardware. Usually their development stops in the the prototype phase, not offering support for future development. Therefore, it makes it difficult to deploy an ambient system using the existing frameworks.

Therefore, in the second part of the literature study we investigated and compared some frameworks, from where we extracted a few characteristics (Table 7.1). Those characteristics allowed us to design and build a new framework called BeAware that is going beyond the barriers of actual frameworks. BeAware is a lightweight framework which supports the majority of those characteristics, making it ready for deploying ambient information systems.

7.2 Contributions

We succeeded in achieving our goals with this thesis. Based on the analysis that was performed on the existing frameworks, we identified the main characteristics that a framework must have and we listed in a table (Table 7.1). This analysis may lead to a first contribution of our work. In accordance to these characteristics, we defined the requirements of our framework and identified the classes of users. Then, we did a modelling of the user tasks and presented the ORM conceptual schema and UML diagram of our framework.

Another part of our contribution is reflected in the implementation of the framework. We constructed our architecture model that works on top of the classical network configuration via REST web services and WebSockets. The BeAware framework consists of a user interface and a core. The user interface allows users to define facts (sensors, devices or web services) and to create rules between the defined facts, in a user-friendly graphical interface. The core is used to process in real-time the facts and rules created.

In Table 2.2 we can see a comparison of our framework (BeAware) and the related frameworks that we have investigated (the meaning of each characteristic was described earlier in Section 2.4.1). We succeeded by developing a framework that supports most of the characteristics.

Framework	**Context**						**Modularity**	**API**	**WOA**	**Visualisation**
	Personalisation	**Adaptation**	**Reasoning**	**Acquisition**	**Discovery**	**History**				
The Context Toolkit	✓			✓		✓	✓		✓	
CASanDRA				✓			✓	✓	✓	
DMS-CA	✓		✓	✓			✓			
HYDRA	✓		✓	✓		✓	✓			
WoTKit						✓	✓	✓	✓	✓
CARISMA	✓		✓		✓			✓		
RecAm	✓		✓			✓				
COSMOS	✓		✓	✓			✓			
Feel@Home	✓	✓	✓	✓		✓	✓	✓		
Octopus				✓			✓	✓	✓	
BeAware	✓	✓	✓	✓	✓	✓	✓	✓	✓	

Table 7.1: Comparison between our framework (BeAware) and other frameworks

API - Application Programming Interface; **WOA** - Web-Oriented Architecture;

The first characteristic of our framework (Context Personalisation), is that it can detect and identify the users that are present in the lab environment. It allows users to create rules that are only considered when those specific users are in the lab. Then, the rules that are created can be used in a large number of lab environments. The framework tracks the location of the user and activates the rule only in that environment where the user is located at that moment. Next, the framework enables the users to share rules with other user accounts, via email addresses, via links or via social media networks such as Facebook or Twitter. The framework also includes support for auto-discovering devices (allowing users to add new devices in an easier manner) and to control multimedia presentations. It has an architecture that permits developers to easy implement new extensions. It is platform-independent, allowing the users to access the user interface with any device that contains a browser. The framework is extensible and uses some of the latest technologies, in order to bring support for future development.

The use of our framework has been demonstrated in the WISE research laboratory via three use cases: Energy Saving Mode, System Relying on Profiles and Meeting Mode (see Chapter 6).

7.3 Limitations and Future Work

In the previous section we presented our contribution and have seen the strengths of our framework when compared with other solutions. This said, it also has some limitations. Because our framework uses the REST communication interface, the sensors and devices that users want to connect to the framework must have their own web server that provides RESTfull API. The second limitation comes up when we want to connect two lab environments to our framework, that are located far to each other. Because our framework uses a LAN connection for device auto-discovery and identification of people, it will be necessary to switch to another network configuration (such as WAN or VPN), in order to be able to connect both environments. The third limitation is related to the identification of people, in which the user's smartphone is used as user identification source. In case the user's smartphone is missing or the WiFi connection of the smartphone is turned off, the framework will not be able to detect and identify that user. A solution to solve this issue is to use another identification technology, for instance RFID.

The limitations described in the previous paragraph, may be considered as potential future work. Moreover, some others could be taken into account. First, it will be useful to be developed an intelligent conflict resolution algorithm, that helps to avoid conflicts between the rules defined by users. In the current version of our framework an algorithm is implemented that restricts access to a device if it would be triggered by multiple rules. For instance, once a rule's condition is true, the framework triggers devices defined in rule consequences section and marks them as being used. As long as they in use, the other rules can not trigger those devices and will be on hold until the initial rule's condition becomes false. Second, it might be beneficial to develop a smartphone application that sends the current GPS location of the user to the framework. This will allow the system to predict the user presence, before the user is physically present in the lab environment. Third, as future work we propose to implement a feature that allows values retrieved from facts to be shown in real-time. This will bring an increase in usability for the user interface and permits to visualise the current values of facts without the refresh of a web page. Last but not least, it may be useful if the framework supports also the "Visualisation" characteristic (Table 7.1). This characteristic allows data received from facts to be shown in a graphical format such as charts, plots, and graphs. Finally, as future work we propose an evaluation of the user interface. This will allow to test the usability of the system and find potential issues.

Bibliography

[1] M. Weiser, "The Computer for the 21st Century," *Mobile Computing and Communications Review*, vol. 3, no. 3, pp. 3–11, July 1999.

[2] S. K. Card, A. Newell, and T. P. Moran, *The Psychology of Human-Computer Interaction.* Hillsdale, USA: Lawrence Erlbaum Associates, January 1983.

[3] H. Ishii, C. A. Wisneski, S. Brave, A. Dahley, M. Gorbet, B. Ullmer, and P. Yarin, "ambientROOM: Integrating Ambient Media with Architectural Space," in *Proceedings of the 16th ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 1998), April 18–23, 1998*, C.-M. Karat and A. Lund, Eds., Los Angeles, USA, April 1998.

[4] A. Mahomed, P. Novais, A. Pereira, G. V. González, and A. Fernández-Caballero, *Ambient Intelligence - Software and Applications. 6th International Symposium on Ambient Intelligence (ISAmI 2015), Salamanca, Spain, June 3-5, 2015.* Springer International Publishing, January 2015.

[5] L. M. Camarinha-Matos and H. Afsarmanesh, "Collaborative Systems for Smart Environments: Trends and Challenges," in *Collaborative Systems for Smart Networked Environments*, L. M. Camarinha-Matos and H. Afsarmanesh, Eds. Berlin, Germany: Springer Berlin Heidelberg, 2014, vol. 434, ch. IFIP Advances in Information and Communication Technology, pp. 3–15.

[6] D. Bandyopadhyay and J. Sen, "Internet of Things: Applications and Challenges in Technology and Standardization," *Wireless Personal Communications*, vol. 58, no. 1, pp. 49–69, May 2011.

[7] D. J. Cook, J. C. Augusto, and V. R. Jakkula, "Ambient Intelligence: Technologies, Applications, and Opportunities (Review)," *Pervasive and*

Mobile Computing, vol. 5, no. 4, pp. 277–298, August 2009.

[8] L. Richardson and S. Ruby, *Restful Web Services*, 1st ed. Sebastopol, USA: O'Reilly Media, 2007.

[9] R. Baldoni, L. Querzoni, and A. Virgillito, *Distributed Event Routing in Publish/Subscribe Communication Systems: A Survey.* Roma, Italy: Sapienza University of Rome, 2005.

[10] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, June 2003.

[11] M. Weiser and J. S. Brown, "Designing Calm Technology," Xerox PARC, December 1995, accessed on 2015-08-08. [Online]. Available: http://www.ubiq.com/weiser/calmtech/calmtech.htm

[12] P. D. Adamczyk and B. P. Bailey, "If Not Now, When? The Effects of Interruption at Different Moments Within Task Execution," in *Proceedings of the 22nd ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 2004), April 24–29, 2004*, G. D. Abowd and A. K. Dey, Eds., Vienna, Austria, 2004, pp. 271–278.

[13] H. Ishii and B. Ullmer, "Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms," in *Proceedings of the 15th ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 1997), March 22–27, 1997*, S. Pemberton, Ed., Atlanta, USA, 1997, pp. 234–241.

[14] C. A. Wisneski, H. Ishii, A. Dahley, M. Gorbet, S. Brave, B. Ullmer, and P. Yarin, "Ambient Displays: Turning Architectural Space into an Interface between People and Digital Information," in *Cooperative Buildings: Integrating Information, Organization, and Architecture. First International Workshop (CoBuild 1998), Darmstadt, Germany, February 25–26, 1998*, ser. Lecture Notes in Computer Science, N. A. Streitz, S. Konomi, and H.-J. Burkhardt, Eds. Darmstadt, Germany: Springer Berlin Heidelberg, February 1998, vol. 1370, pp. 22–32.

[15] I. Siio and N. Mima, "Meeting Pot: Coffee Aroma Transmitter," in *Proceedings of the 3rd ACM International Conference on Ubiquitous Com-*

puting (UbiComp 2001), September 30 – October 2, 2001, G. D. Abowd and A. K. Dey, Eds., Atlanta, USA, 2001, pp. 7–8.

[16] C. Decurtins, M. C. Norrie, E. Reuss, and N. Weibel, "AwareNews - A Context-Aware Peripheral News and Awareness Display," in *Proceedings of the 4th International Conference on Intelligent Environments (IE 2008), July 21–22, 2008*, S. Helal, H. Hagras, and V. Callaghan, Eds., Seattle, USA, 2008, pp. 1–8.

[17] T. Prante, C. Röcker, N. Streitz, R. Stenzel, C. Magerkurth, D. V. Alphen, and D. Plewe, "Hello.Wall – Beyond Ambient Displays," in *Proceedings of the 5th International Conference on Ubiquitous Computing (UbiComp 2003), Octomber 12–15, 2003*, P. Ljungstrand and J. Brotherton, Eds., Seattle, USA, 2003, pp. 277–278.

[18] N. A. Streitz, C. Rocker, T. Prante, D. van Alphen, R. Stenzel, and C. Magerkurth, "Designing Smart Artifacts for Smart Environments," *Computer*, vol. 38, no. 3, pp. 41–49, Mar. 2005.

[19] M. Altosaar, R. Vertegaal, C. Sohn, and D. Cheng, "AuraOrb: Using Social Awareness Cues in the Design of Progressive Notification Appliances," in *Proceedings of the 18th Australia Conference on Computer-Human Interaction: Design: Activities, Artefacts and Environments (OZCHI 2006), November 20–24, 2006*, J. Kjeldskov and J. Paay, Eds., Sydney, Australia, 2006, pp. 159–166.

[20] H. Müller, A. Kazakova, M. Pielot, W. Heuten, and S. Boll, "Ambient Timer – Unobtrusively Reminding Users of Upcoming Tasks with Ambient Light," in *Proceedings of the 14th International Conference on Human-Computer Interaction (INTERACT 2013), September 2-6, 2013, Part I*, ser. Lecture Notes in Computer Science, P. Kotzé, G. Marsden, G. Lindgaard, J. Wesson, and M. Winckler, Eds. Berlin, Germany: Springer Berlin Heidelberg, 2013, vol. 8117, pp. 211–228.

[21] J. K. Hong, S. Song, J. Cho, and A. Bianchi, "Better Posture Awareness through Flower-Shaped Ambient Avatar," in *Proceedings of the 9th International Conference on Tangible, Embedded, and Embodied Interaction (TEI 2015), January 15–19, 2015*, B. Verplank and W. Ju, Eds., Stanford, USA, January 2015, pp. 337–340.

[22] C. A. Wisneski, "The Design of Personal Ambient Displays," diploma thesis, Massachusetts Institute of Technology, Cambridge, USA, 1999.

[23] D. Salber, A. K. Dey, and G. D. Abowd, "The Context Toolkit: Aiding the Development of Context-Enabled Applications," in *Proceedings of the 17th ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 1999), May 15–20, 1999*, M. Altom and M. Williams, Eds., Pittsburgh, USA, 1999, pp. 434–441.

[24] A. K. Dey, G. D. Abowd, and D. Salber, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," *International Journal of Human-Computer Interaction*, vol. 16, no. 2, pp. 97–166, December 2001.

[25] A. K. Dey and G. D. Abowd, "Towards a Better Understanding of Context and Context-Awareness," in *Proceedings of the 18th ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 2000), April 1–6, 2000*, D. R. Morse and A. K. Dey, Eds., The Hague, Netherlands, 2000, pp. 1–12.

[26] A. M. Bernardos, P. Tarrío, and J. R. Casar, "CASanDRA: A Framework to Provide Context Acquisition Services and Reasoning Algorithms for Ambient Intelligence Applications," in *Proceedings of the 10th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2009), December 8–11, 2009*, S.-J. Horng and H. R. Arabnia, Eds., Hiroshima, Japan, 2009, pp. 372–377.

[27] J. Herbert, J. O'Donoghue, and X. Chen, "A Context-Sensitive Rule-based Architecture for a Smart Building Environment," *International Journal of Smart Home*, vol. 3, no. 1, pp. 1–6, January 2009.

[28] A. Badii, M. Crouch, and C. Lallah, "A Context-Awareness Framework for Intelligent Networked Embedded Systems," in *Proceedings of the 3rd International Conference on Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services (CENTRIC 2010), August 22–27, 2010*, R. Hariprakash, E. Egyed-Zsigmond, and M. Hoffmann, Eds., Nice, France, 2010, pp. 105–110.

[29] M. Bali, *Drools JBoss Rules 5.0 Developer's Guide*, 1st ed. Birmingham, UK: Packt Publishing, 2009.

[30] H. Guermah, T. Fissaa, H. Hafiddi, M. Nassar, and A. Kriouile, "An Ontology Oriented Architecture for Context Aware Services Adaptation," *International Journal of Computer Science Issues*, vol. 11, no. 2, pp. 24–33, Mar. 2014.

[31] M. Sarnovský, P. Kostelník, P. Butka, J. Hreňo, and D. Lacková, "First Demonstrator of HYDRA Middleware Architecture for Building Automation," in *Proceedings of Annual Conference on Knowledge Based Technologies Znalosti, February 13–15, 2008*, J. Rauch and P. Navrat, Eds., Bratislava, Slovakia, 2008, pp. 204–214.

[32] M. Blackstock and R. Lea, "WoTKit: A Lightweight Toolkit for the Web of Things," in *Proceedings of the 3rd International Workshop on the Web of Things (WOT 2012), June 19, 2012*, S. Mayer, D. Guinard, and E. Wilde, Eds., Newcastle, United Kingdom, 2012, pp. 1–6.

[33] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the Internet of Things to the Web of Things: Resource-Oriented Architecture and Best Practices," in *Architecting the Internet of Things*, D. Uckelmann, M. Harrison, and F. Michahelles, Eds. Berlin, Germany: Springer Berlin Heidelberg, March 2011, vol. 1, pp. 97–129.

[34] M. Blackstock and R. Lea, "Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED)," in *Proceedings of the 5th International Workshop on Web of Things (WOT 2014), October 8, 2014*, S. Mayer, D. Guinard, and E. Wilde, Eds., Cambridge, USA, 2014, pp. 34–39.

[35] L. Capra, W. Emmerich, and C. Mascolo, "CARISMA: Context-Aware Reflective Middleware System for Mobile Applications," *IEEE Transactions on Software Engineering*, vol. 29, no. 10, pp. 929–945, October 2003.

[36] M. Alhamid, M. Rawashdeh, H. Dong, M. A. Hossain, A. Alelaiwi, and A. E. Saddik, "RecAm: A Collaborative Context-Aware Framework for Multimedia Recommendations in an Ambient Intelligence Environment," in *Multimedia Systems*. Springer Berlin Heidelberg, May 2015, pp. 1–15.

[37] T. Hofmann, "Latent Semantic Models for Collaborative Filtering,"

ACM Transactions on Information Systems, vol. 22, no. 1, pp. 89–115, January 2004.

[38] D. Conan, R. Rouvoy, and L. Seinturier, "Scalable Processing of Context Information with COSMOS," in *Distributed Applications and Interoperable Systems. 7th IFIP WG 6.1 International Conference, DAIS 2007, Paphos, Cyprus, June 6-8, 2007. Proceedings*, ser. Lecture Notes in Computer Science, J. Indulska and K. Raymond, Eds. Berlin, Germany: Springer Berlin Heidelberg, 2007, vol. 4531, pp. 210–224.

[39] B. Guo, L. Sun, and D. Zhang, "The Architecture Design of a Cross-Domain Context Management System," in *Proceedings of the 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), March 29–April 2, 2010*, C. Becker and M. Conti, Eds., Mannheim, Germany, 2010, pp. 499–504.

[40] B. Firner, R. S. Moore, R. Howard, R. P. Martin, and Y. Zhang, "Poster: Smart Buildings, Sensor Networks, and the Internet of Things," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys 2011), November 1-4, 2011*, J. Liu and P. Levis, Eds., Seattle, USA, 2011, pp. 499–504.

[41] S. Mayer, D. Guinard, and V. Trifa, "Searching in a Web-Based Infrastructure for Smart Things," in *Proceedings of the 3rd International Conference on the Internet of Things (IOT 2012), October 24–26, 2012*, L. Zhengseries and H. Min, Eds., Wuxi, China, 2012, pp. 119–126.

[42] W. K. Edwards, M. W. Newman, and E. S. Poole, "The Infrastructure Problem in HCI," in *Proceedings of the 28th ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 2010), April 10-15, 2010*, E. Mynatt and K. Edwards, Eds., Atlanta, USA, 2010, pp. 423–432.

[43] M. Chemuturi, *Requirements Engineering and Management for Software Development Projects*, 1st ed. New York, USA: Springer-Verlag, 2013.

[44] ISO-9241-11, *Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) – Part 11: Guidance on Usability*, 1st ed. International Standard: ISO, 1998.

[45] G. Mori, F. Paternò, and C. Santoro, "CTTE: Support for Developing and Analyzing Task Models for Interactive System Design," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, pp. 797–813, August 2002.