Department of Informatics
University of Fribourg (Switzerland)

Modelling and Controlling Smart Residential Environments

The GF4SRE Software Framework and the GPL4SRE Domain Specific Language

THESIS

submitted to the Faculty of Science of the University of Fribourg (Switzerland)
in fulfilment of the requirements for the degree of Doctor Scientiarum Informaticarum

by

ABDALADHEM ALBRESHNE

from Agelat (Libya)

Thesis No. 1937

UniPrint, Fribourg

2015

Accepted by the Faculty of Science of the University of Fribourg following the proposal of:

- Prof. Dr. Ulrich Ultes-Nitsche, University of Fribourg (Jury President)
- Prof. Dr. Jacques Pasquier, University of Fribourg (Thesis Supervisor)
- Prof. Dr. Marino Widmer, University of Fribourg (Expert)
- Prof. Dr. Jacques Savoy, University of Neuchâtel (External Expert)

Fribourg, November 23th, 2015

Thesis Supervisor	Faculty Dean
[illegible]	F. [illegible]
Prof. Dr. Jacques Pasquier	Prof. Dr. Fritz Müller

Acknowledgements

This thesis is the result of several years of research in the area of software engineering. First of all, I would like to express my heart-felt gratitude and thanks to professor Jacques Pasquier for his advice and unremitting support during all the phases of my thesis research and for giving me the chance to develop my own ideas. He trained me in how to do research and how to write, and encouraged me not to be intimidated by difficult problems. He was very generous with his time and ideas. I can never thank him enough for being an excellent mentor and a wonderful supervisor.

My thanks also go to the informatics' department staff for their generous support and the department's friendly atmosphere. Finally, I am very thankful to my family and my friends for their unconditional support and understanding during all my studies.

Abstract

Today, research into smart environments represents new innovative work and interesting challenges for pervasive computing. Remote services and miniaturized devices are becoming more and more varied, affordable and important in our daily lives, but unfortunately, most of these devices, and services are incompatible, isolated and cannot easily function cooperatively with each other. The domain of smart residential environments (SRE) like smart homes constitutes a typical example of an environment characterized by complex requirements with regard to context awareness, heterogeneity, interoperability, discovery, appliance control and orchestration, as well as easy service management for end users. Innovative development paradigms have been suggested to meet these new demanding conditions. One is service oriented computing, which has been proposed as a new solution for heterogeneous and changing environments. As a consequence, service composition and orchestration research explores ways to combine available services and smart objects to work together to solve complex problems such as energy control, security, health care, etc.

In spite of a number of attempts concerning the control of smart environments, problems remain. These are mainly related to issues of how to describe the smart environment and devices that users, respectively, live in and interact with. There is also the problem of interoperability, heterogeneity, discovery, and the orchestrating and executing of services in SRE, allowing users to control it in both a transparent and abstract way.

To overcome these deficiencies and limitations, this thesis proposes a generic framework for SRE (GF4SRE) architecture offering generic and flexible components that simplify the development of control scenarios in smart environments. The main components consist first in defining a generic Ontology for a Smart Residential Environment (Ont4SRE) and then in using it within a Domain Specific Language (DSL) called Generic Process Language for Smart Residential Environments (GPL4SRE) to build generic process templates oriented toward users' goals. Additionally, a translator is provided to convert the templates into executable ones, and a process execution engine with the capability to execute them.

To address the challenges related to the interoperations between the physical world and digital world across a heterogeneous hardware and software platform, the web services with semantic annotations and their emerging technologies are used. They integrate low-level decoupled services with high-level control processes by providing semantic awareness, and service filtering capabilities.

Concerning the discovery of the physical world (i.e. physical components like object, devices, sensors), especially, how to identify the physical objects which provide the intended services for our applications, a semantic-based discovery and lookup infrastructure that takes into account the particular context of smart environments is proposed.

To evaluate the advantages of applying the proposed framework, the proposed architecture has been implemented and tested through a case study realized in the field of smart residential environments - in particular for security and energy saving scenarios.

Keywords: Smart Residential Environments; Smart-entities, Objects and Appliances; Sensors; Actuators; Web Services Composition and Orchestration; Ontology; Domain Specific Language.

Résumé

De nos jours, les recherches sur les environnements intelligents représentent un nouveau défi intéressant dans le domaine de l’informatique pervasive. Les objets connectés via un réseau informatique avec des actuateurs et des senseurs sont de plus en plus variés, abordables et importants dans notre vie quotidienne. Malheureusement, la plupart de ces objets, leurs dispositifs et leurs services sont encore incompatibles, isolés et ne peuvent pas facilement coopérer entre eux. Le domaine des environnements intelligents, comme les environnements résidentiels intelligents (SRE), constitue un exemple typique qui se caractérise par des exigences complexes en matière de sensibilité au contexte, d’hétérogénéité, d’interopérabilité, de découverte des services, de contrôle, d’orchestration et de gestion conviviale pour les utilisateurs. De nouveaux modèles informatiques sont récemment apparus pour répondre à ces exigences. Un d’entre eux est l'architecture orientée services, qui a été proposée comme une nouvelle solution pour les environnements hétérogènes et changeants. Ainsi, les recherches sur la composition et l'orchestration de services représentent un nouveau chemin à explorer pour combiner les services disponibles et les objets intelligents afin qu’ils puissent coopérer pour résoudre des problèmes complexes tels que le contrôle de l'énergie, la sécurité, etc.

Malgré de nombreux progrès concernant le contrôle des environnements intelligents, des problèmes subsistent. Ces derniers sont principalement liés aux points suivants: le premier consiste à décrire l'environnement intelligent et les objets avec lesquels les utilisateurs, respectivement, vivent et interagissent. Le second est lié à l'interopérabilité, l'hétérogénéité, la découverte, l'orchestration et l’exécution des services dans un SRE, tout en permettant aux utilisateurs de contrôler leur environnement d’une manière transparente et abstraite.

Pour surmonter ces limites, cette thèse de doctorat propose une structure logicielle (Framework), offrant des composants génériques et flexibles, qui simplifie le développement de scénarios de contrôle dans des environnements intelligents. La principale force du Framework consiste à définir une ontologie générique pour environnement résidentiels intelligents et à l'utiliser avec un langage dédié (DSL-Domain Specific Language) pour construire des scenarios de contrôle génériques orientés vers les objectifs finaux de l’utilisateur.

Pour relever les défis liés à l'interopérabilité entre le monde physique et le monde digital à travers un matériel hétérogène et une plate-forme logicielle, les services Web avec annotations sémantiques ont été adoptés. Ils permettent d’intégrer le bas niveau (objets et services) avec le plus haut niveau (processus de contrôle). Pour répondre à la problématique de la découverte des objets physiques (actuateurs, senseurs, etc.) capables d’offrir les services requis, une infrastructure de découverte, basée sur des annotations sémantiques, prenant en compte le contexte particulier des environnements intelligents, est proposée.

Pour évaluer les avantages du Framework proposé, une architecture a été implémentée et testée par une étude de cas réalisée dans le domaine des environnements résidentiels intelligents pour la sécurité et l'économie d'énergie.

Mots-clés: Environnements résidentiels intelligents ; Actuateurs ; Senseurs ; Composition et orchestration de services web; Langage dédié; Ontologie.

Table of Contents

List of Figures

List of Tables

List of Listings

1 Introduction

1.1 Motivation and Problem Statement

The concept of an emerging world of **S**mart **E**nvironments (SE) has existed for decades. In 1991, Weiser predicted "*a physical world that is richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in the everyday objects of our lives and connected through a continuous network*." [1]. The SE concept is linked to many research areas such as ubiquitous computing (also referred to as pervasive computing or ambient computing), context aware computing, wireless, sensors networking, etc. [2].

Nowadays, from the SE technology viewpoint, a growing number of embedded sensor-based, and control devices offer useful functionalities and services. They are becoming more and more diverse and affordable as well as essential parts of our daily activities to improve urban living conditions and make our environment smart. However, these devices and their associated services cannot be easily integrated to create new composite applications, such as controlling appliances and devices for saving energy or enhancing user comfort. Research into SE constitutes a step forward in the design of smart buildings, utilities, industries, homes, and automatic transportation systems. The interdisciplinary scope of the domain includes aspects of smart object networking and software architecture for smart environments, event prediction, and decision-making.

Recent projects and industry solutions in the domain of embedded devices have resulted in a number of low power physical and network protocols such as Zigbee [3], and 6LoWPAN [4]. These initiatives integrate physical components and devices into distributed computer networks. Many interoperability solutions involve an additional middleware layer [5] for smart systems. Examples of such solution are: Web Service[6-8], RESTful Web Service [9-

11] and Platform for Service-Oriented Computing Environment (OSGi) [12, 13]. This layer can be integrated into control appliance activities, share data and capture events. It releases users from needing to physically access appliances and devices while providing a service abstraction and an application-programming interface to the physical objects to facilitate and standardize their access according to application and development needs.

In the higher layer (application layer), suggestions concerning the control of SE have been made [14-17] with the aim of integrating the physical components into the network layer to help developers and users access, discover and control smart objects. However, both the industrial and academic communities are still experiencing a number of challenges in the control of SE. There are two main problems.

1. The first is how to describe the environment and devices that users live in and interact with. Systems and programmers need to have a good understanding of the purpose and the usage of smart objects as well as their associated services. There is the need for a single, well-defined model for describing and linking smart objects, their locations, their associated devices and the services they offer.
2. The second is the problem of modelling, discovering, composing, orchestrating, and executing services in Smart Environments so that users can be in control. This challenge involves:
   - the comprehensive infrastructure necessary for the distributed computing that a smart environment requires. This includes communication and network protocols, middleware and so on.
   - frameworks that consider the requirements and the particularity of smart environments. For instance, interoperability, the dynamic discovery of smart objects, context-awareness, composition and orchestration and human interaction.
   - framework architecture that reduces the complexity of the underlying resources and infrastructure based on modularization, abstraction, transparency, decomposability (dividing the system into modules); understanding the ability of each model; ease (facility) of changing and assembling modules.
   - the context description, so software systems are aware of changes within their environment. This includes the physical environment: location, time, temperature, rainfall, light level, etc.; the human context in terms of identity, preferences, task requirements; the virtual environment context: awareness of the smart components and the services available internally and externally, locally and remotely, in the SE.

1.2 Main Contributions

The main contributions are:

1. **An ontology for Smart Residential Environments**

The first contribution of this thesis is the definition of a generic model of smart space to identify clearly the issues central to **S**mart **R**esidential **E**nvironment (SRE) such as smart objects, actuators, sensors, and services. This has resulted in a proposed **Ont**ology for a **S**mart **R**esidential **E**nvironment (**Ont4SRE**) which can later facilitate the discovery of smart objects as well as reduce the complexity related to managing SRE and orchestrating smart objects.

2. **A Generic Framework Architecture for SRE (GF4SRE)**

The second contribution is the definition of a framework software architecture based on the proposed ontology to build complex control applications for SRE. In particular, the framework supports the following features:

- Firstly, the **Ont4SRE ontology** is integrated into the framework for both discovering and compositing services.
- Secondly, integrating a **Services Discovery** into a SRE occupied by numerous physical components requires identifying the individual objects necessary for providing the intended service. To solve this problem, a discovery and lookup infrastructure is proposed, based on the Ont4SRE ontology and taking into account the particular context of SRE.
- Thirdly, generic control scenarios in the form of templates to manage the SRE are proposed. The templates are workflow-based and written by an original **G**eneric **P**rocess **L**anguage **for S**mart **R**esidential **E**nvironments **(GPL4SRE).** Additionally, a **Process Execution Engine**, which can execute these scenarios, is used.
- Finally, *the framework should include pluggable modules* to cleanly integrate the proven technologies for modelling and controlling different SREs - home, school, hospital, etc., and *it should be a generic software solution* to transparently discover and use smart objects within control scenarios.

3. **A Domain Specific Language for Smart Residential Environments**

The **GPL4SRE** language is an essential addition to complement the software architecture. It is a process-oriented language for describing scenarios to control SRE using a precise vocabulary without requiring program intricacies such as services grounding, and can be translated transparently into a standard process language such as BPEL4WS [18] for executing tasks.

1.3 Thesis Outline

The structure of the thesis is as follows:

- Chapter 2 provides a basic introduction and definition of the SE and its properties and features.

- Chapter 3 gives the necessary background for understanding and for identifying the general problems and limitations of managing SRE. It also analyses and compares the various technologies and architectures within the SRE context used in the rest of this work in terms of constraints and needs.
- Chapter 4 reviews the related research and presents the thesis proposal. It details the requirements for a conceptual framework that can support the building of a SRE Control System. The various software and hardware components, which together form the framework, are outlined.
- Chapter 5 introduces the GPL4SRE proposed language. It shows the actual content and basic syntactical elements of a program written in GPL4SRE.
- Chapter 6 describes the case study of two complex control scenarios using the proposed framework to illustrate the usefulness of this approach.
- Chapter 7 contains a summary of the results and the conclusions drawn, with suggestions for future research.

1.4 Notation and Conventions

The following conventions are used in this thesis.

- Formatting conventions:
  - **Bold** is used for Chapter, Section and Subsection headings and important titles.
  - Keywords and definitions are denoted in *Italics*
  - Important keywords are highlighted in both **Bold** and *Italic* (***Bold***).
  - `Courier New` font is used for code Listings.
- This thesis is divided into chapters. Each chapter is divided into a hierarchy of sections, within each of which are lower subdivisions - subsections - which may consist of several paragraphs.
- Figures, examples, and listings are numbered sequentially according to the chapter they appear in. For example, the second figure of the third chapter is numbered 3.2.

2

Smart Residential Environments: Definitions, Properties, and Features

Many physical world components have been designed to allow users to control them from anywhere at any time through the internet and to effectively optimize their usage in order to make their everyday lives more comfortable, easier and cheaper. Some components have no processing ability but are coupled with miniaturized actuators and sensors to control, measure or perceive their states. Sensors give the software systems the ability to monitor the status of physical components and detect events that indicate changes in their state, e.g., a smoke detector can detect a fire, or door sensors detect when people get closer to them. Similarly, actuators give software systems the ability to control their states - turn lamps, air conditioners, or heaters on and off, or open and close doors and windows. Other objects can be mobile and have processing and data storage ability. They can have the capacity to sense their own movements and determine their location (e.g. global positioning systems (GPS), intelligent wheelchairs, mobile personal computers, and automobiles). Other elements can be portable, such as laptop computers, communicators (phone/PDA) or be implanted or wearable, such as accessories like watches, or medical instruments to support biological functions. Some objects may be passive if they do not have the capacity to interact, and are not attached to any sensor or actuator. Then, they can only be described [2].

The integration of the physical world into the digital one has resulted in the emergence of smart environments. The remainder of this chapter presents some definitions, properties, examples and features related to smart environments, summarised from the literature [2, 19-22], whilst state of the art technologies, standards, and software architectures for SEs are presented in Chapter 3.

2.1 Definition of a Smart Environment

There are many definitions of a smart environment. Some are given below.

Definition 2.1: *Smart is the ability to autonomously acquire and apply knowledge, while environment refers to our surroundings* [2].

Definition 2.2: *Smart environments consist of devices, such as sensors, controllers, and computers that are embedded in, or operate in, the physical environment, e.g., robots. These devices are strongly context aware of their physical environment in relation to their tasks, e.g., a robot must sense and model the physical world in order to avoid obstacles. Smart environment devices can have an awareness of specific user activities, e.g., doors that open as people walk towards them. They often act autonomously without any manual guidance from users. These incorporate specific types of intelligence, e.g., robots may build complex models of physical behaviour and learn to adapt their movements based upon experience* [19].

Definition 2.3: *An object is called smart when it has the ability to describe its possible interactions* [20].

In this thesis, a Smart Environment is

Definition 2.4: *an environment which consists of diverse individual smart physical components (e.g. objects, devices, and appliances) that inhabitants use. These components (also referred to as entities) are not smart in themselves. They are not smart in the sense that they are autonomously intelligent, but in the sense that they can be linked to actuators and sensors (Section 2.2) to control, measure or perceive their states so that they can be networked (Section 2.3) in the digital world and be included in software computing units with Decision Making algorithms (Section 2.4) executed by a multi-agent or centralised system or whatever is required to make the environment reactive and smart.*

A formal definition of a smart environment and its components is presented in Section 2.6. Throughout this thesis, the terms “smart-object, entity, appliance” are used to distinguish them from passive physical components.

2.2 Actuation and Sensing

Sensors and actuators are electromechanical or electronic devices, built as distinct devices or in large arrays. Actuators are designed to convert an electrical signal into a physical phenomenon (e.g. heating, cooling, moving, controlling) while by contrast, sensors convert physical phenomena (e.g. temperature, humidity) into electric signals. Table 2.1 lists some examples of the properties of the environment that can be measured and converted into an electrical signal, and vice versa.

Table 2.1: Environments properties (adapted from [2])

Properties	**Measured data**
Physical properties	pressure, temperature, humidity, flow
Motion properties	position, speed, acceleration,
Contact properties	strain, force, vibration
Presence	contact, proximity, distance, motion, range, location
Biochemical Identification	personal features, personal ID, biochemical agents
Electric properties	switch on, switch off, frequency, brightness
Furniture properties	open, close, lock, unlock

Sensors

Sensors provide input information about the state of the physical world [19]. Embedded sensor-based devices are being increasingly implanted and spread about the physical world environment to measure and collect data. They can sense room temperature, or inform users if a door is opened or closed. Sensors such as fire detectors and motion detectors can be linked to control devices to react to incoming events and alert people if an accident has occurred. Sensors can also be a part of an embedded control system. Many other types of service can be provided by sensors to support activities concerned with food and traffic monitoring, energy and water leak detection, healthcare warnings, etc. Figure 2.1(a) gives some examples of the available industry solutions for wireless sensors. According to [19], sensors can be characterised in terms of:

- ***Physical characteristics***: *including power, mobility, and size*: passive sensors take power from a reader, while active sensors have their own power source; sensors have different sizes, from nanometres upwards; sensors can be in a fixed location or mobile or can be attached to a physical object -human or animal- which can carry them.

- ***Functional complexity:*** some sensors provide a simple functionality by converting a physical phenomenon into data and others have the ability to automatically detect pre-defined events. Sensors can be reconfigurable, self-configurable, and self-optimizing. Multiple sensors can be used to combine the collective data, e.g., more than one fire detection sensor is used in a building to detect fire. An event can be restricted to some predefined threshold value or a period of time to be covered, e.g., after increasing the temperature to a certain degree, a temperature changing sensor must fire an event.

Actuators

Actuators act in response to a control signal to convert energy (e.g. electrical, hydraulic pressure) to move or control a mechanism such as drilling, locking, rotating, or switching. They are usually attached to some objects like a door, fan, lamp, etc. Many actuators have been developed to remotely control an inhabitant's environment through wireless networks. Figure 2.1(b) demonstrates some examples including:

- *Wireless light actuator* used to remotely turn a light or home appliance off or on.
- *Fan controller* for heating and cooling applications in order to switch multilevel fans or to turn heating and cooling on/off.
- *Wireless valve actuator* to control a liquid's flow, like water or fuel. They can be equipped with an electric motor to open and close the valve.
- *Linear actuator* to produce motion in a straight line. They may be mounted on doors and windows to open and close them remotely, or under a table to change its height.

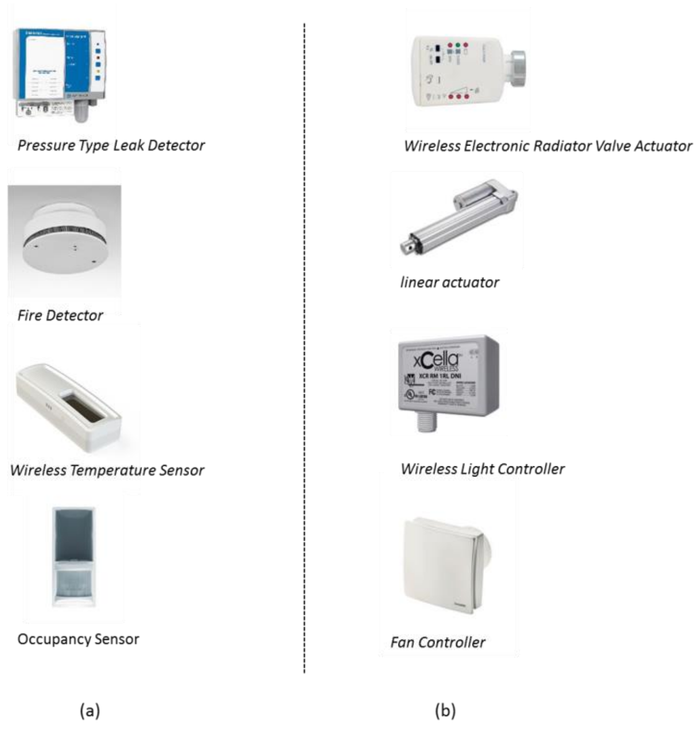

Pressure Type Leak Detector
Fire Detector
Wireless Temperature Sensor
Occupancy Sensor
Wireless Electronic Radiator Valve Actuator
linear actuator
Wireless Light Controller
Fan Controller
(a)
(b)

Figure 2.1: Examples of industrial sensors and actuators (taken from [23])

2.3 Networking Physical Components

Networking smart entities (also referred to as Networking-Appliances - NAs) are not limited to personal digital assistants, web pads or mobile phones, but include many other devices like NAs-navigation systems, heater controllers, light adjusters, fridge, air conditioners, fire detectors or smoke alarms attached to a wall in a room. NAs have already been used in a variety of domains and will emerge in others in the near future.

Integrating these physical components into a distributed network can help them to interoperate and share data. Such tasks include, for example, home automation (e.g. motion sensors could co-operate with a lighting system to control it), security systems (e.g. fire and infraction sensors could co-operate with alarm systems), transport and logistics facilities (e.g. travellers could be informed about their schedules, goods remotely tracked), and healthcare systems (e.g. assisting elderly people, person aware systems).

To enable these activities, recent technologies in the domain of embedded devices have created a number of low power physical and transport network protocols such as Zigbee (over IEEE 802.15.4) [3], IPSO [24], 6LoWPAN [4], and Bluetooth (over 802.15.1) [25], to connect and control devices and appliances in the networks [19].

Many types of network are becoming available and they are distinguished according to their infrastructure, network, and frequency ranges.

- ***Wireless Networks:*** Wireless LANs (also referred to as WLANs) are local area wireless networks based on the IEEE 802.1l set of ratified standards that support different frequency ranges and message transfer rates up to 54 Mbps. In traditional WLAN networks, computers and devices are connected to the network via WLAN cards, which are connected to the Internet via an access node.
- ***Power Line Communication (PLC)*** is a wired network that shares network electrical energy as a channel to send data, audio and video information. For instance, it can be used in the home to control lighting and appliances remotely rather than using wiring control systems. Home-Plug and X10 Alliances specify various power line communication standards [26, 27] to support networking over electrical networks. Electric buses, trains, and trams are other typical examples of networks using electricity which could be used as data networks.
- ***Body Area Network (BAN)*** is used to connect computers to devices or sensors that are wearable or implanted in the human body. BAN applications are often used to store and monitor data related to human bodily health and performance [28].
- ***The Personal Area Network (PAN)*** specified by the IEEE 802.15 working group is a specific type (subset) of local area networks (LAN) designed for one person rather than a group. It enables computers to be connected to the NAs close to the user. It typically involves mobile phones, personal digital assistants (PDA), headphones, microphones,

laptop PCs, medical sensors, body area sensors, cameras, and other devices which provide facilities for the individual. For example, a person working with a laptop could interconnect with wearable devices to exchange human body performance using wireless technology. Protocol like Bluetooth and Zigbee can be used to implement a PAN [29].

2.4 Decision Making Algorithms

Enabling physical components to co-operate and share data is not enough to make the environment reactive and smart. Physical components acquire only limited information and have a partial view of their environment due to their insufficient resources or inability to access global knowledge. By taking advantage of information exchanges between devices and with the help of an intelligent software algorithm existing somewhere (e.g. centralised computing unit, computing agent), it is possible for intelligent software unit(s) to perform decision-making tasks, either autonomously or through human intervention, and in consequence, make the environment reactive and smart. The decision making process depends on collecting the necessary information about the environment, governing dynamic actions, human interactions to make operationally strategic decisions in order to achieve a specific goal.

The decision making task is defined as "*the process of determining the action that should be taken by the system in the given situation in order to optimize a given performance metric*" [2]. Within smart environments, this course of action can take multiple forms as shown in Figure 2.2. It can be a *Single Decision* based on a limited set of information. Another type of decision involves a *Feedback Control System* that adjusts to a particular circumstance in order to accomplish and maintain certain conditions for a particular component. An example of such a component is a heating control system where the room temperature is to be held at a particular level.

The third kind of decision problem common in smart environments is a sequence of actions that need to be performed in order to achieve the final goal such as home security, inhabitant comfort, or energy saving. In contrast to the preceding two decision types, this kind of decision is the most complex and requires co-ordination between the smart entities populating the environment. For example, in a scenario where an occupant leaves home, the decision maker could determine that it should turn off all the lights and reduce the temperature of the heating system. To obtain the optimal combination of actions to be performed in a given scenario, a decision algorithm is run by the software system at regular intervals to determine what action to perform, whether (time-driven) or is fired only when an event is produced by the occupant's action or a change in the sensors' values (event-driven) [2].

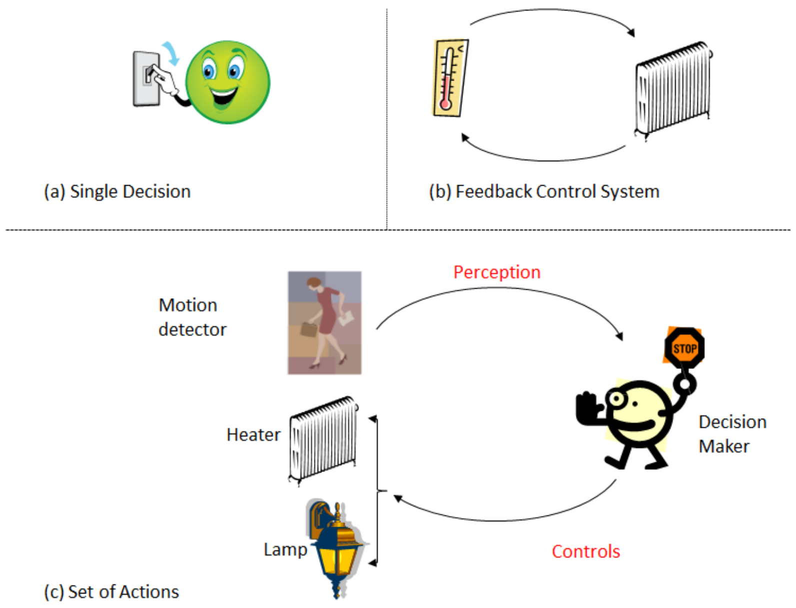


Figure 2.2: Decision making complication

The decision-making properties are classified into different groups: co-ordination interactions, human interactions, and programming techniques. A brief overview follows of each group.

2.4.1 Co-ordination Interactions

There are two types of decision co-ordination.

- ***Orchestration*** *(use of a central coordinator).* The different partners (smart objects, services) are under the control of a single endpoint central unit called a process. The latter controls the logic of the execution and the interaction with other partners.
- ***Choreography*** *(use of distributed coordinators).* In contrast to orchestration, choreography does not depend on a centralized unit but on the distributed and cooperative components (participators). Each participator has to know exactly when to become active and with which component to interoperate.

2.4.2 Human Interactions

Smart environments are often designed to minimize human/device interactions. Human interactions can be classified into three main kinds: automatic, semi-automatic, or manual, according to the level the user interacts with the system. Below is a brief definition of each kind.

- ***Automatic*** *(autonomous)*: "*autonomous systems are defined as systems that are self-governing and are capable of making their own independent decisions and carrying out actions*" [19]. The user only specifies the high-level goals and tasks rather than defining and controlling each low-level task interaction. The system or the agent itself automatically determines what the necessary decisions and actions are and schedules them automatically to achieve the final goal. A software multi-agent system is often characterized as an autonomous system.

- ***Semi-automatic*** *(partially automatic)*: the aim of this type is to reduce and minimize user tasks. The system usually helps the user to find, filter, integrate, and automatically make the desired resources interact, while the programmer's job is to define the control workflow or predefined plan. In addition, end-users can interact with the system during execution.
- ***Entirely human-based (***manual*)*: the user programs and tells the system which operations and decisions to perform, which resources to use, or which actions to take during the development and execution steps. The user defines and controls each low-level task interaction.

2.4.3 Decision Algorithm Techniques (Programming Techniques)

Many programming style techniques are used for carrying out decision-making algorithms (see Subsection 3.5.1). Some of these are:

- ***Planning-based techniques*** take a programming approach that defines a problem, rather than giving a solution, by describing the computational logic without saying how to compute. It permits the system to obtain autonomously the best combination of actions based on the environment description and the effects of the available actions. Finally, the system generates a plan describing the set of actions to be taken to achieve the desired goal.
- ***Rule-Based techniques*** mean decisions are taken by using a set of facts and rules based on the state of the environment and the occupants, together with a set of condition-action subroutines written in a declarative logical programming language. A rule is executed if all the conditions are evaluated as necessary to fulfil. For instance, when a rule encodes if the occupants leave the home, turn off the heater, this leads to a control system turning off the heater each time an occupant leaves the home. This programming style usually implies an automatic approach.
- ***Learning-based techniques***: learning algorithms aim to enable decision makers to improve their performance through training and experience in order to make decisions that are similar to those defined in a training set. This permits the system to apply what has been learned from experience to any new situation and so reduces the time required to achieve the goal.
- ***Workflow-based techniques*** style is generally used when the programmer needs to tell the control system what to do during the execution steps. It follows an imperative procedural

programming style. The workflow defines a sequence of commands for the system to perform a composition of several activities (e.g. interaction with partners) linked to control flow (loops, if, etc.), and execution constructs (computing in sequence or in parallel, etc.). This programming style is usually adopted in manual and semi-automatic interaction approaches.

2.5 Example Domains of Smart Environments

Smart environments can be any environment where a network is available, and smart physical components are connected. Some examples illustrate what could be considered as smart environments.

2.5.1 Smart Residential Building Environment

A smart residential building (e.g. home, school, hospital) is one populated with diverse networked devices and smart objects such as alarms, a washing machine, home heating system, air-conditioners, cameras, door controllers, home-environment sensors, etc. In a future smart building, data from these smart devices and appliances can be easily exchanged to provide a range of smart building services to make the inhabitants' lives easier, cheaper, and more comfortable. These services include energy and safety management, security monitoring, entertainment, health care, and support for the elderly, and other services. Figure 2.3 shows some examples of smart objects and examples of their utility:

- *A Smart Room* could contain a diverse range of sensors in order to measure or detect any change in the state of the environment (e.g. temperature, motion, and humidity). These sensors may include temperature sensors, motion detectors, or fire detectors.
- *A Smart Fridge* is equipped with a set of sensors and scanners that allow its user to view and monitor the current stock of food items without opening the door, to check for expired food dates and it alerts users when food items are low so they can prepare shopping lists, or can suggest possible recipes based on what is stored in the fridge.
- *Smart Door/Window/Curtain* are attached to controller devices (actuators) in order to be opened/closed or locked/unlocked.
- *Smart HVAC (Heater/Ventilation/Air Conditioner)* allow users to personalize their air conditioner and heater systems remotely via a network in order to control temperature, humidity or switch them on or off.
- *Smart Lamps* contain actuators, which enable them to turn the light on and off or dim to control brightness.
- *Smart mirrors* contain a controller to optimize the field of vision.

- *A smart pillow* contains sensors to detect inhabitant behaviour. It can be connected to smart systems which can read a book or play music at bedtime and turn off when the inhabitant goes to sleep.
- *A smart bed* can remember your favourite music, your preferred light, and temperature settings to gently wake you up and give you a smooth start to the day.
- *Smart clocks* can provide context information such as where you are, or what the weather is like at a specific moment in time.
- *Smart chairs* are equipped with a number of sensors to take information about a sitter's behaviour and, a number of mechanical motors so the chair can adapt to them.
- *Smart Washer and Dryer Machines* are equipped with a number of sensors and actuators that can be controlled remotely from a smart application. For example, controlling their start time, notifying the status of the washing and drying, or sending a message when the cycle is complete.

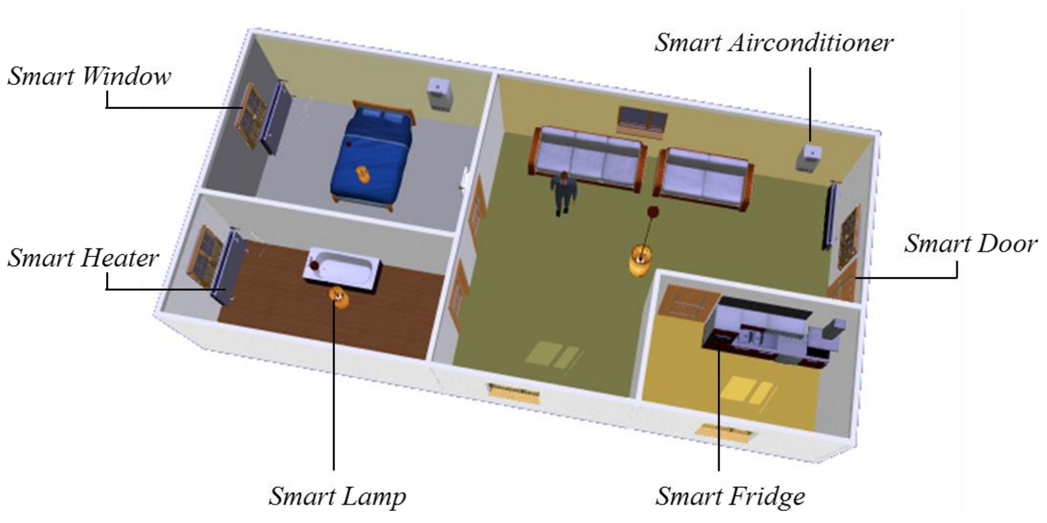
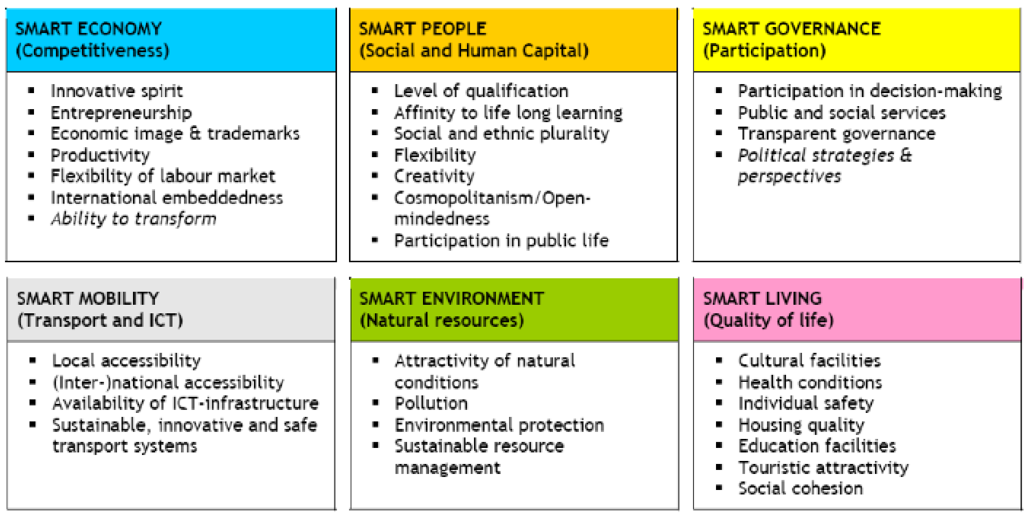


Figure 2.3: Smart building

2.5.2 Smart Cities and Public Environments

Different smart equipment (often also referred to as street furniture) can be installed in streets, parks, parking, and in commercial buildings, and includes such things as public benches and bollards, traffic lights, toilets, sensors nodes to measure pollution, traffic, or noise, to create what are called smart cities. In Figure 2.4, a smart city is classified within six areas: Smart Economy, Smart People, Smart Governance, Smart Mobility, Smart Environment, and Smart Living. These six concepts constitute a forward step in identifying different characteristics, factors and domains which describe areas of a smart city [21].



SMART ECONOMY
(Competitiveness)

- Innovative spirit
- Entrepreneurship
- Economic image & trademarks
- Productivity
- Flexibility of labour market
- International embeddedness
- *Ability to transform*

SMART PEOPLE
(Social and Human Capital)

- Level of qualification
- Affinity to life long learning
- Social and ethnic plurality
- Flexibility
- Creativity
- Cosmopolitanism/Open-mindedness
- Participation in public life

SMART GOVERNANCE
(Participation)

- Participation in decision-making
- Public and social services
- Transparent governance
- *Political strategies & perspectives*

SMART MOBILITY
(Transport and ICT)

- Local accessibility
- (Inter-)national accessibility
- Availability of ICT-infrastructure
- Sustainable, innovative and safe transport systems

SMART ENVIRONMENT
(Natural resources)

- Attractivity of natural conditions
- Pollution
- Environmental protection
- Sustainable resource management

SMART LIVING
(Quality of life)

- Cultural facilities
- Health conditions
- Individual safety
- Housing quality
- Education facilities
- Touristic attractivity
- Social cohesion

Figure 2.4: Smart city dimensions (taken from [21, 22])

In [22], Luis et al. gathered some domain examples that could increase the importance of smart cities of the future:

1. *Developing E-Government Services* to enable citizens to communicate and interact with public administration services via the internet using smart applications, or have video conversations in an efficient and cost effective manner rather than using the traditional boring methods like the mailing services or being present in person.
2. *Health Inclusion and Assisted Living Services* can offer a good quality of life for elderly people, help people to monitor their health at home or to be connected with a hospital if they become ill.
3. *Intelligent Transportation Services* would mean vehicles could sink and raise bollards remotely to guarantee secure access control of special roads and streets for buses. Traffic lights can be programmed in term of traffic density rather than a time scheduler.
4. *Energy Efficiency and Environment Issues* to lower energy consumption, reduce pollution and climate change are the main concerns and backbone issues in smart city environments. For example:
   - *Pollution alert systems* can monitor the level of pollution in each street of the city or warn if pollution increases above a certain level.
   - *Lighting* can be used to optimize the utilization of electricity by controlling remotely the lighting of parks and streets.
   - *Public services infrastructure* can detect water leaks and obtain a noise maps or send an alarm when rubbish bins are full.

However, many research challenges need to be addressed in order to achieve all the goals of smart cities. According to [22], these challenges can be grouped into two categories:

1. ***Privacy, Security and Trust Issues***
   - How will people identify themselves electronically via networks to applications and services providers in a standard and secure manner?
   - How can the security issues in complex and distributed systems be handled?
   - How can interoperability issues between identity management systems be solved?
   - How can diverse security techniques like encryption, access control, and intelligent data aggregation be offered?
2. ***Technical Requirements***
   - How can the complexity of the interaction and collaboration between systems, which are used for different purposes (e.g. transport control and energy management), be reduced?
   - Different operators often provide wireless and citywide networks technologies, so how can public and private networks be given equal access to both technical developments and regulatory changes?
   - There is a need for active integrated networks to link people, businesses, governments, and infrastructures.
   - How can the increasing volume of data collected be handled?
   - Many smart applications require personalized and location-based service infrastructures.

This thesis focuses on Smart Residential Building Environments (SRE) - defined in Section 2.5.1 - where security and privacy are considered a less complex issue and where a homogenous infrastructure and technical requirements can be achieved.

2.5.3 Smart Vehicle Environment

Smart vehicles constitute another example of a smart environment. Nowadays, vehicles are increasingly equipped with embedded smart systems to assist the vehicle operator and to improve and automate control operations such as parking assist systems, cruise speed control, traffic sign recognition, climate regulation, collision avoidance through automatic braking, etc. Some of these systems can be connected via the internet to provide more functions and utilities. Below are examples of smart equipment.

- *Smart doors, seats, and steering wheel* controllers can be embedded in doors and seats, and be unlocked remotely and automatically. The seat and steering wheel can be adjusted to suit the driver as (s)he sits down.
- *Location determination systems* enable the tracking of vehicles and goods remotely and inform of their delivery schedules.

- *Stations schedule service*s allow train, flight, and bus services to be connected to the internet or transport central to inform passengers of their arrival and departure schedules. Passengers can use their mobile devices to show their transport tickets or to buy them remotely.
- *A head-up display (HUD)* provides for drivers a transparent display that presents information without looking away from the road.

2.5.4 **Example Scenarios of Living in a Smart Environment**

Smart Environment where people live, visit and work could mark a powerful shift in people's life styles, creating a world where people are surrounded by smart things and a computing infrastructure supporting humans in everything they do. To imagine how people's lives would be, here are some possible scenarios.

1. **Daily Life Scenario**

In [2], Cook et al. describe an example scenario of a smart office, home, and vehicle.

> Dave prepares to leave the office a little earlier than usual, since he managed to get some work wrapped up more quickly than expected. As he heads out of the door, his pc automatically locks and switches over the screensaver, the office lights dim, the air conditioner switches off, and the door locks shut behind him(PAN device as personal talisman integrated with PC and office environmental systems). As he walks out of the building his status on the corporate phone directory automatically changes to out (PAN locating tracking). Reaching his car, he hits a key on his phone. The door unlocks automatically and the engine starts. The seat and steering wheel adjust to suit him as he sits down (his wife gave him a lift to work this morning, so the seat was set for her). (Automobile and PA integration). He notes that he needs gasoline and should get it on the way home. He calls his wife to let her know he has set off for home; using the hands-free capability build into the car stereo and the phone that's still in his pocket (PAN dynamic integration with automobile). When he reaches the gas station , he gets out to use the pump, as he steps out of the car, he notices that his music collection is being updated using the local wireless fidelity (WiFi) that exists around the pumps(automobile dynamic network integration with WiFi). He wonders if they will send a software update for that irritating display bug in the car (dynamic software update). After he gets the gas and drives of (PAN talisman for payment applications), he arrives home to see the garage door rising just as he pulls around the corner. He parks the car and walks to the front door and a warm, pleasant house with the smell of dinner wafting from the kitchen. It started cooking before he left the office, and the central heating turned on when he was about a mile away from home (home environment and device control). His car, in the garage, syncs the music it got at the gas station with the other media devices in the house (automobile and home networking integration) and he settles down in front of the TV with his dinner).

2. **Comfort Scenario**

In [30] Sang Hyun et al. discuss range ideas for smart homes.

The smart wardrobe digitally looks up the weather forecast for the user so that they can comfortably and adequately coordinate what they wear with the outside environment before they leave the house. A smart bed can be programmed to remember your preferred sound, smell, light, and temperature settings to gently wake up all your senses and give you a good start every morning. A smart pillow can read any books of your choice to you at bedtime and can play your favourite music to drift off to when you start to get sleepy. Once your body goes into deep sleep, it will automatically check the condition and quality of your sleep, gradually reducing the volume of the music accordingly and, eventually, turning it off completely. A smart mat situated at the entrance of every home can be used to sense the body weight and footprint of the users, enabling the smart mat to perhaps differentiate and recognise who is stepping on the mat. A smart sofa can enhance your experience when watching the television or playing video games. Depending on the visuals and the sounds on the screen, it uses vibrations to enhance the viewing experience in action scenes.

3. **Energy Saving Scenario**

Jane controls her home environment in a way that allows her to save energy and to adapt her environment to her habits and living conditions. She knows that a well-planned energy control system can reduce the total energy consumption of the home. Experts and the experience of saving energy indicate that total energy consumption could be reduced in the following ways:

- Reducing wastage: lighting, and air-conditioning (HVAC), and other systems can be turned off when not needed. When occupants leave the home, the system can turn all the lights off. When a room or level of the home is not being used, the lights can be turned off. Similarly, the heating for the swimming- pool or garage can be turned off when not in use or on when needed.

The energy saving scenario programmer has used the recommendations described above in order to create an energy control system for the final user, proposing the following:

The system knows what time of day it is, and when darkness occurs.

Case I: When the house's occupants are outside

- Turn off the lights.
- Turn off the air-conditioners (AC).
- Turn off the TVs and Radios.

Case II: When the house's occupants are at home

1. Daylight and lights hours
    - Turn off the lights in rooms, which are not occupied.
    - Turn off the AC if the outside temperature is less than x°. User selects which AC and the target temperature.
    - Set AC temperature. User determines which AC and what temperature.
    - Close windows if AC is turned on. User chooses which windows.

- Close curtains if AC is turned on. User defines which curtains.
- Turn off the TV if nobody has been watching it for 20 minutes.

2. Shower use
   - Stop shower water when use exceeds 10 minutes. User defines time limit.
   - Stop shower water heating during absence.
3. Dishwasher use
   - Turn the dishwasher on when it is ready during low energy cost hours.

2.6 Formalization

This section gives a formal definition of SRE and its meanings by clearly describing the smart physical components and by presenting their relationship with the actuators and sensors, they are attached to, together with the services that they might provide. This definition results in the proposed Ont4SRE ontology, which will be described in details in Section 4.4.

As mentioned in Definition 2.4, an entity is a real world object, location, or person. An entity is said to be smart when associated with a set of sensors and/or actuators. A residential environment is smart when it embodies smart entities. More formally, an entity, a smart entity, and a residential environment are defined below (see also [31]).

Definition 2.5 (Entity): An Entity is a 2-tuple $E = <c, P>$, where:

- c is the category of the Entity;
- P is a set of parameters.

The category is an object, a location, or a person. P has required and optional parameters. The required ones are {*entityID, entityName*}.The optional parameters depend on the category. At a given point in time, the parameter values give information about the entity. They represent the ***entity state***.

Definition 2.6 (Smart Entity): A Smart Entity is an entity with sensors and/or actuators. It is a 4-tuple $SE = < c, P, S, A>$, where:

- *c is the category;*
- *P is a set of parameters;*
- *S is a set of sensors;*
- *A is a set of actuators.*

Definition 2.6.1 (Sensor): A sensor makes available smart entity parameters at a given point in time. It might have associated publishers. It is a 3-tuple $S = <P, Q, U>$, where:

- *P* is a set of parameters;
- *Q* is a set of queries;
- *U* is a set of publishers.

P has required and optional parameters. The required ones are: {*sensorID, sensorName*}.

Definition 2.6.1.1 (Query): A query is a service provided by a sensor. In order to execute a query, a sensor gets the parameter values of the associated smart entity.

It is a 2-tuple $q= <Q_{name,} Q_{out} >$, where:

- Q_{name} is the query name;
- Q_{out} is a set of query result parameters.

Definition 2.6.1.2 (Publisher): A publisher produces a set of events.

Definition 2.6.1.2.1 (Event): An event occurs when the parameter value of a Smart Entity changes. An event is detected by a sensor and published by a publisher associated with this sensor. It is a 2-tuple $e= <E_{name,} E_{in} >$, where:

- E_{name} is the event name;
- E_{in} is a set of the data received from the event.

Definition 2.6.2 (Actuator): An actuator allows the modification of smart entity parameters at a given point in time. It is a 2-tuple $A = <P, C>$, where:

- *P* is a set of parameters;
- *C* is a set of actions.

P has required and optional parameters. The required ones are: {*actuatorID, actuatorName*}.

Definition 2.6.2.1 (Action): Actions are services provided by actuators. In order to execute an action, an actuator modifies the parameter values of the associated smart entity. It is a 2-tuple $Action= <Action_{name,} Action_{in} >$, where:

- $Action_{name}$ is the action name;
- $Action_{in}$ is a set of action input parameters.

Definition 2.7 (Smart Residential Environment): A Smart Residential Environment SRE is a set of smart entities coordinated by some kind of algorithm.

3 Smart Residential Environments: Technologies, Software Architecture

3.1 Introduction

Using pluggable devices like actuators with simple power-lines or remote controllers can help turn lamps, heaters, and other appliances on or off, but the majority of these devices are isolated, incompatible and provide a single function. As mentioned in Section 2.3, a number of low power physical and transport network protocols such as Zigbee, IPSO, and 6LoWPAN have been created to integrate physical components into the digital network. Furthermore, integrating the physical world into distributed networks has resulted in a number of paradigms such as context awareness [32], events management [33], prediction and decision making techniques [2]. This means, software architecture for smart components distributed over networks must contain these different elements.

In [34], Degeler et al. summarize some of the proposed smart environment architectures. The main components of SE Architectures are shown in Figure 3.1. Automation in a smart space can be achieved by perceiving, describing, and analysing the state of the environment, using the collected surrounding information to deduce the environment situation and take the suitable smart actions to change that state. Smart computing systems generally follow bottom-up design methodologies. Sensors monitor the environment by collecting information from physical components (*Physical Layer*) and then route this information via the *Communication Layer*. Physical and communications layers consist of hardware components including sensors, actuators, and the underlying low-level network protocols. Context and Event management components in the *Information Layer* collect the received information and transform it into high-level domain knowledge. Depending on the information collected and stored in the database (*knowledge Base*), the *Decision Maker* on the *Application Layer* takes and executes the appropriate actions in a top-down process. Actions are sent to the physical components via the communication layer to actuators in order to change the state of the physical components [35].

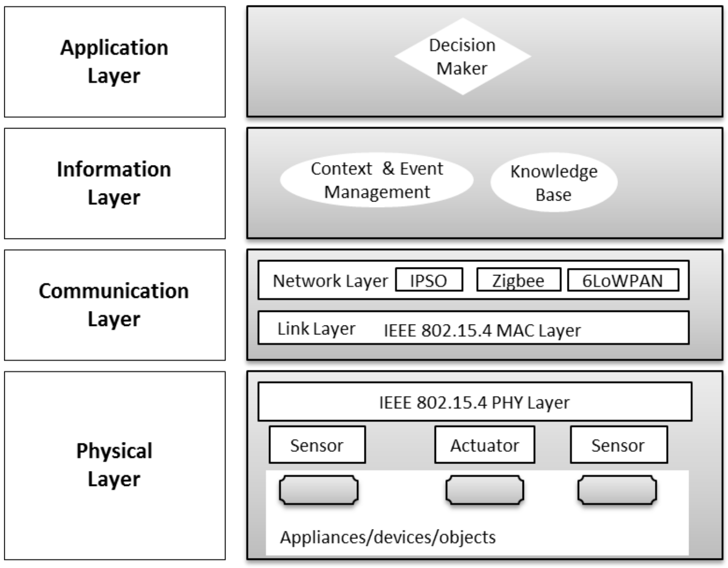


Figure 3.1: Basic smart environments architecture (inspired from [35])

Recently, some automation-building systems have been developed by commercial companies to use information technologies and microcontroller-based networks to control home appliances and features (e.g. heating, lamps, and doors). Some like KNX [36], BACnet [37], and EnOcean [23] implement their own exclusive protocols commonly based on the OSI model. They define their own data presentation system and how to structure and interpret the data inside telegrams to guarantee the interoperability between different sensors and actuators. Function blocks are used to describe device functionalities while complex control programs are implemented in a central unit.

However, from the smart environment control and physical components discovery point of view, hardware components still need more mechanisms to describe themselves and their functionalities in order to be identified (discovered) and used efficiently. Developing systems using their own set of low-level protocols and data models make coexistence hard to achieve and require users to have expert knowledge of each device platform. Several other challenges, which are still being explored, include heterogeneity, accessibility, invisibility, services discovery (fundability), and the ability to model inhabitant behaviour through appliance and object orchestration.

From these perspectives, middleware technologies and languages have been proposed on top of the communication and network layers such as those based on Service Oriented Architecture (SOA) [38], including Big web services (WS-*) [6, 7], RESTful [10, 11] and OSGi [13, 39] technologies. A middleware layer is additionally integrated to provide a service abstraction and an application-programming interface for those physical entities (see Figure 3.3).

3.2 Service Oriented Architecture for SE

Service oriented architecture has been proposed as a middleware solution for heterogeneous and changing environments like smart environments [38, 40-42]. The idea behind service oriented computing is to distribute services over the network and to make them available to clients using open standards. These services can be implemented with any language and can be discovered or invoked as well as composed to build complex scenarios. Applications interact with services through an interface endpoint and not at the implementation level. Thus, applications become more flexible due to their ability to interact with any implementation of a contract [43].

SOA Main Roles

The SOA has three major roles (see Figure 3.2).

- ***Service provider*** builds the service and offers it on the Internet for consumers;
- ***Service requestor*** invokes an existing service by opening a network connection and sending a request;
- ***Service registry & discovery*** is a central place where providers or developers publish new services or find existing ones.

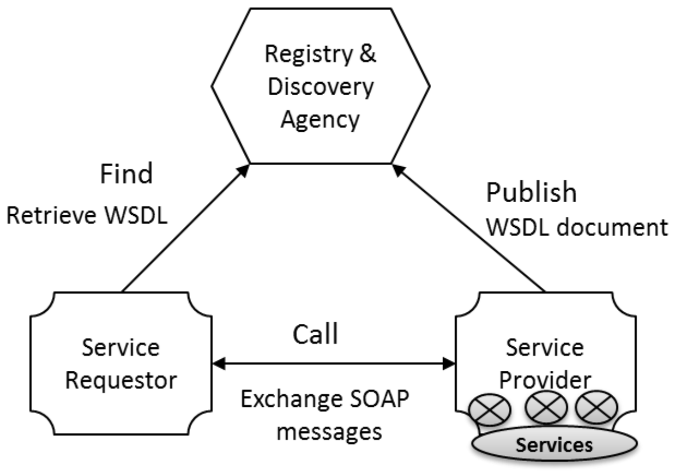


Figure 3.2: SOA principle roles (adapted from [40])

In the smart environment context, several approaches based on SOA have been introduced [5, 42, 44]. The architecture varies from one approach to another. Figure 3.3 illustrates a global SOA architecture upon which smart environments are generally based. The next sections explain the main additional layers: *Service* and *Discovery & Registry* as well as the services *Orchestration* and *Choreography* in the *Application Layer*.

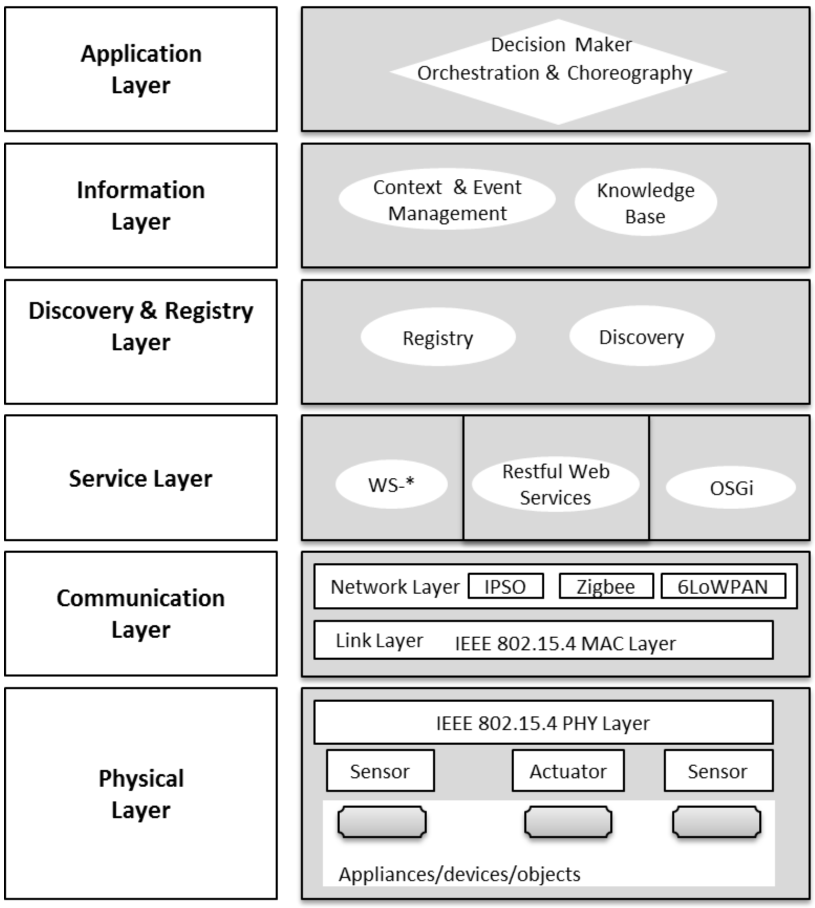


Figure 3.3: SOA-Based architecture for smart environment (inspired from [34])

3.3 Service Layer

Developing software systems using smart entities (appliances, objects) through their associated sensors or actuators where users have to manage the complexity of the underlying sources of functionality, protocols, and libraries is still a complex, error-prone task and needs expert knowledge of each embedded device platform. The goal of the *Service Layer* is to avoid users having to deal with these complexities. As a middleware layer, it provides a service abstraction and application-programming interface to the physical components, which facilitates and standardizes their access by applications and developers. Smart components expose their capabilities through web services. [43] gives a definition of what a web service is.

> **Definition 3.1:** *A web service is a self-describing, self-contained software module available via a network, such as the Internet, which completes tasks, solves problems, or conducts transactions on behalf of a user or application. Web services constitute a distributed computer infrastructure made up of many different interacting application modules trying to communicate over private or public networks (Including the Internet and WEB) to form virtually a single logical system.*

The client requests a web service often by sending a XML message and receiving a corresponding XML response. In this way, different operating systems and programming languages can talk to each other. For example, Java can talk with C# and Windows applications can talk with Linux applications [45].

There are three widely accepted standards for developing services for service-oriented paradigms- RESTful Web Services [46], the Open Service Gateway Initiative (OSGi) [12], and Big Web Services (WS-*) [47]. The remainder of this section gives a brief introduction to these three technologies, whilst other service-oriented technologies like UPnp and Jini are briefly introduced in Section 3.4.

3.3.1 RESTful Web Services

Representational State Transfer (REST) designates an architecture style of networked applications. The terms "representational state transfer", and "REST" were first introduced in 2000 in the doctoral thesis of Roy Fielding [48]. REST (also referred to as RESTful) uses a stateless, client-server, cacheable communications protocol which is almost always the HTTP [49] protocol. Its original feature uses HTTP as its application protocol to communicate and send messages between machines instead of using complex middlewares such as CORBA [50], and RPC [51]. RESTful web services are a lightweight alternative approach to the design of web services [52].

3.3.1.1 **RESTful & Resources**

The main feature of Restful web services technology is its representation of information elements as resources. Information exchanged is considered a set of resources identified by global identifiers URLs [49]. Restful applications use HTTP operations (CRUD) to create, update, read, and delete data. In order to interoperate with a resource, an application must possess both the resource's identifier and the required method. There is no need to know the service implementations or system configuration, i.e. whether there are caches, firewalls, gateways, or proxies between the application and the server which host the resources. However, the application must be capable of interpreting the data format (representation) returned from the resource, usually described in an HTML, JSON or XML document form, though it may also be an image, plain text, or any other content format [52, 53].

3.3.1.2 **RESTful Principle**

To summarise, RESTful architectural style is based on four principles as presented in [52, 53].

1. *Resource identification through URI.* Resources have URI identifiers used as global addresses for resource and services discovery.
2. *Uniform interface*. Four operations (create, read, update, delete) are responsible for the manipulation of resources. They correspond to the HTTP verbs (PUT, GET, POST, and DELETE).
3. *Self-description*. Resource content can be presented in several formats (e.g. XML, HTML, JSON, PDF, or JPEG). Metadata about the resource help find transmission errors and provide access control and authentication.
4. *Stateless interactions.* Interactions between resources are stateless.

The core advantage of a RESTful API is its flexibility. Various applications can be provided with a system's resources through data formatted in a XML document. However, RESTful present several limitations. According to [52, 53], unanswered questions remain for RESTful: (i) from the beginning, there is no common standard for the formal REST service description; (ii) RESTful has no common standards applicable for all web services functions, like Events and Notifications, Transactions, Security, Addressing, Trust and Orchestration.

3.3.1.3 **Web of Things (WoT)**

Internet of Things (IoT) [54, 55] and Web of Things (WoT) [52, 56] initiatives represent an example of bringing RESTful web services technologies into resources constrained devices by relying on IP addressing. The Internet of Things allows objects to communicate from a network point of view but it does not support the application layer, so things are still isolated and incompatible to understand each other. For this reason, the Web of Things has been suggested to overcome this limitation and to provide an application layer to help the creation of Internet of Things applications. In spite of this possibility, integrating the WoT patterns from developers’ viewpoints is still a rigid process and lacking framework architectures to facilitate their integration based on modularization and separation of concerns, so is still a problem as far as this thesis is concerned.

In this thesis, WS-* technology is adopted to control the smart environment using Event notification and orchestration strategies. The interested reader is referred to [10, 11, 46, 52] for a good introduction and a thorough discussion of RESTful technology and Web of Things since an in-depth analysis of this topic is outside the scope of this work.

3.3.2 **OSGi**

OSGi defines another standard that assures interoperability for applications and devices at the service level. OSGi is based on Java Virtual Machine (JVM) running on the device. The main part of the OSGi Service Platform specification is composed of so-called bundles. They are executed together in the same JVM. Each bundle has an API to access any other bundle function [57]. The OSGi solution provides high flexibility where devices and applications change over time. Unfortunately, OSGi-based applications need a java virtual machine in order to run. Additionally, the OSGi framework needs too much knowledge to design, build, implement, deploy, and maintain such service-oriented applications. Thus, most programmers do not take full advantage of the OSGi facilities. For further reading, review of OSGi techniques for ambient computing is given by [13, 58-60] as an in-depth presentation of OSGi is outside the scope of this work.

3.3.3 **Big Web Services (WS-*)**

The other standard is that of the Big Web Services (also referred to as WS-*). These are built using a specific programming language and published using a Web Service Description Language (WSDL) interface [61]. Web services are presented to clients as a set of operations that provide business logic on behalf of the provider. They must be deployed on a server container to be available for consumers.

The remainder of this subsection gives a brief introduction of WS-*-definition, characteristics, and standards summarised from [40, 43, 47, 61] where an in-depth discussion of WS-* is provided.

W3C working group [47] defines WS-* as follows.

> **Definition 3.2:** *A web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically*

conveyed using HTTP (Hypertext Transfer Protocol) with an XML serialization in conjunction with other Web-related standards.

3.3.3.1 **SOAP: Simple Object Access Protocol**

The messaging protocol currently used by WS-* is SOAP [62]. SOAP is designed to enable separate distributed computing platforms to interoperate. Figure 3.4 shows that SOAP messages can be built using different protocols such as HTTP to transport messages. SOAP defines how a message is formatted but not how the message is delivered. HTTP is the most commonly used transport protocol. However, other protocols, such as SMTP or FTP may be used. SOAP features include: simplicity, flexibility, firewall friendliness, platform neutrality as well as XML based messaging [43].

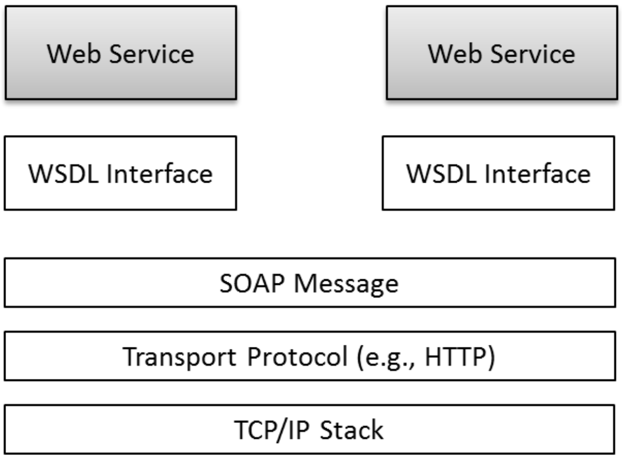


Figure 3.4: The web services communication and messaging network (taken from [43])

The structure of SOAP messages is relatively simple. It consists of a header and a body with some fault sections, all defined in an envelope, as shown in Listing 3.1.

- **`Envelope`** is a mandatory root element which defines the beginning and the end of the message. All elements of a SOAP envelope are defined using W3C XML schema.
- **`Header`** is an optional element to host additional features and functionalities, e. g., security, transactions, QoS attributes without modifying the specification.
- **`Body`** is a mandatory element which envelopes the message to be sent in XML format. This element holds the requested/response data or an error message (fault). Requested data can be XML data or parameters to a method call. Inside the SOAP body, the operation’s name and its related arguments are given.
- **`Fault`** is an optional element that provides information about errors that may occur while processing the message.

```xml
<Envelope xmlns="http://www.w3.org/2001/12/soap-envelope">
<Header>
  ...
</Header>
<Body>
  ...
  <Fault>
    ...
  </Fault>
</Body>
</Envelope>
```

Listing 3.1: SOAP message structure

3.3.3.2 **Web Services for Embedded Devices**

Integrating and adapting WS-* technology to embedded devices has been realized in many industrial and research processes [6, 63, 64] in spite of these devices limited computational memory, code space and communication bandwidth. There are two basic requirements.

(i) Using a TCP/IP protocol generally over a IEEE 802 (Ethernet) or IEEE 802.11 for WiFi network; and

(ii) Implementing an embedded web server which supports the HTTP protocol.

Further, many optimization techniques like in [8, 65-67] are used to implement web services on sensor nodes in order to be efficient in term of message, and memory size. Figure 3.5 illustrates a typical example of a software development kit developed by [68] which enables devices to host an embedded web server with web service framework over an HTTP protocol. This kind of kit is designed for a wide range of smart devices like thermostats air conditioners, sensors, etc.

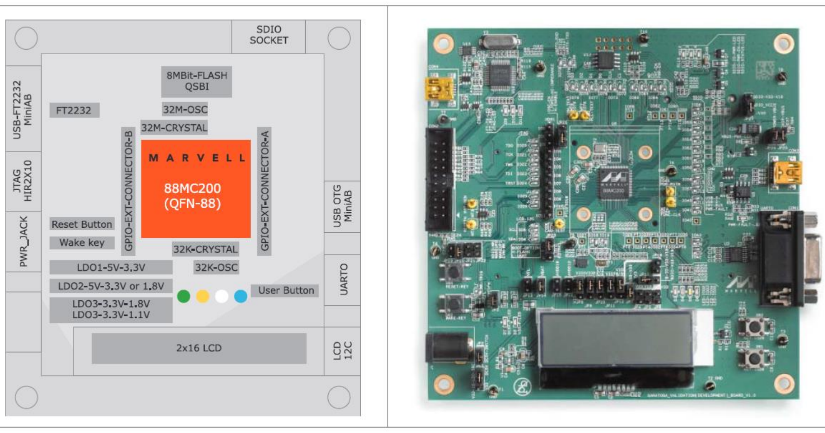


Figure 3.5: Marvell software development kit (taken from [68])

Device Profile for Web Service (DPWS) [7] and Web Services for Devices (WS4D) [6, 69] initiatives represent other examples of bringing WS-* technologies into resources constrained devices. It is built on the core of the WS-* standards, e.g., WSDL 1.1, XML Schema, SOAP

1.2 etc. and enables the minimum set of web service specifications to provide secure messages transmission, dynamic discovering capabilities, subscribing and receiving events to and from a web service.

3.3.3.3 Describing Web Services with WSDL

W3C in [61] give a definition of what a WSDL is.

> **Definition 3.3:** *WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.*

A WSDL document provides all the necessary information for a client to invoke the operations of a web service. According to [40] a WSDL description interface, as shown in Figure 3.6, has two parts.

1. *Abstract description (Interface Definition)*

   An abstract description portion describes the web service characteristics without any description or details concerning the technology used to implement it. It consists of several parts:

   - *Datatype* is used as a container to include all used data types;
   - *Messages* specify the payload of the incoming and outgoing messages; and
   - *Port type* includes methods (referred to as operations) names supported by the service and their signatures (input and output parameters).

2. *Concrete description (Implementation Part)*

   The concrete description portion of the WSDL file defines the connection to the real implementation of the web service. This description contains two parts: *Binding*, and *Service*. These define the Internet transport protocols and message formats used for each operation and the URL network address to access the web service implementation.

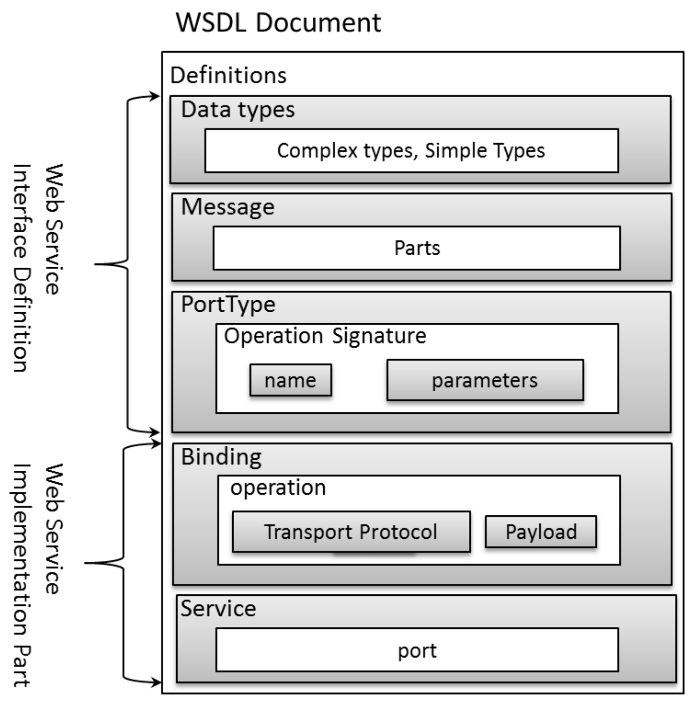


Figure 3.6: WSDL document structure (adapted from [40])

Example 3.1

Consider a smart object like an air conditioner coupled with an actuator to offer a web service named **`setTemperature`**, for adjusting a room's target temperature. The syntactical implementation of WSDL elements are shown in Listing 3.2 and defined according to [40] as follows:

- The **`definitions`** element encapsulates the entire WSDL document and usually contains several namespaces definitions used in the remainder of the document.
- The **`types`** element serves as a container for all the abstract data types. It can contain XML schema that define other data types. In this example, it is not used because there is a single data type.
- The **`message`** and **`part`** elements: The **`message`** element determines the payloads of messages, which are sent or received by a web service. Messages are composed of **`part`** elements. Each part stands for an instance of a particular type. Line 4 specifies a part as follows. **`<part name="parameter" type="xsd:float"/>`**.
- The **`portType`** and **`operations`** elements: The **`portType`** element represents several abstract operations with the outgoing and incoming messages while **`operations`** define the actions supported by the web service. There is one operation **`setTemperature`** declared (see line 10) as follows. **`<operation name="setTemperature">`**.
- The **`input`** and **`output`** elements: For every operation, there are optional input and output child elements. An operation uses inputs as method arguments and outputs for the returned messages (see lines 11-12).

- The **`binding`** element determines which communications protocol (e.g. SOAP protocol) can be used in order to invoke the web service (see lines 15-26).
- The **`service`** and **`port`** elements: The **`service`** element provides the physical address of the service. It contains the **`port`** element that defines the physical location. It defines where the service is located or to which network address the message has to be sent (see lines 27-31).

```
1    <definitions targetNamespace="http://ws.diuf/..." >
2    <types/>
3    <message name="setTemperature">
4       <part name="parameter" type="xsd:float"/>
5    </message>
6    <message name="setTemperatureResponse">
7       <part name="return" type="xsd:boolean"/>
8    </message>
9    <portType name="setTemperature">
10      <operation name="setTemperature">
11         <input wsam:Action=
             "http://ws.diuf/setTemperatureService/setTemperatureRequest"
              message="tns:setTemperature"/>
12         <output wsam:Action=
             "http://ws.diuf/setTemperatureService/setTemperatureResponse"
             message="tns:setTemperatureResponse"/>
13      </operation>
14   </portType>
15   <binding name="setTemperaturePortBinding"
              type="tns:setTemperatureService">
16      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" .../>
17      <operation name="setTemperature">
18         <soap:operation soapAction=""/>
19         <input>
20            <soap:body use="literal" namespace="http://ws.diuf/"/>
21         </input>
22         <output>
23            <soap:body use="literal" namespace="http://ws.diuf/"/>
24         </output>
25      </operation>
26   </binding>
27   <service name="setTemperature">
28       <port name="setTemperaturePort"
          binding="tns:setTemperaturePortBinding">
29          <soap:address location=
            "http://diufpc10.unifr.ch:8080/SmartHome3/setTemperature"/>
30       </port>
31   </service>
32   </definitions>
```

Listing 3.2: WSDL elements

3.3.3.4 Consuming the Web Service

The WSDL standard provides two main techniques for calling web services within HTTP.

- Using HTTP-POST: A SOAP message is sent to the web service as the body of an HTTP POST request. The message body contains the method name and its arguments. The Server decodes the XML string and extracts the input parameter values.

- Using HTTP-GET (URL encoding): All the required data and method names are passed in the URL string. For example, the **`setTemperature(20)`** method presented in the above example, can be encoded as **`http:/servername/setTemperature?parameter=20`**. It is similar to an HTML form when submitted with a GET request.

Example 3.2

Consider the above simple example to adjust the target temperature of an air conditioner using the HTTP-Post method. The end user uses a client application built with Visual Basic or java to control his appliances remotely by adjusting the target temperature. The steps for using the service are as follows.

1. The service provider (i.e. air conditioner actuator) makes its service available through a web service interface (WSDL document) shown in Listing 3.2, and deployed on an embedded server container.
2. The service requestor (client side) bundles the **`setTemperature`** request into a SOAP message, as shown in Listing 3.4.

```
<S:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header/>
  <S:Body>
       <ns2:setTemperature xmlns:ns2="http://ws.diuf/">
          <parameter>20.0</parameter>
       </ns2:setTemperature>
  </S:Body>
</S:Envelope>
```

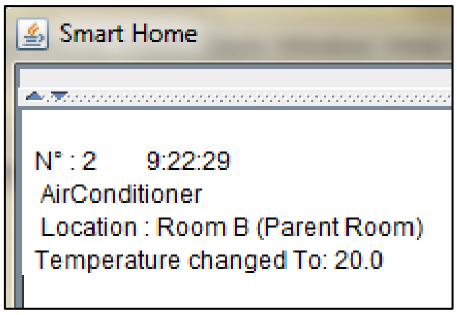
Listing 3.3: SOAP request message

3. The SOAP message is sent to the web service address as the body of an HTTP POST request.
4. Once the web service at the server end receives the request, it unpacks the SOAP request and converts it into a command that the program can interpret. The embedded application executes the command as required and responds with details concerning the execution result.
5. The web service packs up the appropriate response into another SOAP message and sends it back to the requestor, as shown in Listing 3.4.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Body>
        <ns2:setTemperatureResponse xmlns:ns2="http://ws.diuf/">
            <return>true</return>
        </ns2:setTemperatureResponse>
    </S:Body>
</S:Envelope>
```

Listing 3.4: SOAP response message

6. The service requestor unpacks the SOAP message to get the results (the return result of the called for method). In Figure 3.7, the client's application displays a confirmation message about the adjusted temperature.



N° : 2 9:22:29
AirConditioner
Location : Room B (Parent Room)
Temperature changed To: 20.0

Figure 3.7: A confirmation of web service invocation on behalf of the client side

3.3.3.5 Web Services Notification & Eventing

In smart environments, sensors and actuators are designed to host embedded web servers with web services over HTTP protocols. Usually, the applications and systems query and pull information from servers, but this is not appropriate if applications need to be notified immediately about any changes that might occur in the environment. Consider, for instance, the situation where part of a control workflow or a process is designed to be triggered based on events fired from a motion detection sensor attached in a given space. Rather than the workflow requesting constant updates from the sensor, where, in most cases, the requestor ends up with a no change response, it is better in practice if the event-based interaction pattern is applied, where the sensor takes responsibility for notifying the process asynchronously about any changes that may occur. In this way, HTTP calls (raising the scalability) and energy consumption are minimized [52]. Web Services Notification and Eventing mechanisms can be helpful for these needs. They are used to define the relationship between the service providers (e.g. actuators, sensors) and service requestors (e.g. processes, applications, software systems) using the publish/subscribe pattern (Event-Driven pattern). This involves an interested receiver (*Subscriber*) which is registered to consume the published information and a *Publisher*, which makes information available to all services subscribed for a specific topic. In this pattern, the sent message represents information produced about an event. WS-

Notification and Web Services Eventing (WS-Eventing) specifications are proposed to support event-driven patterns. Both specify a set of protocols, message formats and interfaces to enable event subscription and notification [40, 43].

In general, there are two methods for communication between subscriber and publisher.

- ***Interaction via Broker Service***. Subscribers and publishers communicate via an intermediary event manager called a *Broker*, which performs broadcast and subscription tasks (see Figure 3.8(a)). The advantage of a broker is to decouple the publisher from the subscriber (referred to as loosely coupled), allowing both to work independently and without having any knowledge of each other.
- ***Direct Interaction***. A subscriber interacts directly with the publisher (see Figure 3.8 (b)). When a new piece of information on a given topic is available, the publisher broadcasts this information to all the services interested in it. It supports a tight coupling between publishers and subscribers.

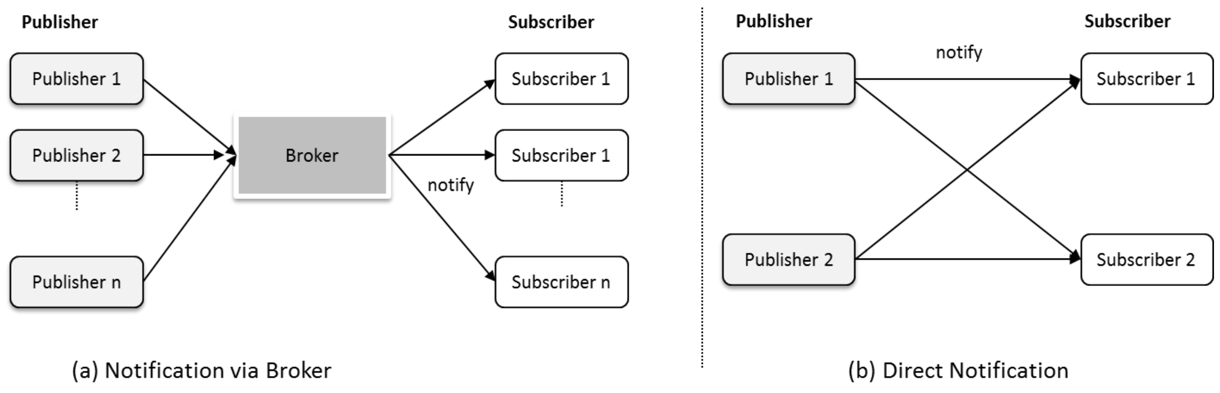


Figure 3.8: Web services notification methods (inspired from [40])

3.3.4 Semantic Web Services

Exploiting web services is difficult using exclusively the WSDL descriptions, since each service description lacks a description of the service properties and capabilities as well as its non-functional attributes, such as service name, type, or location. To be able to describe services, semantic web languages like the Ontology Language for Web services (OWL-S) [70], Semantic Annotations for WSDL (SAWSDL) [71] and Web Services Modelling Ontology (WSMO) [72] have been proposed. They introduce an additional level of abstraction where a declarative description of the meaning of the service's functionalities is given. This helps to distance the user from the syntactic description and thus provides semantic awareness and adds machine interpretable information to the service content, which facilitate service discovery and composition. The remainder of this subsection explains briefly the two main technology implementations for semantic web services: SAWSDL and OWL-S.

3.3.4.1 **Semantic Annotations for WSDL and XML Schema (SAWSDL)**

W3C in [71] defines SAWSDL as follows:

> **Definition 3.4:** *Semantic Annotations for WSDL and XML Schema (SAWSDL) defines how to add semantic annotations to various parts of a WSDL document such as input and output message structures, interfaces and operations.*

Semantic annotations in SAWSDL are used for mapping WSDL elements (operation, inputs, outputs, etc.,) to their corresponding semantic meanings and concepts.

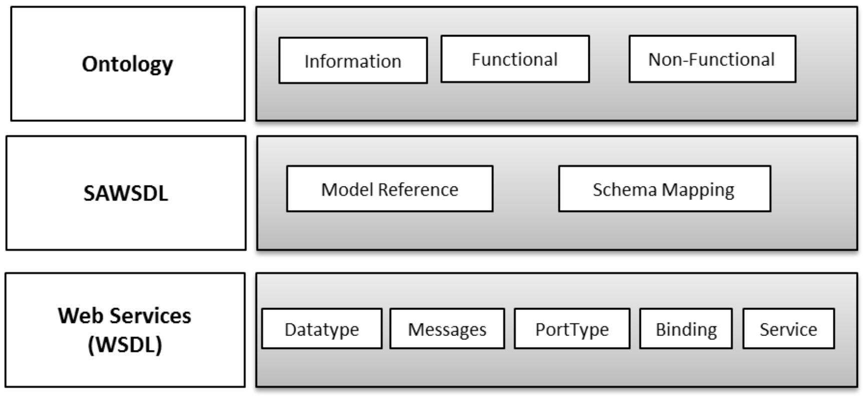


Figure 3.9: SAWSDL elements (adapted from [73])

The SAWSDL use two extensions (see Figure 3.9) to annotate WSDL operations and their parameters as defined in [73].

- *Model reference* is used to link WSDL elements to semantic concepts based on an ontology.
- *Schema Mapping* enables XML data to be transformed into and from its related semantic data (see Figure 3.10). It enables semantic clients to communicate with a web service. Mapping languages like XSLT can be used for mapping and SPARQL to query the ontology. Two elements are used for data mapping:
  - **`sawsdl:liftingSchemaMapping`** is used to map data from a web service XML message into its corresponding semantic elements.
  - **`sawsdl:loweringSchemaMapping`** is used to map data from the semantic elements into their corresponding web service XML message.

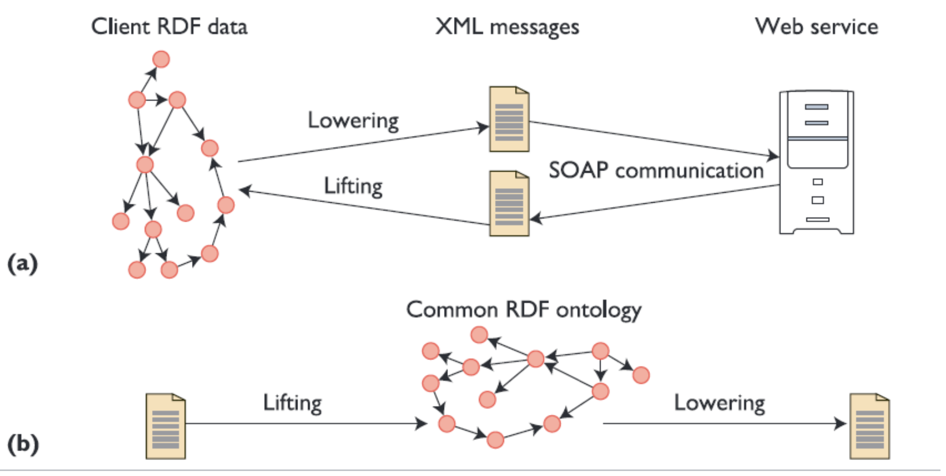


Figure 3.10: (a) Using Lifting and Lowering transformations and (b) XML data mediation (Taken from [73])

Example 3.3

Consider the previous adjusted target temperature example to illustrate how to use SAWSDL annotations and specifically how to use **`loweringSchemaMapping`** to call the **`setTemperature`** web service.

- Listing 3.5 represents the WSDL document with SAWSDL annotations.
  - Line 2 specifies the location of the **`loweringSchemaMapping`** document.
  - Line 3 specifies that the **`modelReference`** of the variable **`parameter`** is a **`temperature`**.
  - Line 7 specifies that the **`modelReference`** of the operation **`setTemperature`** is **`adjustTargetTemperature`**.

```
1   <definitions targetNamespace="http://ws/">
2     <message name="setTemperature"
                sawsdl:loweringSchemaMapping=
                "http://.../ont2adjustTargetTemperature.xslt">
3       <part name="parameter" type="xsd:float"
                      sawsdl:modelReference= "http://...#temperature"/>
4     </message>
5     ...
6     <portType name="setTemperature">
7       <operation name="setTemperature" sawsdl:modelReference=
        "http://...#adjustTargetTemperature">
8         <input  ... message="tns:setTemperatureRequest"/>
9         <output ... message="tns:setTemperatureResponse"/>
10      </operation>
11    </portType>
12    ...
13  </definitions>
```

Listing 3.5: Using modelReference for semantically annotating a WSDL web service

- Listing 3.6 represents the result in XML format of a SPARQL query on the ontology to retrieve the target temperature value, which should be sent as input parameter for **`setTemperature`** operation.

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
    <results>
        <result>
            <binding name="temperature">
                <literal>20</literal>
            </binding>
        </result>
    </results>
</sparql>
```

Listing 3.6: Result of applying SPARQL to ontology

- Listing 3.7 represents the XSLT **`loweringSchemaMapping`** to map data from Listing 3.6 into its corresponding web service request message.

```
<xsl:transform version="2.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:sp="http://www.w3.org/2005/sparql-results#"
 xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
 <xsl:output method="xml" version="1.0" encoding="UTF-8"/>
  <xsl:template match="sp:result">
   <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header></S:Header>
    <S:Body>
      <ns2:setTemperature xmlns:ns2="http://ws.diuf/">
        <parameter>
          <xsl:value-of select="sp:binding(@name='temperature')
                                              /sp:literal"/>
        </parameter>
      </ns2:setTemperature>
     </S:Body>
    </S:Envelope>
  </xsl:template>
</xsl:transform>
```

Listing 3.7: An XSLT sample that shows loweringSchemaMapping notion

- The result of executing the XSLT program is shown in Listing 3.8 As can be noted, the produced XML message corresponds to the **`setTemperature`** request message presented in Listing 3.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
    <S:Header/>
    <S:Body>
      <ns2:setTemperature xmlns:ns2="http://ws.diuf/">
            <parameter>20.0</parameter>
      </ns2:setTemperature>
    </S:Body>
</S:Envelope>
```

Listing 3.8: Result of applying XSLT from Listing 3.7 to SPARQL result of Listing 3.6

3.3.4.2 Semantic Mark-up for Web Services (OWL-S)

W3C in [70], defines a OWL-S as

> ***Definition 3.5:*** *OWL-S is the ontology, within the OWL-based framework of the Semantic Web, for describing Semantic web services. It can enable users and software agents to automatically discover, invoke, compose, and monitor Web resources offering services, under specified constraints.*

OWL-S is an expressive mark-up language build by the DARPA DAML program [74], to provide semantic annotations for declaring and describing web service content, capabilities and properties including the names of operation, input and output parameters, and possible other services parameters such as the quality of service (QoS), and security parameters.

OWL-S provides three kinds of semantic description to represent different aspects of a web service: *Service Profile*, *Service Model*, and *Service Grounding* (see Figure 3.11).

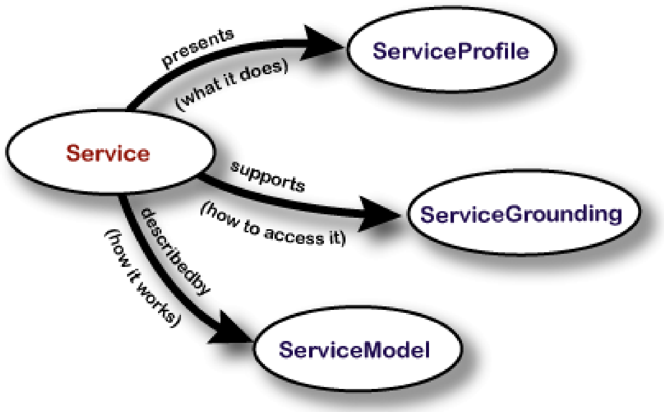


Figure 3.11: The top level of OWL-S ontology (taken from [70])

The following briefly describes the main elements of the OWL-S as defined in [70].

1. ***Service Profile Model*** describes the service features to other services or agents that want to use it (what the service provides). It defines the service inputs, outputs, effects, and precondition parameters. The service profile describes the service as a function of three basic types of information.

- The *Provider information* consists mainly of the **`serviceName`** and the **`textDescription`** (a brief description of the service);
- The *functional description* consists of the **`hasInput`**, **`hasOutput`**, **`hasPrecondition`** and **`hasResult`** constructs;
- The *properties description* allows for the description of a host of properties used to describe features of the service for example its category.

2. ***Service Model*** specifies how the service works. OWL-S services can be composed using a combination of atomic, simple, or composite services. An atomic process represents mainly a WSDL operation. It specifies the inputs, outputs, and effects and it can be consumed by the requestors in a single call.
3. ***Grounding Model*** defines how to interact with the service by providing the necessary concrete details related to the transport protocol, URL location of the service and message format. The Grounding model refers to specific elements within the WSDL specification by using a set of constructs such as the **`wsdlDocument`**, **`wsdlOperation`**, **`wsdlInput`**, etc.

Mapping Between OWL-S and WSDL

Both OWL-s and WSDL schema are designed to clearly link the semantic and implementation description of the web service. The two concepts could be naturally mapped from one to the other. Figure 3.12 shows how different elements should be mapped including:

- How WSDL operation can be transferred into OWL-S Atomic Process.
- How WSDL incoming and going messages can be transferred into OWL-S inputs/outputs.
- How WSDL binding element can be transferred into OWL-S grounding model.

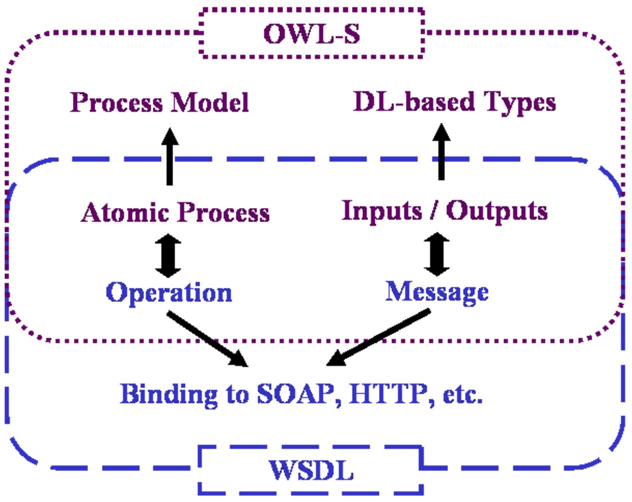


Figure 3.12: Mapping between WSDL and OWL-S (taken from [70])

Example 3.4

Consider the **`setTemperature`** web service example using OWL-S. Listing 3.9 shows the code of the OWL-S process equivalent to the **`setTemperature`** web service.

- Lines 4-23 define the service profile information as follows.
  - Lines 5-11 specify **`confirmation`** as the semantic output parameter.
  - Lines 12-18 specify **`temperature`** as the semantic input parameter.
  - Lines 19-21 specify that the semantic service name is **`adjustTargetTemperature`**.
  - Lines 25-30 specify the service model main body defining the atomic process **`adjust TargetTemperatureProcess`**.
- Lines 32-59 define the grounding model of the process, which maps the elements of the WSDL and OWL-S documents. It provides the necessary concrete details related to the transport protocol and messages format including:
  - Lines 33-40 map output element **`return`** from a web service XML message into its corresponding semantic element (**`confirmation`**).
  - Lines 41-48 map input element **`parameter`** from a web service XML message into its corresponding semantic element **`temperature`**.
  - Lines 49-51 refer to the WSDL URL address.
  - Lines 52-58 refer to the corresponding web service operation **`setTemperature`**.

```xml
<?xml version="1.0"?>
<rdf:RDF>
...
  <profile:Profile rdf:about="#adjustTargetTemperatureProfile">
    <profile:hasOutput>
      <process:Output rdf:ID="confirmation">
        <process:parameterType rdf:datatype="http://.../XMLSchema#anyURI">
          http://.../XMLSchema#boolean</process:parameterType>
        <rdfs:label>confirmation</rdfs:label>
      </process:Output>
    </profile:hasOutput>
    <profile:hasInput>
      <process:Input rdf:ID="temperature">
        <process:parameterType rdf:datatype="http://.../XMLSchema#anyURI">
          http://.../XMLSchema#float</process:parameterType>
        <rdfs:label>temperature</rdfs:label>
      </process:Input>
    </profile:hasInput>
    <profile:serviceName>
      adjustTargetTemperature
    </profile:serviceName>
    <service:presentedBy rdf:resource="#adjustTargetTemperatureService"/>
  </profile:Profile>
...
  <process:AtomicProcess rdf:about="#adjustTargetTemperatureProcess">
    <process:hasOutput rdf:resource="#confirmation"/>
    <process:hasInput rdf:resource="#temperature"/>
    <service:describes rdf:resource="#adjustTargetTemperatureService"/>
    <rdfs:label>adjustTargetTemperatureProcess</rdfs:label>
  </process:AtomicProcess>
...
<grounding:WsdlAtomicProcessGrounding
rdf:about="#adjustTargetTemperatureAtomicProcessGrounding">
  <grounding:wsdlOutput>
      <grounding:WsdlOutputMessageMap>
          <grounding:wsdlMessagePart>
           http://diufpc10:8080/smarthome/setTemperature?WSDL#return
          </grounding:wsdlMessagePart>
        <grounding:owlsParameter rdf:resource="#confirmation"/>
      </grounding:WsdlOutputMessageMap>
    </grounding:wsdlOutput>
    <grounding:wsdlInput>
      <grounding:WsdlInputMessageMap>
          <grounding:wsdlMessagePart>
            http://diufpc10:8080/smarthome/setTemperature?WSDL#parameter
          </grounding:wsdlMessagePart>
          <grounding:owlsParameter rdf:resource="#temperature"/>
      </grounding:WsdlInputMessageMap>
    </grounding:wsdlInput>
    <grounding:wsdlDocument rdf:datatype="http://...#anyURI">
       http://diufpc10:8080/smarthome/setTemperature?WSDL
    </grounding:wsdlDocument>
    <grounding:wsdlOperation>
      <grounding:WsdlOperationRef>
        <grounding:operation rdf:datatype="http://.../XMLSchema#anyURI">
          http://diufpc10:8080/smarthome/setTemperature#setTemperature
        </grounding:operation>
      </grounding:WsdlOperationRef>
    </grounding:wsdlOperation>
  </grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

Listing 3.9: A simplified OWL-S description of the setTemperature web service

3.4 Discovery and Registry Layer

Services discovery is defined as *“the process of finding available services and retrieving their service descriptions”* [43], while Registry is “*the process of publishing services to keep track of what services offer”* [43]. Registry and Discovery components are used as central places where providers or developers can publish new services or find existing ones. In a SOA concept, to discover services, consumers generally either use web services already known to them, or find new services by consulting a Services Discovery. Additionally, the discovered services and data mapping are often achieved manually. This is highly inconvenient, especially for complex composition scenarios. However, integrating the SOA paradigm into the design of SE has resulted in a number of different technologies for services discovery aiming to satisfy the specific requirements of the SE [75-78]. Some syntactic and semantic-based services’ discovery approaches are now introduced briefly focusing on their suitability for usage in SE.

3.4.1 Syntactic Discovery

The syntactic-based service discovery process focuses on operational and syntactic details for matching and describing different services. Many existing technologies use syntactic discovery such as Bluetooth [25], Service Location Protocol (SLP) [79], Jini [75], UPnP [80], UDDI [76], etc. The reminder of this subsection gives a short introduction to some of these technologies.

3.4.1.1 Jini

Jini is a distributed service-oriented architecture produced by Sun Microsystems. It is based on three Java language protocols: discovery, join, and lookup. The Jini framework provides administration facilities to discover the available services. One of the main parts of Jini infrastructures is the Jini Lookup Service (JLS), which stores dynamic information about the available services. Every service announces itself by publishing its information on the Jini Lookup Service.

Jini includes:

- **Clients:** They are the service consumers. Jini is used by clients, PDAs, mobiles, and PCs.
- **Jini infrastructure** needs a web server, Java, RMI invocation, and lookup service.
- **Lookup Service** is a central registry for services, allowing search and selection of services.
- **Proxy Object** is used as the mediator of the service.

- **Notification Method**: applications can subscribe to be notified about any change within the lookup services.

Unfortunately, Jini is based on JVM and RMI, which may not be installed in many devices. Service matching is performed by using simple matching schemata and at a syntactical level.

3.4.1.2 Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) is based on internet standards and technologies, such as XML, TCP/IP, SOAP and HTTP. It allows the discovery of the presence of physical components on the network and offers mechanisms to communicate and share data using services. UPnP implements the Simple Service Discovery Protocol (SSDP) to discover services on distributed networks. The main UPnp capabilities and functions are:

- SSDP enables a newly installed device to announce its services to the control point on the network. The exchange of messages between the device and the control point is in XML syntax and contains a description of the device like its URL location, services description (a list of commands and actions), etc.
- The control point is then able to send commands and actions to the device using an appropriate control message for each targeted service.
- UPnP also provides an event notification capability called General Event Notification Architecture (GENA).
- Devices can be controlled and monitored via a web browser.

UPnP is an independent platform with zero configurations and the dynamic discovery of devices and services. Unfortunately, UPnP is not frequently used due to a lack of management applications. Most industry solutions lack the tools for the simple configuration and control of gateways and devices. However, according to [81], the UPnP discovery protocol algorithms dramatically decrease with an increase of the services in the ubiquitous computing environment due to the impact of network congestion caused by multicast messages.

3.4.1.3 Universal Description, Discovery, and Integration (UDDI)

Universal Description Discovery and Integration (UDDI) is proposed as an open industry standard supported by the OASIS community. It is an independent platform which enables the business worldwide to register and discover web service applications on the internet. The UDDI directory helps services requestors to access the available WSDL documents by providing the required protocol bindings and message formats required to invoke them.

UDDI [76] supports three kinds of service description: white-pages, yellow-pages, and green-pages services.

- *White pages* contain information such as the business's name, text description (a list of multi-language text strings) and contact information (names, phone numbers, and web sites).

- *Yellow pages* provide business categories organized according to certain taxonomies such as industry (a six-digit code for classifying companies), products and services, and geographical (location of countries and region codes).
- *Green pages* contain the business information used to describe how other businesses can conduct electronic commerce with them. It is a nested model comprising business processes, service descriptions, and binding information.

Two kinds of APIs allow access to the UDDI Registry and Discovery: firstly, the enquiry API used to retrieve information from the Discovery and, secondly, the publishing API used to store information in the registry. Figure 3.13 summarizes the interaction between the different web services partners. With UDDI, the following information can be requested [43]:

- Discovery of web services interfaces.
- Discovery of services based on keywords.
- Discovery of web service providers.
- Determination of security and transport protocols for a given web service.

However, according to [82, 83] UDDI has some shortcomings, for instance, its keyword-based matches by service name or service classification, and UDDI supported syntactic interoperability, but it does not support any semantic description of its content.

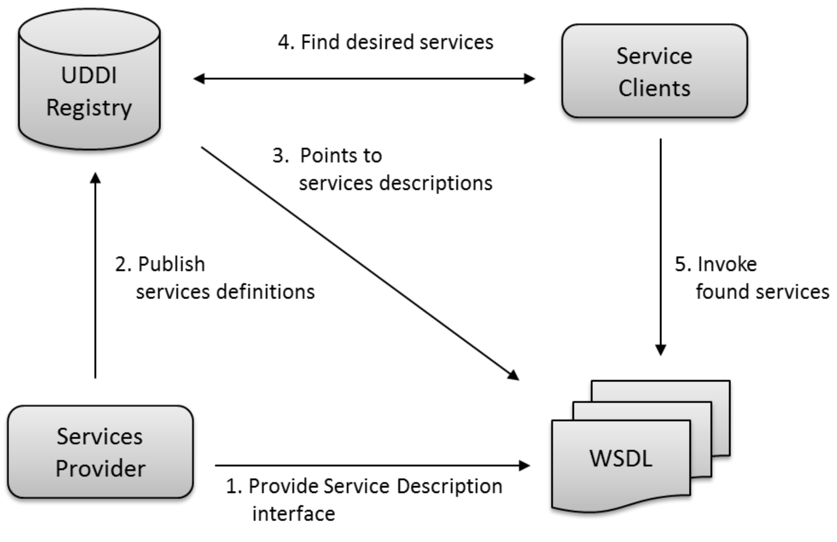

UDDI Registry
4. Find desired services
Service Clients
3. Points to services descriptions
2. Publish services definitions
5. Invoke found services
Services Provider
1. Provide Service Description interface
WSDL

Figure 3.13: Interaction between web services partners (adapted from [43])

3.4.2 Semantic-based Services Discovery

Service discovery technologies like UDDI, UPnP, and Jini focus on syntactic details for services discovery. This limits the discovery process to services with keyword-based searches. They enable explicit services to be found, only where the search can be made by keywords such as the service and provider name, location, or business category.

Recently, semantic web technologies have been proposed for providing semantic service descriptions by reference to ontologies using a formal language such as OWL. Adding semantic information to syntactical web service definitions can help interpret the purpose and usage of those services. Several semantic discovery algorithms have been proposed [77, 84, 85]. The main idea behind these approaches is to use semantic matchmaking mechanisms to discover and select the suitable services based on the comparison of service capabilities. The matchmaker component is the core of the discovery process. First, it retrieves the request and initiates the discovery algorithm. Then, the matchmaker executes the algorithm and returns a service list that makes a match between the underling semantic descriptions of the available web services and the client request, depending on the required degree of match. Figure 3.14 shows matchmaking between requested and candidate services. The candidate service, in general, has to fulfil the following conditions.

1. Candidate operation name should match the operation name specified in the request.
2. Candidate service inputs (name, type) should match the inputs specified in the request.
3. Candidate service outputs (name, type) should match the outputs specified in the request.

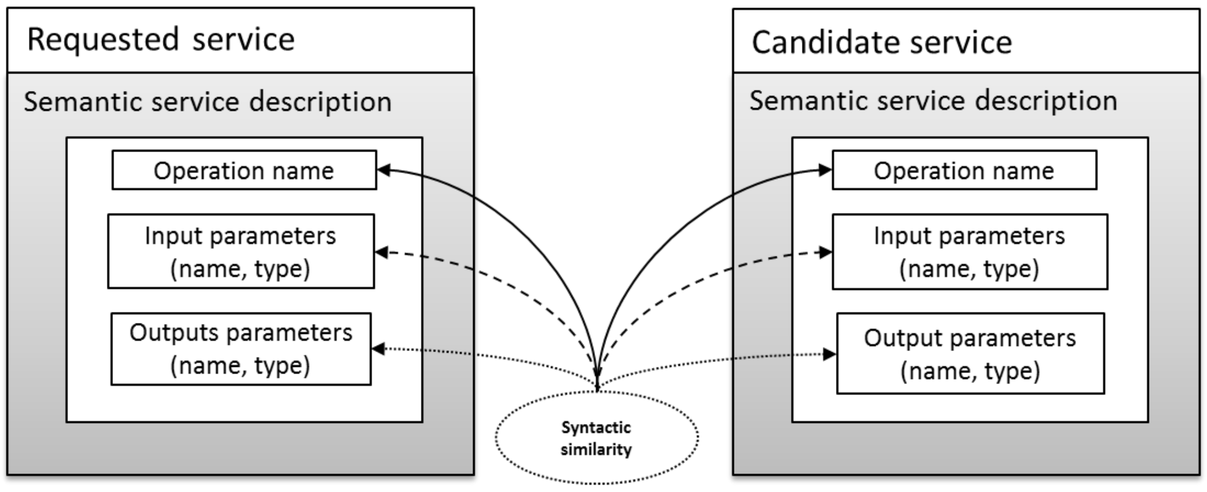

Requested service
Semantic service description
Operation name
Input parameters (name, type)
Outputs parameters (name, type)
Candidate service
Semantic service description
Operation name
Input parameters (name, type)
Output parameters (name, type)
Syntactic similarity

Figure 3.14: Semantic web services matchmaking

3.5 Application Layer

One advantage that distinguishes smart environment from environments with user control is the ability to model inhabitant behaviour. Smart models can be used to obtain a suitable decision by combining and reusing the available services in order to customize the environment and to reach specific goals such as automation, security, or energy saving. The SE and web services share the same services composition motivation to find a suitable solution by combining and reusing the available resources. Within SE, service composition is related to the decision-making process. SE is composed of smart entities coupled with networked devices and services that contain well-defined programming interfaces to enable the creation of composed and distributed applications, allowing users to interact with and control them. There are two ways to combine web services: through orchestration or choreography.

3.5.1 Orchestration

In orchestration, the web services involved are under the control of a single endpoint central unit (process). This process controls the logic of the execution and the interaction with other partners (web services). The partners and the process are loosely coupled. There are three main orchestration-programming techniques for the decision-making. *Automatic composition* (without human interaction), which enables the decision maker to autonomously take suitable actions using an internal model of the environment. *Manual composition* is commonly used in situations where the programmer has a well-defined process model (workflow). In a *semi-automatic or interactive composition*, a software system usually assists users with service selection, filtering, ranking, and integrating automatically. Whatever the technique used, it needs a programming language to express the service composition algorithms. Automatic techniques follow, in most cases, the declarative programming style using a declarative language like GDL4WSAC [86], PDDL [87] or SHOP2 [88]. With a manual composition technique, a number of tools and languages have been proposed by the software industry like BPEL4WS [18] and BPMN [89]. Table 3.1 gives a comparison of three approaches regarding their programming style, composition programming techniques, and programming languages.

Table 3.1: Comparison between composition programming methods

Theme	**Composition Approaches**		
	Automatic	**Manuel**	**Semi-Automatic**
Programming style	declarative	imperative	partially declarative
Programming techniques	AI-Planning, Rule-based, Machine learning	workflow	workflow
Programming languages	PDDL[87], GDL4WSAC [86], SHOP2 [88]	BPEL4WS, BPMN [90]	OWL-S [70], Script

3.5.1.1 BPEL4WS

In this thesis, workflow-based orchestration strategy is adopted to control the SRE, so it is important to introduce the basics of BEPL4WS.

BPEL4WS (also referred to as BPEL) is a web service workflow specification language developed by IBM, Microsoft, and BEA. BPEL4WS allows the behavior of web services in a business process interaction to be modeled. The specification has an XML-based grammar to implement and define the control logic needed to orchestrate the web services involved in the process flow. This implementation language can then be interpreted and executed by an orchestration engine such as Oracle SOA Suite [91], apache ODE [92], or jBPM [93]. A BPEL4WS process defines the order in which the appropriate web services are composed, either in sequence or in parallel. BPEL4WS allows the description of conditional activities. The invocation of a web service can, for example, rely on the results of another web service invocation. With BPEL4WS, it is possible to create loops, declare variables, copy, and assign values.

Example 3.5: The Structure of a BPEL4WS Process

To understand how business processes are described using BPEL4WS, the example of a simplified business process to control an air conditioner based on room temperature is taken again. Consider that a temperature sensor is able to publish an event about temperature changes. Each time the BPEL4WS process receives such an event, it checks if the temperature is greater than 30°; if yes, it switches the air conditioner on. Listing 3.10 shows a simplified view of the process used in this example.

- The **`process`** element is the root element of the BPEL4WS process definition. It has a name attribute and it is used to specify the definition related namespaces.
- The **`import`** element is used to import the WSDL document of the process.
- The **`PartnerLinks`** are used to identify the services participating in the process. It links the process with the air conditioner services (**`switchOn`**, **`switchOff`**).
- The **`Variable`** elements are used to declare the variables in order to receive, manipulate, and send data. Variables are defined in either WSDL message types, or XML schema types. In this example, there are three variables: **`inputVariable0`** to hold the received data of the event, **`Invoke1_OutputVariable`** to hold the return value of calling the **`switchOn`** service, and **`Invoke1_InputVariable`** to hold the data sent to the **`switchOn`** web service.
- The **`sequence`** element is used to declare the beginning of the main body of the process. It allows several activities to be defined that will be performed sequentially.
- The **`receive`** element is used to wait for the temperature sensor to notify the process about any change.
- The **`if-else`** element is a conditional statement used to control the program flow. There are other control constructs like **`while`** and **`forEach`**.
- The **`documentation`** element is used to insert comments.

- The **`invoke`** element is used to invoke the web service **`switchOn`**.

```xml
<?xml version = "1.0" encoding = "UTF-8"?>
<process name="BPELProcess1"
         targetNamespace="http://.../Project13/Project1/BPELProcess1"
         xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
         xmlns:client="http://.../Project13/Project1/BPELProcess1"
         xmlns:ora="http://schemas.oracle.com/xpath/extension"
         xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
         xmlns:bpws="http://.../wsbpel/2.0/process/executable"
         xmlns:ns1="http://.../events/edl/TemperatureEventDefinition"
         xmlns:ns2="http://xmlns.oracle.com/TemperatureEvent"
         xmlns:ns3="http://parent.room.ws/"
         xmlns:bpm="http://xmlns.oracle.com/bpmn20/extensions"
         xmlns:xdk="http://.../bpel/extension/xpath/function/xdk"
         xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap">
    <import namespace="http://.../Project13/Project1/BPELProcess1"
            location="BPELProcess1.wsdl" importType="http://.../wsdl/"/>
    <partnerLinks>
           <partnerLink name="PartnerLink1"
                  partnerLinkType="ns3:ParentAirconditioner_PL"
                  partnerRole="ParentAirconditioner_Role"/>
    </partnerLinks>
    <!-- VARIABLES List of messages and XML documents used by BPEL -->
    <variables>
       <!-- Reference to the message passed as input during initiation -->
        <variable name="inputVariable0" element="ns2:process"/>
        <variable name="Invoke1_InputVariable"
                  messageType="ns3:switchOnAC"/>
        <variable name="Invoke1_OutputVariable"
                  messageType="ns3:switchOnACResponse"/>
    </variables>
    <!--
      ORCHESTRATION LOGIC
      Set of activities coordinating the flow of messages across the
      services integrated within this business process
    -->
    <sequence name="main">
        <!-- Receive input from requestor.  -->
        <receive name="receiveInput0"
                 bpelx:eventName="ns1:TemperatureEvent"
                 variable="inputVariable0" createInstance="yes"/>
        <if name="If1">
            <documentation>if temperature &gt;30°</documentation>
            <condition>$inputVariable0/ns2:temperature&gt;30</condition>
            <invoke name="Invoke1" partnerLink="PartnerLink1"
                    portType="ns3:ParentAirconditioner"
                    operation="switchOnAC"
                    outputVariable="Invoke1_OutputVariable"
                    inputVariable="Invoke1_InputVariable"
                    bpelx:invokeAsDetail="no"/>
            <else>
                <documentation>do nothing</documentation>
                <empty name="Empty1"/>
            </else>
        </if>
    </sequence>
</process>
```

Listing 3.10: A simplified BPEL4WS process for air conditioner control

3.5.2 Choreography

Choreography, in contrast to orchestration, does not depend on centralized processes but on the distributed and co-operative components (participators). Each participator has to know exactly when to become active and with whom to interoperate. A multi-agent system is an example of applying the choreographic composition paradigm. It consists of self-governing entities named agents. These agents interact and cooperate to control different aspects of their environments. Ontologies are often used to add semantic descriptions to communication messages between agents. One of the main challenges in this approach is the distribution of shared resources. The problem is that agent decision makers can depend on resources that might already be in use by other agents. Service composition is done dynamically through the group's interactions. When a decision or action is required, the entities express it as a service invocation. The interested reader is referred to [43, 94] for a thorough discussion in choreography and to [17, 95, 96] specifically for multi-agents systems in pervasive computing environments, as an in-depth discussion is outside the scope of this work.

3.6 Ontology for a Semantic Web

The Ontology is one of the most used technologies for providing the vocabulary and describing the data knowledge structures, concepts, and constraints concerning a domain of interest. It allows the description of a set of hierarchical concepts and the relationships between those concepts. Ontologies enable people and machines to share a common view in a heterogeneous environment and makes information understandable, sharable, and reusable. Web Ontology Language (OWL) [97] is an example of an ontology that supports a semantic web. OWL is based on the Resource Description Framework (RDF) [98] and the RDF schema recognized by the W3C.

3.6.1 RDF

RDF is a data model similar to an entity-relationship. The core structure of RDF is a set of statements about resources. A resource can be anything- a physical object (e.g. vehicle, home), a service (e.g. goods, sales, booking service) and so on. The RDF database is stored in XML syntax as a set of statement expressions, each consisting of a *subject,* a *predicate* and an *object* (known as triples in RDF terminology). The *subject* represents a resource; the *predicate* specifies the relationship between the *subject* and the *object,* which can be a resource or a value like a string or a number. For example, "*Lamp has Location Room*" is a triple. The subject is "*Lamp*", the predicate is "*has Location*", and the object is "*Room*". The triples are identified via a uniform resource identifier (URI). Figure 3.15 shows a graph-based model to represent a statement.

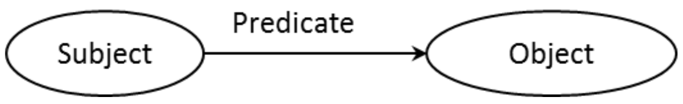


Figure 3.15: An RDF graph with Subject, Object nodes and a triple connector (Predicate)

3.6.2 OWL

OWL represents a vocabulary set to be used in RDF models. It defines the classes for the resources, and the properties that belong to each resource. The main OWL concepts are *Class*, *Instance*, *Object property*, *Data property*, *Domain*, and *Range*.

3.6.2.1 Class

A class represents a set of resources. A class is conventionally named using nouns. It contains a set of instances which represent real things and share the same properties. For example, in Figure 3.16, a class **`Room`** contains instances named **`room_1, room_2`** and so on. It is possible to add additional class information by including properties such as the **`subClassOf`** property to relate a class to its parent class. For example, the class **`Room`** is a subclass of class **`Location`**.

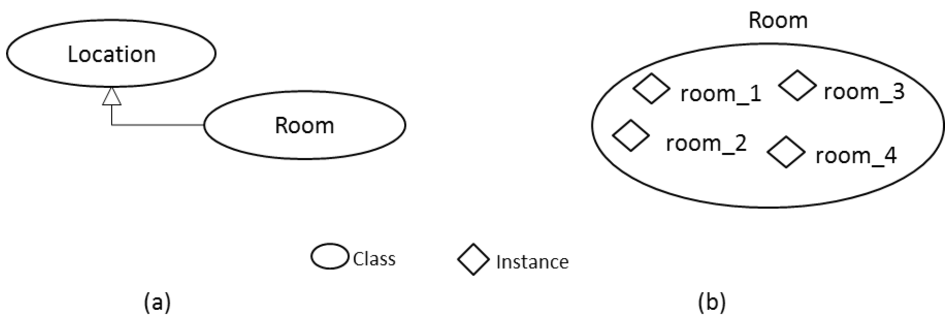


Figure 3.16: (a) Class presentation; (b) Representation of instances

Listing 3.11 shows an example of how OWL statements are written in RDF. It declares that class **`Room`** is a subclass of class **`Location`**.

```
<owl:Class rdf:about="http://www.unifr.ch/...HomeOntology#Room">
   <rdfs:subClassOf>
        <owl:Class rdf:about="http://.../HomeOntology.owl#Location"/>
   </rdfs:subClassOf>
</owl:Class>
```

Listing 3.11: A simplified OWL representation in RDF syntax

3.6.2.2 Object Property

An object property links a class instance with another class instance. With an object property, a statement consists of a domain (equivalent to the subject), a property (equivalent to the predicate), and a range (equivalent to the object). For example, in Figure 3.17 the object property **`hasLocation`** links the domain **`Lamp`** class with the range **`Room`** class.

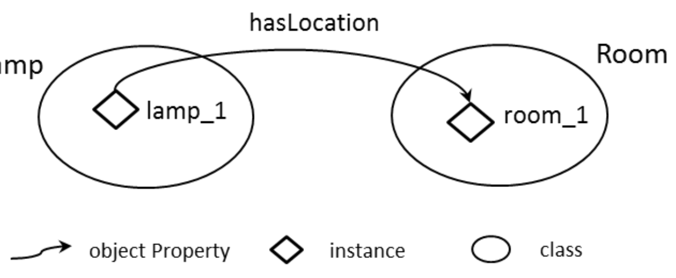


Figure 3.17: Representation of ObjectProperty

Listing 3.12 shows how the object property is declared in RDF

- Line 1 declares **`lamp_1`** is an instance of class **`lamp`**.
- Line 2 declares that the instance **`lamp_1`** has location **`room_1`**.

```
<diuf:Lamp rdf:about="http:///HomeOntology.owl#lamp_1">
  <diuf:hasLocation rdf:resource="http://.../HomeOntology.owl#room_1"/>
</diuf:Lamp>
```

Listing 3.12: A simplified ObjectProperty example in RDF syntax

An object property can be functional, inverse, transitive, and symmetric.

- **Functional Property***:* In a given instance; there can be -at most- one instance that is related to the instance via the property. For example, the **`hasLocation`** property is functional, which implies that each lamp instance must belong to one location.
- **Inverse property**: If a property links instance R1 to instance R2 then its inverse property will link instance R2 to instance R1, for example, if the **`hasEntity`** property is the inverse property of the **`hasLocation`** property. This implies if `lamp_1` has location **`room_1`**, then **`room_1`** has entity **`lamp_1`**.
- **Transitive property:** If the property P1 is transitive, and it links instance R1 to instance R2, and also instance R2 to instance R3, then it can be inferred that instance R1 is linked to instance R3 via property P1.
- **Symmetric property:** If a property P1 is symmetric, and the property links instance R1 to instance R2 then instance R2 is also linked to instance R1 via property P1.

3.6.2.3 **Data Type Property**

A data type property links class instance (domain) to data type values (range). Data type ranges can be limited to a single type (i.e. integer, float, or string) and can be restricted to a range of possible values (e.g. switch state could be a string value equal to “on” or “off”).

3.6.2.4 **Property Restrictions**

Restrictions specify the constraints on the properties (object properties or data type properties). There are two types of restrictions: type constraints (restrictions on type) and cardinality constraints.

- ***Type Constraints***: This kind of restriction restricts the type range that can be mapped to a domain via a given property. For example, the `hasLocation` property restricts all instances of an `Oven` class to `Kitchen` instances as location. There are three types in this restriction: *allValueFrom, someValueFrom* and *hasValue*.
- ***Cardinality Constraints*** restrict the number of values that an instance in the domain can have with a given property. For example, an actuator should have at least one service. There are three types of cardinality constraints: *Minimum Cardinality, Maximum Cardinality, and Cardinality Restriction* Constraints.

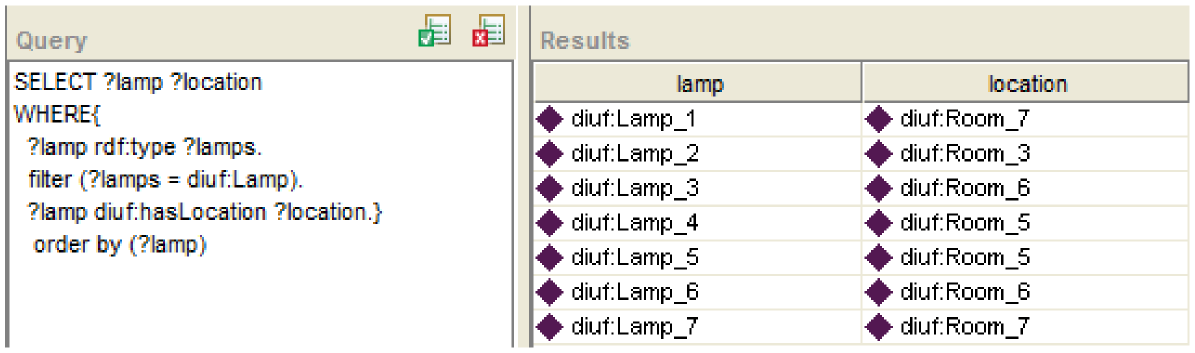
3.6.2.5 **SPARQL Protocol and RDF Query Language**

One of the main features of using an ontology like OWL is that it can retrieve information using queries. Query languages like SPARQL [99] can be used to query information from ontologies stored in Resource Description Framework (RDF) format. For example, ontology classes hierarchy can be retrieved or a check on whether a class is a subclass of another. Additionally, it is possible to query instances such as displaying a set of instances belonging to a given class which fit specific constraints.

Example 3.6

Consider a home consisting of several rooms, each containing some lamps. An ontology defining the classes `Room` and `Lamp` with their property relationships is created similar to Figure 3.17. A number of instances are instantiated for each class to get a list of all lamp instances in each room. Figure 3.18 shows how to use SPARQL query to get a response with the help of the Protégé [100] tool. The query consists of a `SELECT` clause used to retrieve lamp and room instances, and a `WHERE` clause, which includes a comparison predicate to eliminate from the result set rows which do not satisfy the following conditions:

1. `?lamp` must be an instance of class `Lamp`.
2. `?lamp` must be a lamp instance located in `?location` instance.



```
SELECT ?lamp ?location
WHERE{
  ?lamp rdf:type ?lamps.
  filter (?lamps = diuf:Lamp).
  ?lamp diuf:hasLocation ?location.}
   order by (?lamp)
```

lamp	location
◆ diuf:Lamp_1	◆ diuf:Room_7
◆ diuf:Lamp_2	◆ diuf:Room_3
◆ diuf:Lamp_3	◆ diuf:Room_6
◆ diuf:Lamp_4	◆ diuf:Room_5
◆ diuf:Lamp_5	◆ diuf:Room_5
◆ diuf:Lamp_6	◆ diuf:Room_6
◆ diuf:Lamp_7	◆ diuf:Room_7

Figure 3.18: Result of SPARQL query

3.7 Challenges of Managing SRE

So far, the technologies related to smart environments have been outlined. In the following section, the challenges of controlling smart residential environments through a case study realized in the field of a smart home is explored as a guiding thread for the remainder of this thesis.

3.7.1 Motivating Scenario

The final user, Jane, controls her home environment in a way that allows her to save energy and to adapt her environment to her habits and living conditions. Figure 3.19 describes Jane's home populated with a set of smart entities, including:

1. *Smart Room*: Each room (A, B, C) has two sensors. The first is a presence detection sensor which detects user presence and publishes an event when the associated state changes. The second is a temperature sensor which monitors the temperature and publishes an event when the associated state changes.
2. *Smart Entrance Door*: The home entrance door has an access control actuator, which provides open/close, and lock/unlock actions. The door also has an access sensor to detect if somebody enters or leaves the house, a lock sensor to detect if the door is locked or unlocked and another sensor to detect if the door is opened or closed.
3. *Smart Lamp*: Each lamp in rooms (A, B, and C) has an actuator to switch it on/off.
4. *Smart Heater*: Each heater in the rooms (A, B, and C) has an actuator to switch it on or off, and adjust the target temperature.
5. *Smart Shower*: The shower has an actuator to open and close the shower faucet. It also has a faucet sensor to detect whether it is open or closed, and to publish an event when the associated state changes.

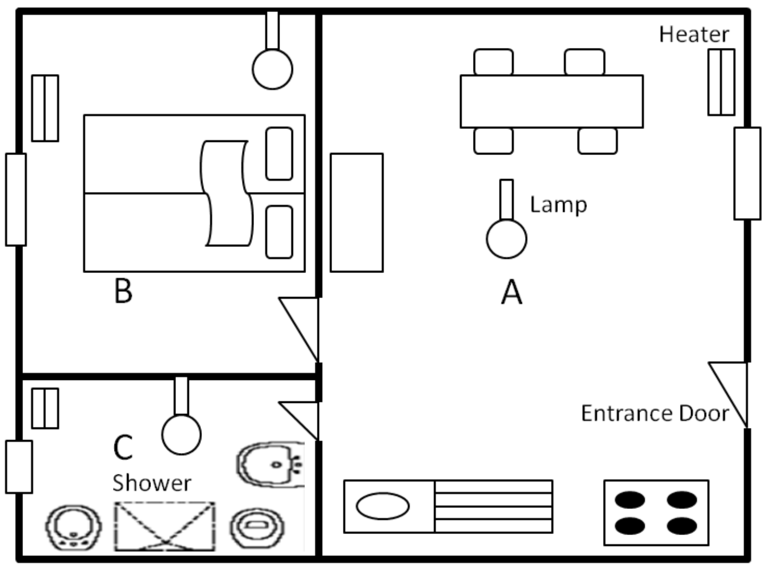


Figure 3.19: Smart home scenario

Table 3.2 gives an example of how a Home Control System (HCS) could use these smart entities to save energy (other possible examples would be: comfort, surveillance and access control, security, healthcare, etc.).

Table 3.2: Energy saving scenario

Time	Description
T1	The faucet sensor informs via its event publisher, the HCS when Jane starts her shower in Room C.
T2	The HCS triggers a timer to wait for 10 minutes.
T3	After 10 minutes, if the shower is still open, the system forces it to close.
T4	The presence detection sensor informs via its event publisher, the HCS that Jane has left Room B.
T5	The HCS waits for 3 minutes, and then switches off the lamp in Room B if it is on.
T6	The temperature sensor informs via its event publisher the HCS that there is a significant temperature change (i.e. >2°) in Room A.
T7	The HCS checks whether the current temperature in Room A is more than 18 and less than 22. If so, the system takes the appropriate action to bring the temperature within these intervals.
T8	The access sensor of the entrance door informs the HCS that Jane has left the home.
T9	The HCS configures the house in the best way to save energy taking into account Jane's stored preferences and that there is nobody at home.

3.7.2 Requirements and Challenges

The above scenario demonstrates the requirements to be satisfied in a home energy saving scenario. Below are examples of such requirements.

Modelling and Describing SRE

1. There is a need to define a generic model of the smart space in order to answer clearly questions such as what are the smart entities present in a home; what actuators and sensors they are attached to; what services (actions, queries and/or events) might they provide? Such a model could help programs to have a high degree of abstraction by using declarative elements and hiding several programming details. The smart space model should be a location-aware model. For example, when an event occurs, the control system often depends on the location of the event to determine the appropriate available services in the zone of space where suitable action is required.

Accessibility & Interoperability of Services in SRE

2. Although an additional service layer solves the heterogeneity and interoperability problems, the functionalities of the actuators and sensors could be implemented in different ways depending on the constructors. For example, to switch a lamp on, one constructor might provide an `on` operation, while another would call it `switchOn`. The question now is how to enable a standard description to operate the different services of the physical components of an SRE? Hardware components (actuators and sensors) need a mechanism to describe themselves and their services. Thus, there is a need for an ontology-based semantic description of services to help service discovery and composition by providing standardized and unified semantic descriptions.

Service Discovery in SRE

3. The control system not only needs to have a precise idea of the explicitly involved services, but also be able to link them with the associated smart entities (through actuators and/or sensors). The question now is how can the discovery process find available smart entities, and link them with their associated services and locations? Using an ontology and semantic-based discovery mechanism can help link and interpret the purpose and usage of both smart entities and their associated services.

Service Orchestration in SRE

4. The decision making process is often a conditional set of actions that need to be taken in order to achieve the desired goal. For example, in a scenario where an inhabitant leaves the home, the decision maker could determine that it should turn off all the lights and reduce the temperature of the heating system.

5. The service implementation and composition details should be hidden from the user. This might imply that an abstract generic process template is defined to raise the level of abstraction of the composing services to user interactions with the system.

Process-Oriented Domain Specific Language for SRE

6. The orchestration of hundreds of applicants is difficult using only existing solutions like web service technologies with a composition language like BPEL4WS. It is difficult to build control systems when all the service software components are explicitly defined. There is a lack of process-oriented domain specific languages to facilitate the integration of the smart entities involved and their services in a declarative manner to automatically discover and transparently use smart objects under the control of a centralised process.
7. The processes should support event handlers (an event represents the occurrence of something interesting). Event handlers allow processes to respond to the expiration of timers or to events by executing a specified set of operations independently from the rest of the process. The decision-making is often related to the efficient sensing of the current situation in the smart space. In the home scenario, movement detection or a temperature change can happen at any moment. Providing an event handler mechanism makes the process more aware of the context.
8. The process-oriented language should provide control flow constructs (while, if, pick (switch), flow, etc.) to allow for creating complex orchestration control scenarios.
9. Parallel execution is extremely useful whereby a process, such as an energy saving scenario, must react to events simultaneously over a long period of time.
10. The process should be generic in order to support multi-scenarios in a smart environment context like saving energy, security etc.
11. Finally, the end-user should be able to interact with the system to reconfigure and customize the processes according to her needs.

3.8 Summary

This chapter presented the background and current status of smart environment architecture, mainly focusing on the collaboration aspects and challenges related to physical components accessibility, discovery, and control. Physical spaces are characterized by being highly distributed and very physically coupled. This involves challenges unknown in traditional distributed network systems. The chapter examined the available technologies and the unresolved challenges in SE both in lower layers (sensors and actuators that enable coupling between the virtual and physical worlds) and in the higher layers (application and middleware).

On the one hand, emerging SOA concepts and technologies from many initiatives related to smart automated environments help data sharing and solve heterogeneity problems because of the different network protocols. According to [43], a service-oriented paradigm offers the following advantages:

- *Interoperability*: Services enable the sharing of data and communication between different smart entities and applications in a flexible manner. Thus, services and applications become platform and technology independent enabling new applications without adding new hardware components to be deployed.
- *Reusability*: A service is a reusable component. Services provide the means to make a pre-existing code available through the internet and can be remotely accessed using standards protocols such HTTP. As a result, the program's functionalities can be invoked by many applications for different purposes.
- *Standardized Protocol*: Web services are implemented with standard protocols for communication, service description, and discovery using XML messages. This has many advantages like unified and standardised access, heterogeneity, wide range of choices, and reduction in costs.
- *Services Discovery*: SOA offers a service discovery ability to find the desired services.
- *Loose Coupling*: Service consumers and providers are loosely coupled. This implies that the service requestor can ignore any implementation or technical details of the service provider (e.g. programming language, platform).
- *Composability*: SOA enables the use of composing languages like BPEL4WS to combine the available services to address new requirements. Smart systems rely more and more on services composition to provide intelligent automation to their final users.

On the other hand, from the smart environment services discovery and composition perspective, hardware components still need methods to describe themselves and their services to allow them to be (automatically) discovered and used. In the application layer, embedded devices remain hard to integrate into complex control applications. Several challenges still need to be addressed in order to build a smart system able to orchestrate smart objects: for instance, there is a lack of frameworks and domain specific languages to facilitate the orchestration of smart spaces, and a need to involve user preferences and context-awareness in the control process. These concerns are the subjects of the next chapters.

4

A Framework for Controlling Smart Residential Environments

The previous chapter demonstrated the current technologies and challenges related to controlling SRE. The goal of this chapter is to clearly define all the concepts involved in the GF4SRE framework proposed for managing SRE. This chapter is a reworked and extended version of the publications in [31, 101-104].

4.1 Related Work

A number of attempts related to the control of SRE have been made. According to the orchestration techniques mentioned in Section 3.5.1 and shown in Figure 4.1, *manual compositions* [90, 105-109] are often defined as workflow using a procedural process-oriented language like BPEL4WS or BPMN. The users program and tell the system which operations to perform or which actions to take during the composition process development steps. Many tools like Oracle SOA Suite [91], JOpera [110] are available to enable the creation of workflow-based processes. The problem with this approach is that users have to make tedious efforts to find, and understand detailed knowledge about many web service resources, and demand usually explicit details about web service grounding. This approach does not often deal with changing environments, where services and devices might change their addresses, or be replaced. Web services have to be known during the process design phase, and discovery is done in the most cases manually.

The *automatic composition* technique (without human involvement) is used when the user has no process model, but does have a set of predefined goals, constraints (rules) and preferences. It involves the representation of possible actions and their effects, and techniques for efficiently searching for possible plans. Dynamic composition methods are required to generate the final plan automatically. A common characteristic of these approaches is the use of a declarative programming style. Several such approaches have been introduced in the literature [111, 112], e.g., those based on Linear-time Temporal Logic (LTL) [113, 114], Artificial Intelligence Planning (AI) [16, 115-118], or Rule and Policy-Based [77, 119-124]. However, these techniques are still viewed as highly complex because of the rapid proliferation of available services to choose from, and the risks of producing side effects or drifting away from the initial user's goal. Rule-based systems often require a large number of rules that programmers define manually to deal with the environment context and which might produce a conflict between different rules when several rules are active. The automatic approach is relatively inefficient and is significantly more complex when programmers know which set of actions and conditions to take to achieve a predefined goal such as saving energy or for security.

The third technique (which is close to this thesis) is the *semi-automatic* (interactive) one. In this kind of orchestration, the system usually helps users to find, filter, and integrate automatically the desired services while the programmer defines the control workflow [125-127]. In addition, it may enable end-users to interact with the system during the workflow execution. Some research efforts have exploited ontology and semantic data models to partially automate the services' discovery and compositions [33, 77, 128]. However, orchestrating the interaction between smart entities to direct their behaviours is still a challenging task that needs solving both in terms of low-level accessibility problems and high-level discovery and composition challenges. This thesis intends to deal with these issues by proposing a generic framework for managing SRE.

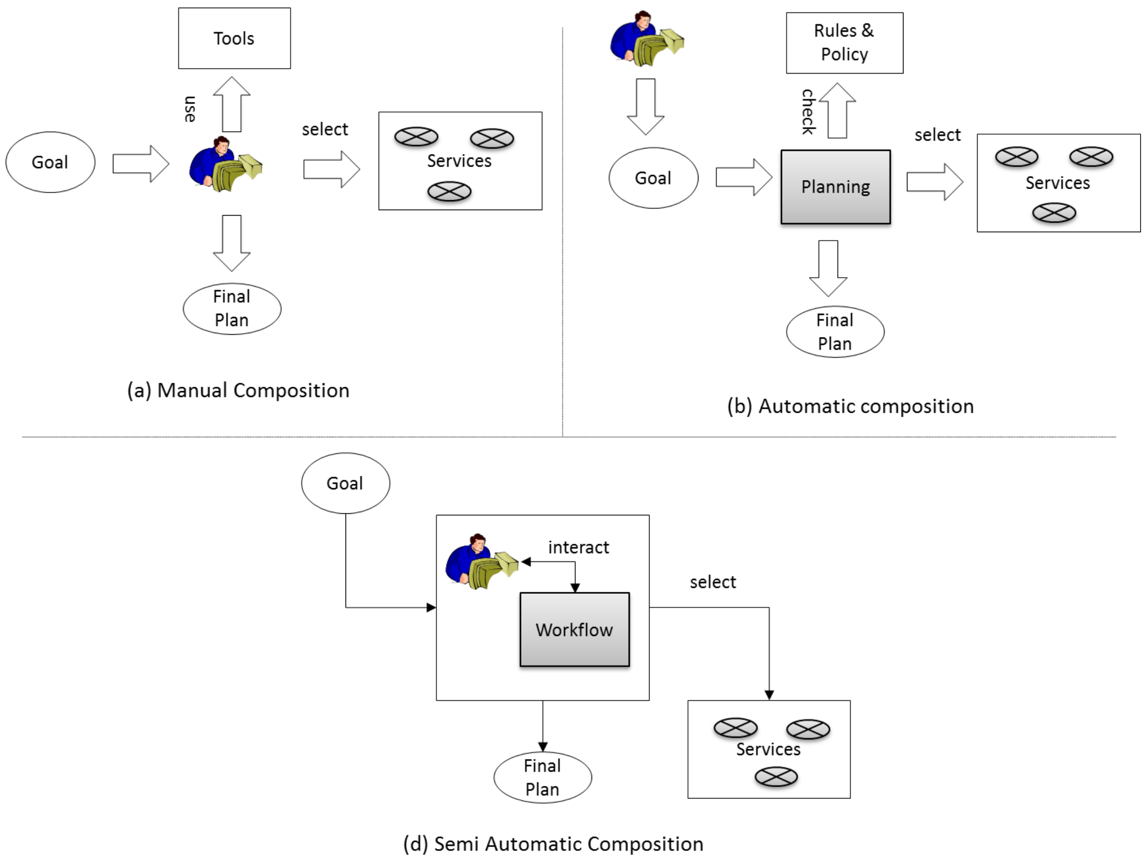


Figure 4.1: Orchestration techniques

4.2 The GF4SRE Framework

The overall goal of this thesis is to define a framework and its software architecture and the infrastructure necessary to support the integration of the physical objects in the digital world and facilitate the modelling and implementation of the tasks associated with SRE management. The main refined elements that together embody the GF4SRE framework and satisfy the requirements discussed in Section 3.7.2 are the following.

1. **Modelling and Describing SRE:** Consider there is a smart environment, which we want to control. It consists of diverse networked smart objects that inhabitants use. These smart entities are coupled with miniaturized actuators and sensors which provide services in the form of actions, queries and events to change, measure, or perceive their states. One of the contributions of this thesis is to propose the Ont4SRE ontology for describing and modelling the smart residential environment. It is necessary to have a clear description of the smart entities' properties and capabilities. It must describe the building environment resources, including household appliances, lights, windows, doors, etc., as well as their properties, locations, and functions.
2. **Accessibility & Interoperability of Services in SRE:** In the network layer, actuators and sensors are supposed to be connected using the adapted protocols introduced in Section 2.3. In the Service Layer, WS-* is a commonly used technology to solve heterogeneity and interoperability challenges in the middleware layer. Web servers are embedded in sensors and actuators and apply the WS-* web service style to expose their services through a WSDL interface, as explained in Subsection 3.3.3.2. How to model the functionalities and the services of SRE using the WS-* principles is explained and supporting the WSDL syntactic description with Ont4SRE-based semantic annotations to enable the discovery of relevant smart entities and their services automatically in an abstract manner is also suggested.
3. **Services Discovery in SRE:** A software component, which deals with the registry and discovery requirements in the SRE context, is proposed. The Ont4SRE ontology and a proposed query-based discovery mechanism are suggested in order to link and interpret the purpose of both smart entities and their associated services.
4. **Services Orchestration and a Process-Oriented DSL for SRE:** Another fundamental element is the scenario templates used to control the SRE, written within an original process-oriented domain specific language, and characterised by the following.
   - It is a workflow-based composed of several activities linked with control flow and execution constructs.
   - It enables the discovery of the smart entities automatically, and supports events handling.
   - The end user is able to configure and customizes each template according to her requirements and preferences.
   - There are translation and execution software components used to compile and run the templates.

4.3 The Global Software Architecture

The GF4SRE framework architecture proposed in this thesis is illustrated in Figure 4.2. It comprises the following components and functionalities.

- The final user can choose a recommended ***Generic Process Template*** (a control scenario) from the ***Process Templates Repository***. Templates are defined using the ***GPL4SRE*** domain specific language and based on a pluggable ***Ont4SRE*** ontology.
- The ***Process Generator*** converts the chosen template into an executable process.
- At the execution stage, the semantic smart entities' characteristics that are declared in the process are captured and sent to the ***Registry & Discovery Engine*** as a service query. Smart entities and their available services are usually published in the service registry.
- It is possible for the final user to precise his context preferences through a simple ***Client User Interface***.
- The ***Process Execution Engine*** is the runtime environment.

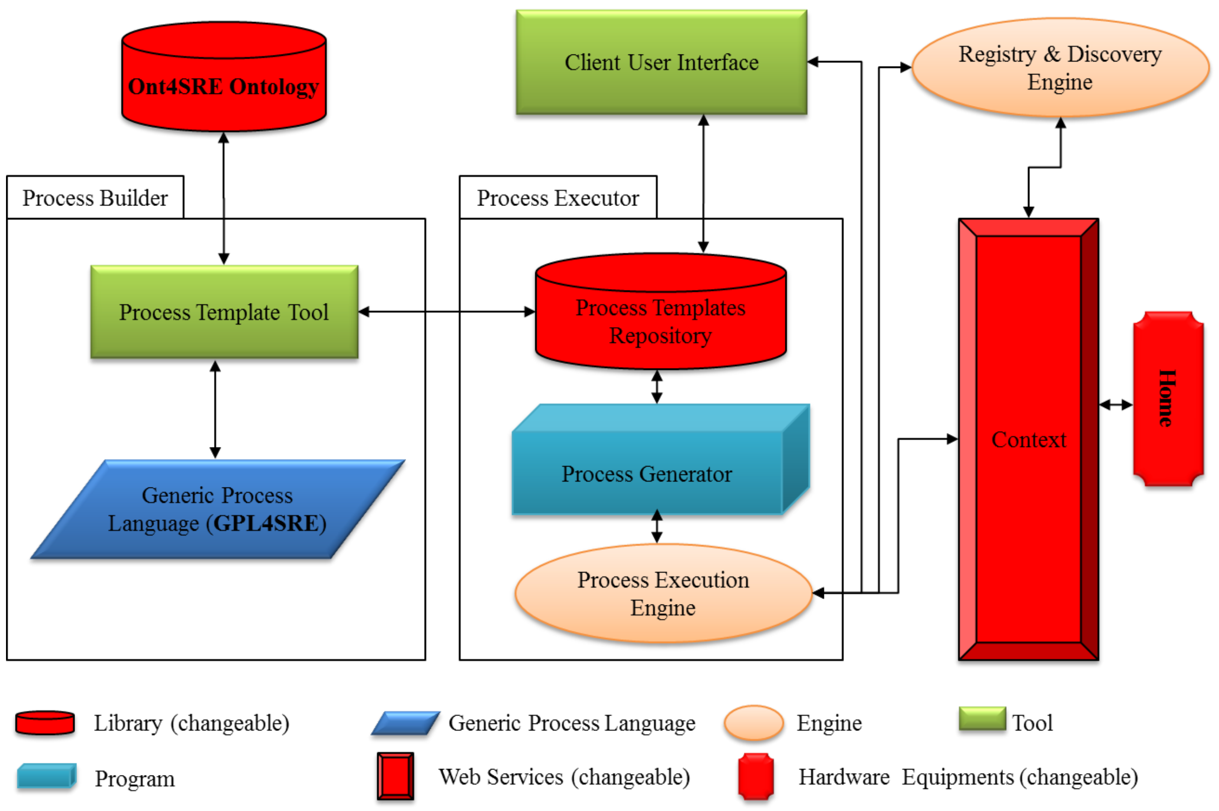


Figure 4.2: GF4SRE framework architecture

The remainder of this chapter explains in details the concept and role of each component in the GF4SRE framework.

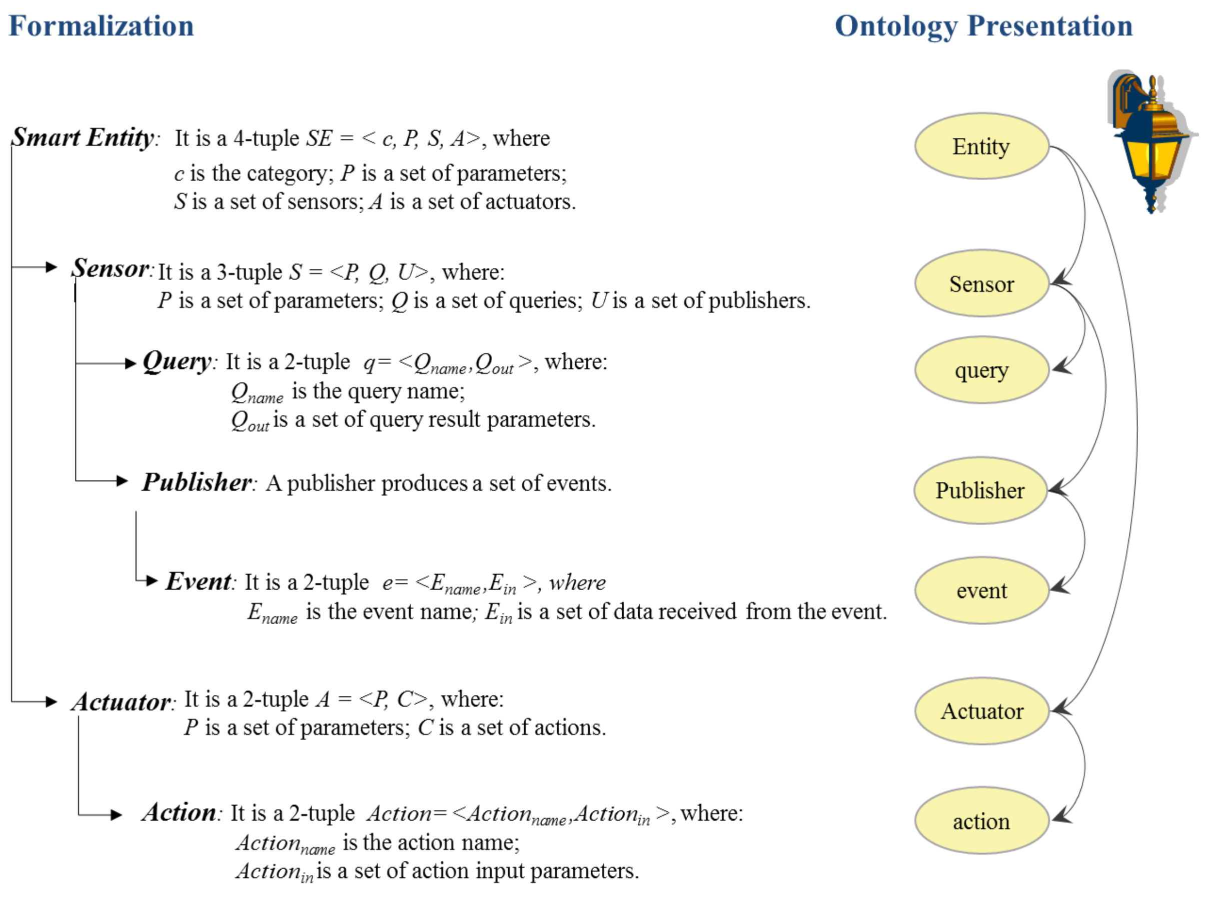
4.4 The Ont4SRE Ontology for SRE

Several efforts have been introduced for modelling household environments or describing device hardware using ontologies, including [129, 130] which introduce an ontology to describe the hardware, software and measurement capabilities of devices like sensors. Ontologies like [131-133] have been proposed to reason, manipulate and access context information in smart environments. They offer inference rules to reason about the context and facilitate data sharing. In [134], Natalia et al. surveyed other ontologies which support modelling personal profiles and human behaviour recognition to help build assistance systems in smart homes. However, it is not enough to have wide information about appliances and environment context alone; many other essential concepts like sensors, actuators, services and their parameters to help functionality modelling in ambient systems are required.

From the functionalities and service concept perspectives, DomoML [135] is another example of an ontology for building household models. It consists of two main ontologies: DomoML-environment is to describe the objects in the house; and DomoML-function is to specify the devices' functionalities. Therefore, this work benefits from this and similar ontologies like [136, 137], to propose the Ont4SRE ontology with more concepts and relations to sufficiently model SREs. The Ont4SRE ontology serves as the link between the entities in the physical world and the services of the virtual one in order to model and control the SRE at a high level of abstraction.

4.4.1 Ont4SRE Concepts

The Ont4SRE defines an ontology model to share a common presentation of the SRE, including the smart entities, locations, associated embedded devices (sensors, actuators), services (action, query, and event) and the relationships between them. This enables questions such as what the smart entities present in the space are; where they are located; to which actuators and sensors are they attached; and what are the services capabilities that they might provide, to be answered. The formalisation of the SRE introduced in Section 2.6 is transformed into an Ont4SRE ontology model, as illustrated in Figure 4.3.



Smart Entity: It is a 4-tuple $SE = < c, P, S, A>$, where
c is the category; P is a set of parameters;
S is a set of sensors; A is a set of actuators.

Sensor: It is a 3-tuple $S = <P, Q, U>$, where:
P is a set of parameters; Q is a set of queries; U is a set of publishers.

Query: It is a 2-tuple $q= <Q_{name}, Q_{out}>$, where:
Q_{name} is the query name;
Q_{out} is a set of query result parameters.

Publisher: A publisher produces a set of events.

Event: It is a 2-tuple $e= <E_{name}, E_{in}>$, *where*
E_{name} is the event name*;* E_{in} is a set of data received from the event.

Actuator: It is a 2-tuple $A = <P, C>$, where:
P is a set of parameters; C is a set of actions.

Action: It is a 2-tuple $Action= <Action_{name}, Action_{in}>$, where:
$Action_{name}$ is the action name;
$Action_{in}$ is a set of action input parameters.

Figure 4.3: Mapping between formalization & ontology presentation

Figure 4.4 shows the schema of the proposed Ont4SRE ontology. It is modelled with the web ontology language (OWL) and realised with Protégé as the tool to describe the different concepts of a SRE. Each concept of the Ont4SRE ontology is represented as an OWL class. The Ont4SRE model consists of the following concepts: *Entity*, *Actuator*, *Sensor*, *Publisher*, and *Services*: *action*, *event*, and *query*, and the relevant relationships between them to fully describe the smart entities occupying the space.

These main class concepts can be applied to any smart residential building populated with diverse networked devices and smart entities (e.g. home, school, hospital), while the hierarchy in each class can be differentiated from one building to another depending on the building's characteristics and usage. In this thesis, the Ont4SRE ontology is modelled to fit a smart home building environment. Details of each concept follow below.

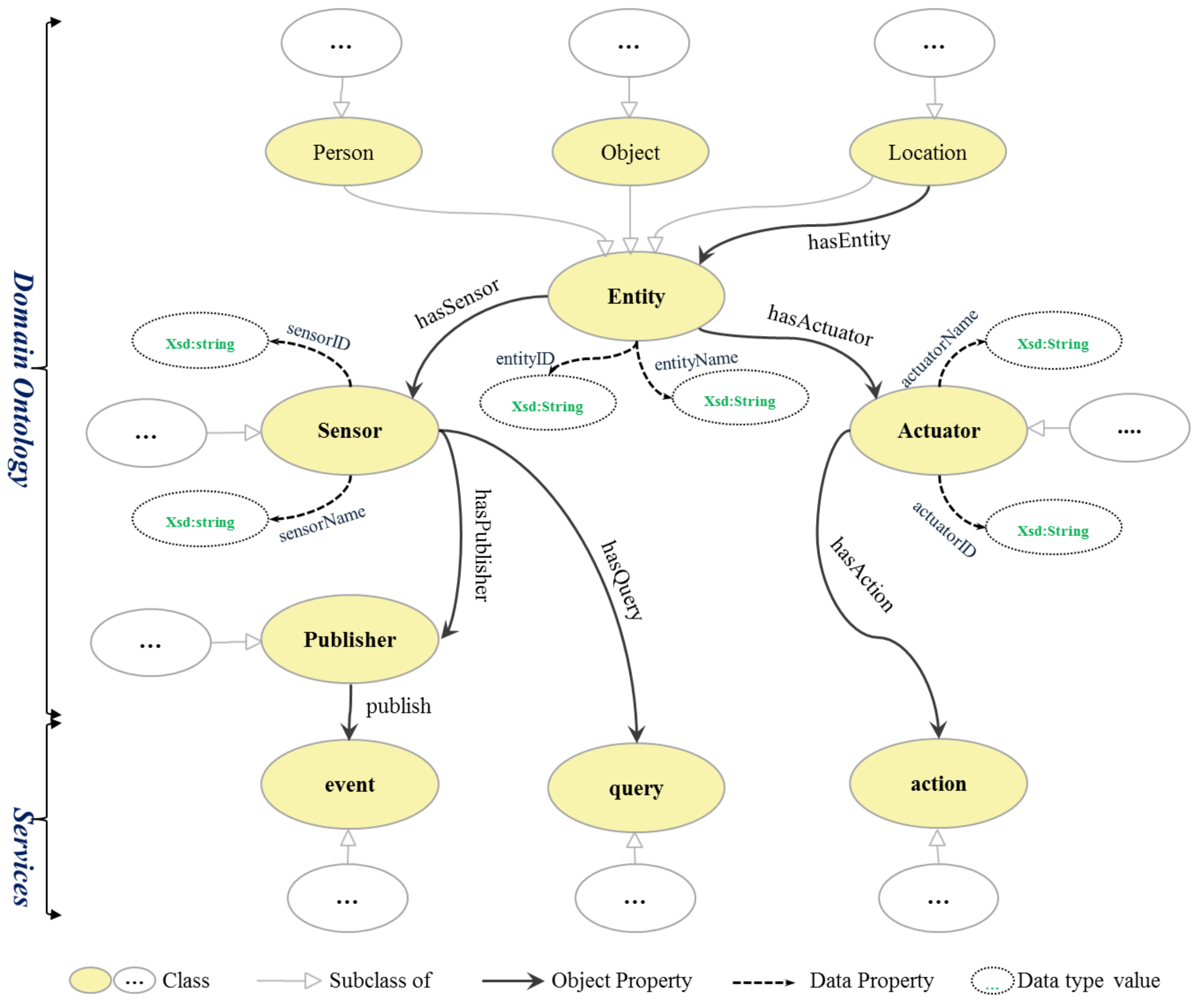


Figure 4.4: Smart residential environment ontology

- **Entity Modelling**

The **`Entity`** class concept represents and models which objects, persons, and locations may occupy the space and could be smart. It is the superclass of all the entities. The hierarchy of entity subclasses is clearly defined. Figure 4.4 shows the first subclasses of the **`Entity`** class which are **`Object`**, **`Location`**, and **`Person`**.

1. **`Object`** class represents different appliances, objects, and devices that may occupy the space. Figure 4.5 illustrates the different objects which are classified into different types and extended to subclasses to better navigate (e.g. safety warning equipment, white and brown goods, furniture, etc.).

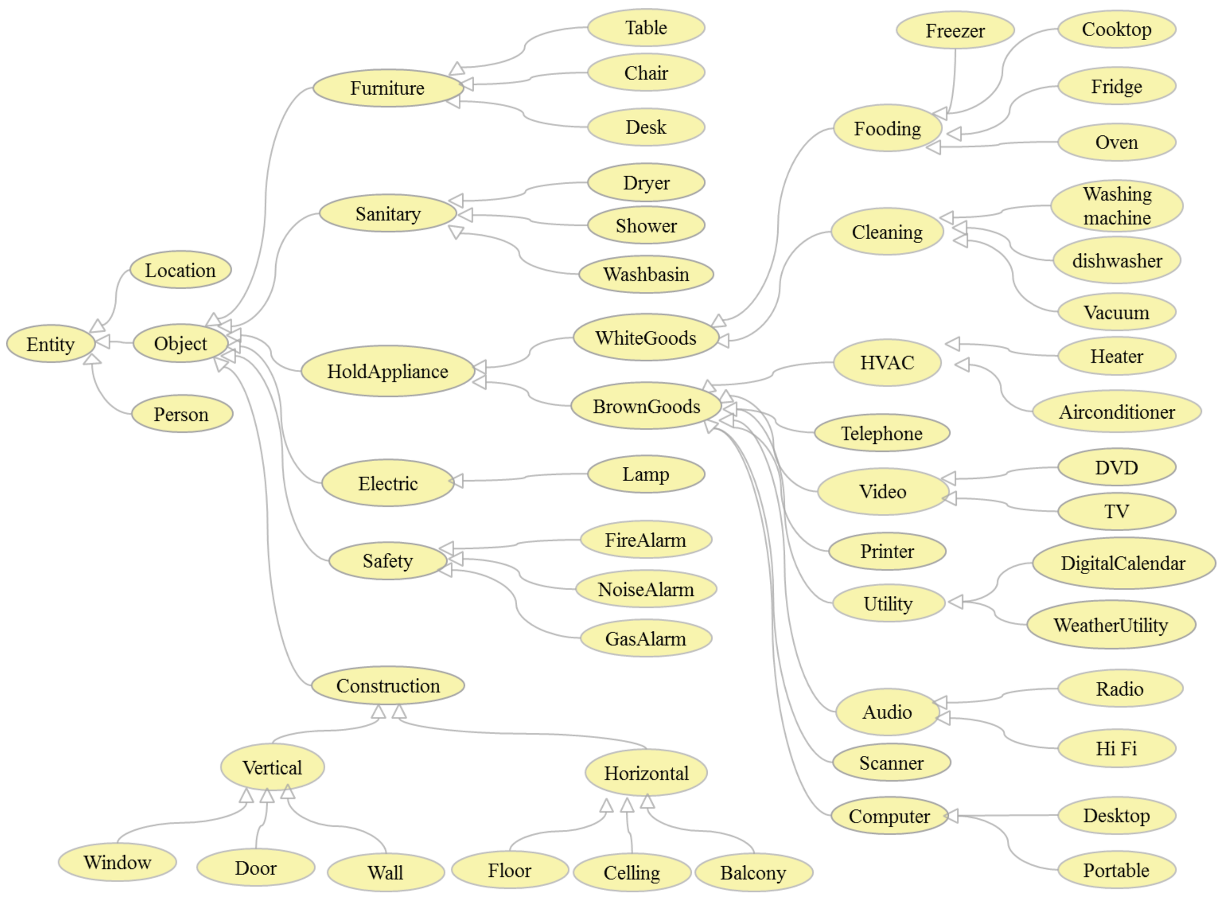


Figure 4.5: A presentation of Object subclasses

2. **`Location`** class is where smart entities could be located. Figure 4.6 illustrates different location places which may be present in a smart home environment. The location tree supports the usual space locations including, for example, room, bedroom, bathroom, garage, and garden. One of the most important aspects of contextual information is where a physical entity is located. In the GF4SRE framework, the notion of location is used to represent where an entity is in an indoor environment rather than where a service is in an outdoor environment. As an example, in a saving energy scenario, if there is nobody in room x, switching off lights and unnecessary electrical equipment during the inhabitant's absence is a good idea to economize energy. To achieve this, the home control system needs to know which lamps and electric devices are in room x.

3. **`Person`** class represents who occupies the space. As shown in Figure 4.6, the person tree supports different physical actors (e.g., dweller, guest, or a family member). A physical person is considered a smart entity, as an object, since she can, for example, wear, or implant in her body small devices and sensors which can determine her presence, identity or health.

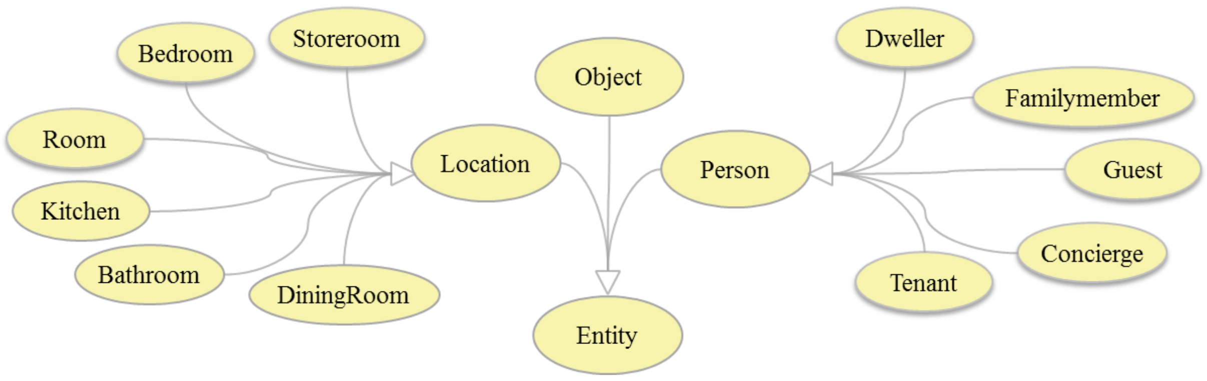


Figure 4.6: A presentation of Person and Location subclasses

- **Actuator Modelling**

  An **`Actuator`** class represents actuators which could be associated to entities in order to change their states. As shown in Figure 4.7, the **`Actuator`** class has many subclasses to represent different actuator types. For example, there is a **`DoorController`** class to control access to a door, a **`LampController`** class to control a lamp and so on. Several smart entities may use the same type of actuator when they offer the same services. Therefore, it is not necessary to have the same number of actuators as entities.

- **Sensor Modelling**

  The **`Sensor`** class concept refers to all the kinds of sensor associated with entities in order to read their states. The **`Sensor`** class has many subclasses to represent different sensor types. Figure 4.7 shows the screenshots of some of these created classes. For example, the class **`FireSensor`** is for sensors able to detect a fire, while the class **`MotionSensor`** contains sensors able to detect any movement that might occur in a given space.

- **Publisher Modelling**

  The **`Publisher`** class refers to all kinds of publisher that may be associated with sensors in order to publish one or more events. As shown in Figure 4.7, the **`Publisher`** class has many subclasses to represent different publisher types. For example, a **`FireEvent Publisher`** is used to publish a fire event while a **`MotionEventPublisher`** is used to publish an event concerning any movement.

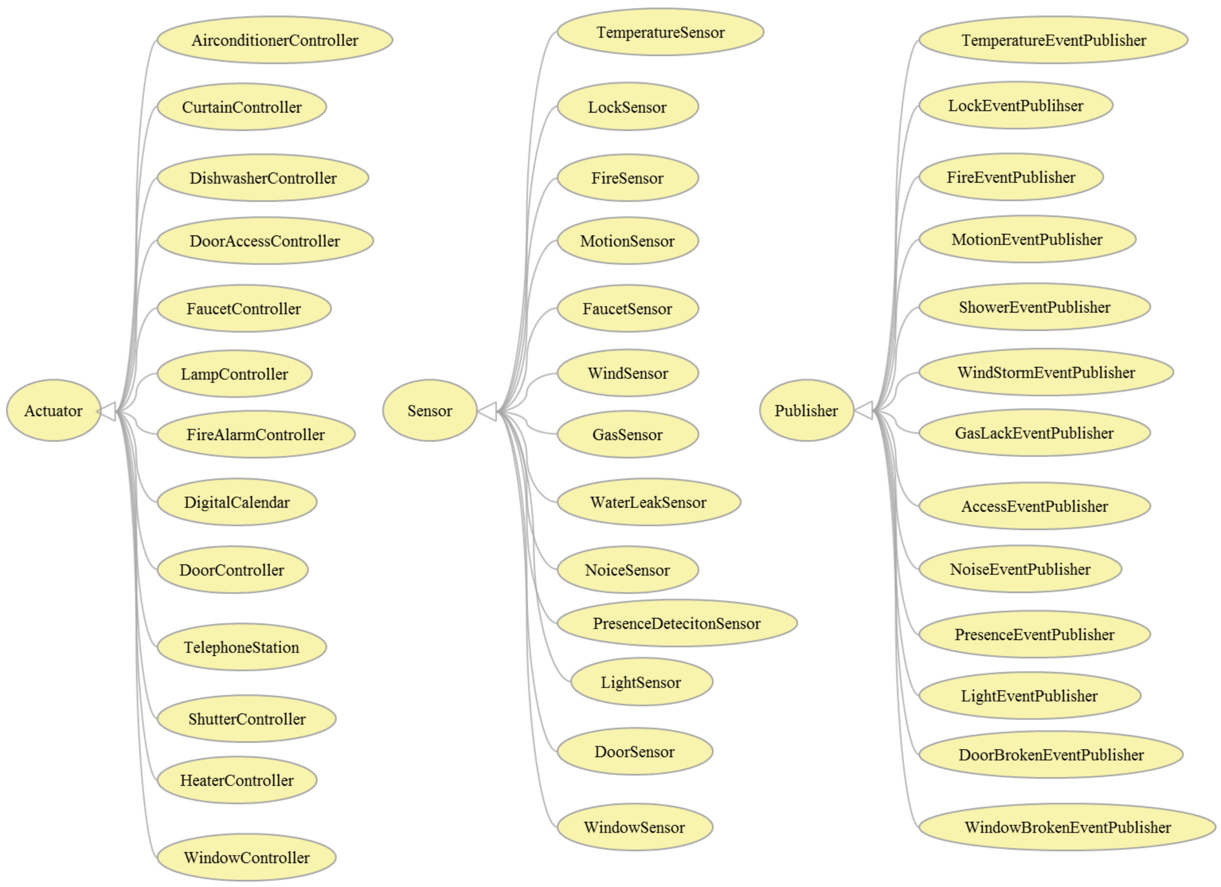


Figure 4.7: A presentation of Sensor, Actuator, and Publisher subclasses

- **Service Modelling**

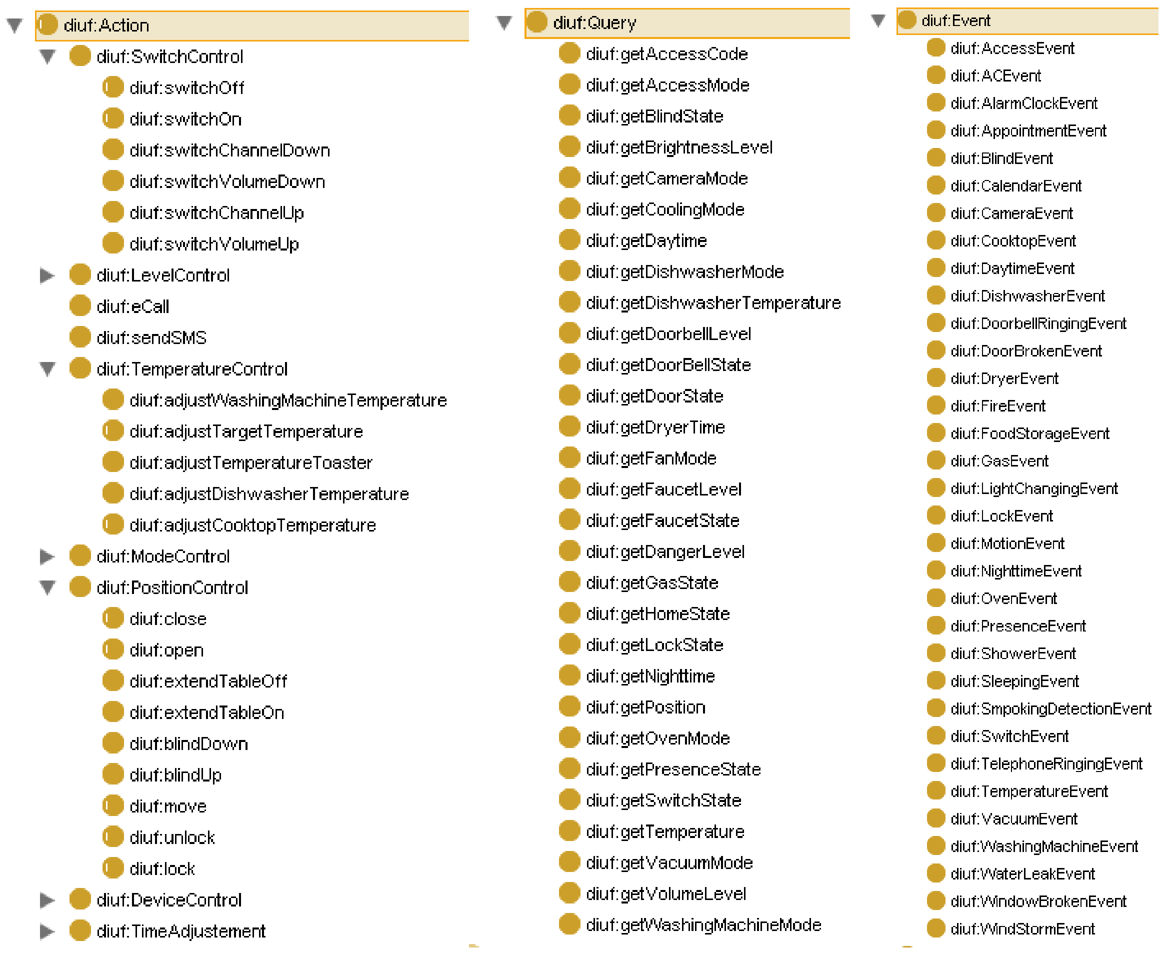
The goal of the **`Service`** concept is to model what actuators, sensors and publishers can do in the form of action, queries, and events. As shown in Figure 4.8, they are divided into three main subclasses, as follows.

1. The **`action`** class refers to all actions provided by actuators to change the state of entities. Each actuator can perform one or more actions. For example, A **`Heater Controller`** actuator can switch a heater on or adjust its target temperature using actions **`switchOn`**, and **`adjustTargetTemperature`** respectively; another actuator named **`FaucetController`** provides actions (**`open`**, **`close`**) to control the valve by adjusting the water temperature and opening or closing it. Figure 4.8 shows an extract of the created actions.

2. The **`query`** class refers to all queries that may be provided by sensors in order to retrieve the current state of the smart entities. Each sensor can respond to one or more queries. For example, a **`PresenceSensor`** can inform us if a given room is occupied by using a **`getPresenceState`** query. Another query like **`getSwitchState`** can check if the TV or the lamp is on or off. Figure 4.8 shows some examples of possible queries. Further software applications can provide virtual web services such as weather station software which can be used to provide information about weather

forecasts or an electronic version of a calendar to say whether it is daytime or when the holidays are.

3. The **`event`** class refers to all the events that may occur. An event occurs when the property value of a smart entity changes. An event is detected by a sensor and published by the publisher associated with this sensor. Each publisher can provide one or more events. For example, **`DoorBrokenEventPublisher`** is the publisher of a door sensor, which can produce an event when it is broken. Another event like **`TemperatureEvent`** notifies the inhabitant if the temperature changes Figure 4.8 shows examples of possible events. Further software applications can be designed to publish events about particular information such as an electronic calendar and time management software agent with an appointment reminder when meetings are due, or when user holidays start and end.



- diuf:Action
  - diuf:SwitchControl
    - diuf:switchOff
    - diuf:switchOn
    - diuf:switchChannelDown
    - diuf:switchVolumeDown
    - diuf:switchChannelUp
    - diuf:switchVolumeUp
  - diuf:LevelControl
  - diuf:eCall
  - diuf:sendSMS
  - diuf:TemperatureControl
    - diuf:adjustWashingMachineTemperature
    - diuf:adjustTargetTemperature
    - diuf:adjustTemperatureToaster
    - diuf:adjustDishwasherTemperature
    - diuf:adjustCooktopTemperature
  - diuf:ModeControl
  - diuf:PositionControl
    - diuf:close
    - diuf:open
    - diuf:extendTableOff
    - diuf:extendTableOn
    - diuf:blindDown
    - diuf:blindUp
    - diuf:move
    - diuf:unlock
    - diuf:lock
  - diuf:DeviceControl
  - diuf:TimeAdjustement

- diuf:Query
  - diuf:getAccessCode
  - diuf:getAccessMode
  - diuf:getBlindState
  - diuf:getBrightnessLevel
  - diuf:getCameraMode
  - diuf:getCoolingMode
  - diuf:getDaytime
  - diuf:getDishwasherMode
  - diuf:getDishwasherTemperature
  - diuf:getDoorbellLevel
  - diuf:getDoorBellState
  - diuf:getDoorState
  - diuf:getDryerTime
  - diuf:getFanMode
  - diuf:getFaucetLevel
  - diuf:getFaucetState
  - diuf:getDangerLevel
  - diuf:getGasState
  - diuf:getHomeState
  - diuf:getLockState
  - diuf:getNighttime
  - diuf:getPosition
  - diuf:getOvenMode
  - diuf:getPresenceState
  - diuf:getSwitchState
  - diuf:getTemperature
  - diuf:getVacuumMode
  - diuf:getVolumeLevel
  - diuf:getWashingMachineMode

- diuf:Event
  - diuf:AccessEvent
  - diuf:ACEvent
  - diuf:AlarmClockEvent
  - diuf:AppointmentEvent
  - diuf:BlindEvent
  - diuf:CalendarEvent
  - diuf:CameraEvent
  - diuf:CooktopEvent
  - diuf:DaytimeEvent
  - diuf:DishwasherEvent
  - diuf:DoorbellRingingEvent
  - diuf:DoorBrokenEvent
  - diuf:DryerEvent
  - diuf:FireEvent
  - diuf:FoodStorageEvent
  - diuf:GasEvent
  - diuf:LightChangingEvent
  - diuf:LockEvent
  - diuf:MotionEvent
  - diuf:NighttimeEvent
  - diuf:OvenEvent
  - diuf:PresenceEvent
  - diuf:ShowerEvent
  - diuf:SleepingEvent
  - diuf:SmpokingDetectionEvent
  - diuf:SwitchEvent
  - diuf:TelephoneRingingEvent
  - diuf:TemperatureEvent
  - diuf:VacuumEvent
  - diuf:WashingMachineEvent
  - diuf:WaterLeakEvent
  - diuf:WindowBrokenEvent
  - diuf:WindStormEvent

Figure 4.8: Examples for action, query, and event subclasses

4.4.2 Ont4SRE Properties and Restriction Types

Having defined the different classes that represent the necessary concepts to describe a smart space, this subsection introduces the necessary property elements in order to enrich the model, especially the relationship between the entity, sensors, actuators, and services (action, query, and event). To create these relationships, the **`ObjectProperty`**, **`DatatypeProperty`** relationships, and restriction constraints are used.

4.4.2.1 Object Properties

An object property links a class instance with another class instance. For example, the object property **`hasAction`** links an actuator instance with a specific action instance (e.g. a **`LampController`** instance with a **`switchOn`** instance). OWL Language enables a class to inherit all the properties of its ancestor classes. According to the ontology schema introduced in Figure 4.4, and as shown in Figure 4.9, the following object properties are defined.

- **`hasEntity`** links an entity to its location. Its inverse object property is **`hasLocation`**.
- **`hasActuator`** links an entity to its actuators. Its inverse object property is **`isActuatorOf`**.
- **`hasAction`** links an actuator to its actions. Its inverse object property is **`isActionOf`**.
- **`hasSensor`** links an entity to its sensors. Its inverse object property is **`isSensorOf`**.
- **`hasPublisher`** links a sensor to its publishers. Its inverse object property is **`isPublisherOf`**.
- **`hasQuery`** links a sensor to its queries. Its inverse object property is **`isQueryOf`**.
- **`hasEvent`** links a publisher to its events. Its inverse object property is **`isEventOf`**.

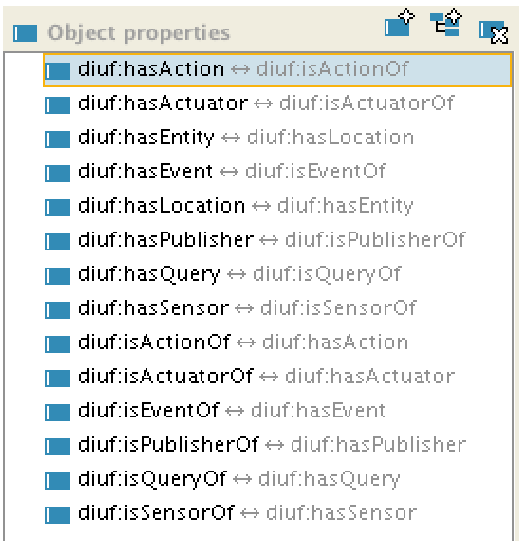

Object properties
diuf:hasAction ↔ diuf:isActionOf
diuf:hasActuator ↔ diuf:isActuatorOf
diuf:hasEntity ↔ diuf:hasLocation
diuf:hasEvent ↔ diuf:isEventOf
diuf:hasLocation ↔ diuf:hasEntity
diuf:hasPublisher ↔ diuf:isPublisherOf
diuf:hasQuery ↔ diuf:isQueryOf
diuf:hasSensor ↔ diuf:isSensorOf
diuf:isActionOf ↔ diuf:hasAction
diuf:isActuatorOf ↔ diuf:hasActuator
diuf:isEventOf ↔ diuf:hasEvent
diuf:isPublisherOf ↔ diuf:hasPublisher
diuf:isQueryOf ↔ diuf:hasQuery
diuf:isSensorOf ↔ diuf:hasSensor

Figure 4.9: A presentation of Ont4SRE object properties [138]

In order to enrich the Ont4SRE model, other object properties are introduced such as **`hasFloor`** to specify on which floor each entity is. The **`hasOwner`** is used to identify the owner of each location space. The more object properties declared, the more useful queries can be asked and the more indirect result predicted from the ontology.

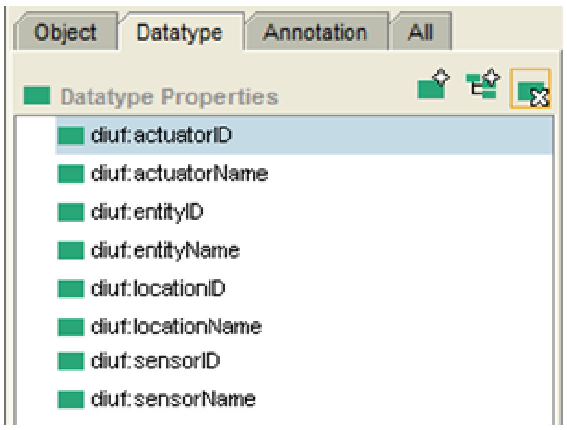
4.4.2.2 Ont4SRE Properties' Characteristics

One smart space characteristic is that each smart entity can have at a given time only one location in the smart space. For example, if the instance **`window_1`** is located in **`room_1`**, it could not be located in **`room_2`** at the same time. To satisfy this condition, the functional property is used where instances can be restricted to belonging to a unique property. Thus, the property **`hasLocation`** must be functional. Further, an actuator or a sensor instance can belong to only a single entity instance. Similarly, this happens with action, query, and event instances where they belong to only a single actuator, sensor, or publisher instance, respectively. To satisfy these conditions, the properties **`isActuatorOf`**, **`isSensorOf`**, **`isPublisherOf`**, **`isActionOf`**, **`isQueryOf`**, **`isEventOf`** are functional.

4.4.2.3 Datatype Properties

To describe well a smart entity, mandatory information about each class instance is defined such as providing a name and a unique identifier for each entity and its associated devices. This is to distinguish the different instances of the same type. As shown in Figure 4.10, the following data type properties are created.

- An actuator has a Datatype property **`actuatorID`** and **`actuatorName`**.
- An entity has a Datatype property **`entityID`** and **`entityName`**.
- A location has a Datatype property **`locationID`** and **`locationName`**.
- A sensor has a Datatype property **`sensorID`** and **`sensorName`**.



diuf:actuatorID
diuf:actuatorName
diuf:entityID
diuf:entityName
diuf:locationID
diuf:locationName
diuf:sensorID
diuf:sensorName

Figure 4.10: Some of Ont4SRE data type properties

4.4.2.4 Entity State modelling

According to the SRE formalization introduced in Section 2.6, each smart entity is characterised by a set of optional parameters (entity states) which give information about the entity at a given point in time. For this purpose, additional data types are specified for each entity declared in the Ont4SRE ontology which link one or more entity (domain) to data type values (range). For example, Figure 4.11 shows a property **`switchState`** for electrical and hold appliance subclass instances (e.g. lamp, air conditioner, heater, or TV). It has a range of string values equal to "**`on`**" or "**`off`**". Further, each data type declared may be associated with an action, query or event, if the latter are responsible for changing the entity concerned state. For example, a **`getSwitchState`** query is a service provided by a sensor to get the **`switchState`** value of its associated smart entity.

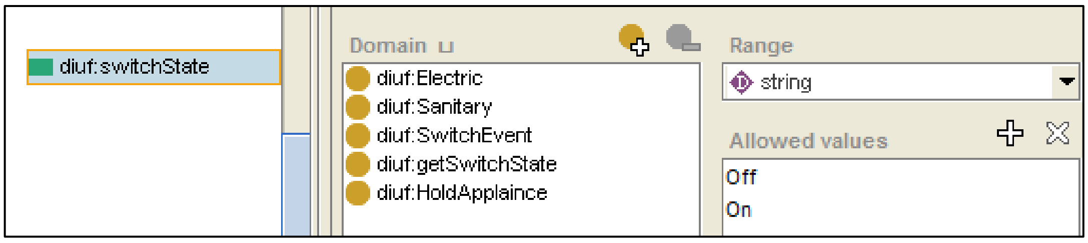


Figure 4.11: The relationship between the switchState property and related classes

4.4.2.5 Restriction Types

Restriction types are used to determine which kind of actuators, sensors, and services can be associated with each entity. For the Ont4SRE ontology, the following restrictions are defined.

1. Each entity can have zero, one or several actuators, and/or sensors, but can be linked only to a certain type of actuators and sensors. For example, as shown in Listing 4.1, a **`Lamp`** entity instance can have only a **`LampController`** actuator and a **`LampSensor`**. The *allValueFrom* type constraint is used to specify these restrictions.

```
<owl:Class rdf:ID="Lamp">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasActuator"/>
        </owl:onProperty>
        <owl:allValuesFrom>
          <owl:Class rdf:ID="LampController"/>
        </owl:allValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Listing 4.1: The restrictions declaration between a Lamp and its types of Actuators

2. Each Actuator can have one or several actions, but it can provide only a certain type of actions. For example, a **`LampController`** actuator provides only three actions **`switchOn`**, **`switchOff`**, and **`adjustBrightnessLevel`**.
3. A sensor can have zero, one or several publishers, and one or several queries. Each publisher or query can be linked only to a certain type of sensor. For example, a **`LampSensor`** can have a **`LigthEventPublisher`** publisher and **`getSwitchState`** and **`getBrightnessLevel`** queries.
4. The publisher has one or several events. Each event can be produced by only a certain type of publisher. For example, a **`MotionEvent`** event can be fired by a **`MotionEvent-Publisher`** and cannot be fired by a **`FireEventPublisher`**.

4.4.3 **Key Benefits of the Ont4SRE Ontology**

Example 4.1

To illustrate the Ont4SRE benefits, consider a home consisting of four rooms each containing: a **`PresenceDetectionSensor`**, which provides a **`getPresenceState`** query and a **`PresenceEvent`** event via a **`PresenceEventPublisher`**; and a smart lamp, which provides a **`switchOn`** and **`switchOff`** service via a **`LampController`** actuator.

An instance is created for each room, lamp, actuator, sensor and service in the Ont4SRE ontology with properties and relationships similar to Figure 4.12.

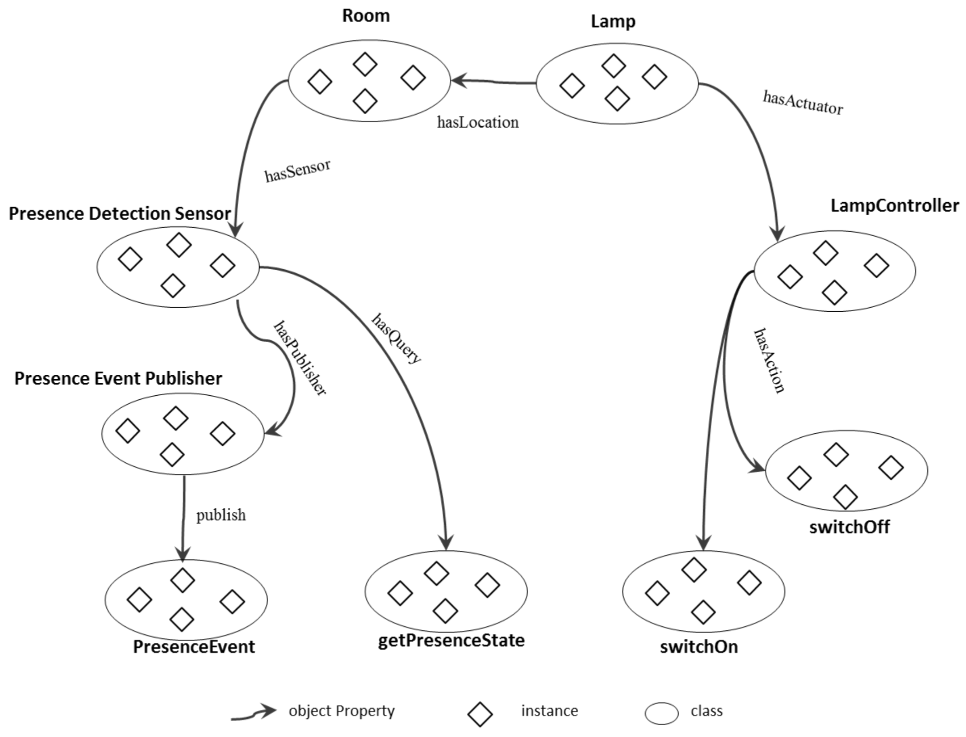


Figure 4.12: An example of Lamp entity relationships

Consider the following questions:

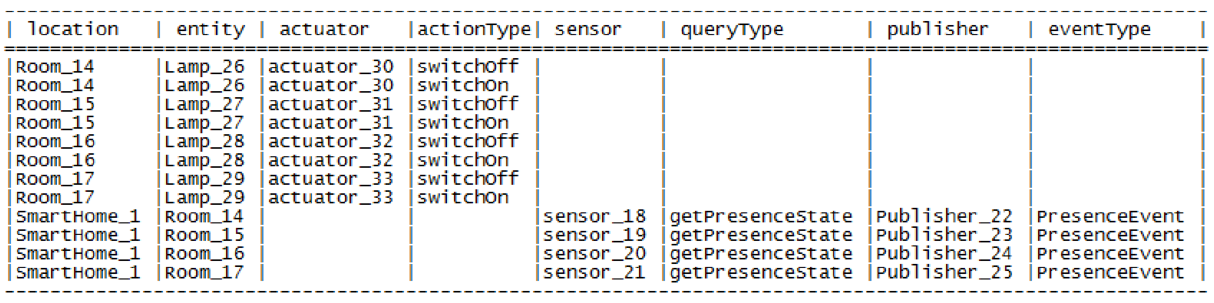
1. Which smart entities are present in the space?
2. Where are they located?
3. To which actuators and sensors are they attached?
4. Which are the service capabilities they might provide?

Listing 4.2 shows how to use a SPARQL query to get a response to each of these questions and Figure 4.13 shows the answers. The query consists of a **SELECT** clause used to retrieve the instances, and a **WHERE** clause, which includes the comparison predicates to select the instances which satisfy the question requirements.

- Line 4 declares the retrieved instances.
- Lines 6-19 use the UNION operator to combine the results of two condition sets.
  - For the first condition set, lines 6-10 specify that each entity instance must have an actuator instance and each actuator must have an action instance.
  - For the second condition set, lines 12-19 specify that each entity instance must have a sensor instance and each sensor have a query instance, and a publisher instance. Finally each publisher must have an event.
- Line 20 specifies that each entity should have a location and line 22 orders the result by entity names.

```
1     PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2     PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3     PREFIX diuf: <http://www.unifr.ch/owl/2011/03/HomeOntology.owl#>
4     SELECT distinct  ?location ?entity ?actuator ?actionType ?sensor
                       ?queryType ?publisher ?eventType
5     WHERE{
6           {
7            ?entity diuf:hasActuator ?actuator.
8            ?actuator diuf:hasAction ?action.
9            ?action rdf:type ?actionType.
10          }
11          UNION
12          {
13           ?entity diuf:hasSensor ?sensor.
14           ?sensor diuf:hasQuery ?query.
15           ?query rdf:type ?queryType.
16           ?sensor diuf:hasPublisher ?publisher.
17           ?publisher diuf:hasEvent ?event.
18           ?event rdf:type ?eventType.
19          }
20          ?entity diuf:hasLocation ?location.
21        }
22    order by (?entity)
```

Listing 4.2: A SPARQL query to get a list of the smart entities



```
--------------------------------------------------------------------------------------------------------------
| location     | entity | actuator     |actionType| sensor     | queryType         | publisher     | eventType     |
==============================================================================================================
|Room_14       |Lamp_26 |actuator_30 |switchOff |            |                   |               |               |
|Room_14       |Lamp_26 |actuator_30 |switchOn  |            |                   |               |               |
|Room_15       |Lamp_27 |actuator_31 |switchOff |            |                   |               |               |
|Room_15       |Lamp_27 |actuator_31 |switchOn  |            |                   |               |               |
|Room_16       |Lamp_28 |actuator_32 |switchOff |            |                   |               |               |
|Room_16       |Lamp_28 |actuator_32 |switchOn  |            |                   |               |               |
|Room_17       |Lamp_29 |actuator_33 |switchOff |            |                   |               |               |
|Room_17       |Lamp_29 |actuator_33 |switchOn  |            |                   |               |               |
|SmartHome_1   |Room_14 |            |          |sensor_18   |getPresenceState   |Publisher_22   |PresenceEvent  |
|SmartHome_1   |Room_15 |            |          |sensor_19   |getPresenceState   |Publisher_23   |PresenceEvent  |
|SmartHome_1   |Room_16 |            |          |sensor_20   |getPresenceState   |Publisher_24   |PresenceEvent  |
|SmartHome_1   |Room_17 |            |          |sensor_21   |getPresenceState   |Publisher_25   |PresenceEvent  |
--------------------------------------------------------------------------------------------------------------
```

Figure 4.13: SPARQL query result

To summarise, the key benefits of Ont4SRE are as follows.

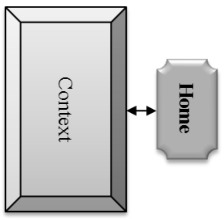
- It enables the construction of a formal model of the SRE that helps computers and programmers to have a good understanding of the smart components that might occupy an SRE and shows their relationships to their associated devices, services, and locations.
- With the Ont4SRE ontology, the different smart entities and services share the same axioms as these are specified in the ontology. This ensures consistency between the different components.
- The Ont4SRE ontology offers a semantic description of the representation of the device's capabilities whereby the device's services content can be marked up with ontology-based semantic annotations, as will be shown in the next section.
- The Ont4SRE ontology can be used as a database store for the registry and discovery of smart entities. As will be shown in Section 4.6, the Ont4SRE ontology helps the discovery engine to construct a representation of a real SRE, including smart entities, actuators, sensors, services, and locations.
- The proposed ontology helps, as outlined in this thesis, to create a domain specific language for describing control scenarios using a precise vocabulary without having to deal with program intricacies such as services grounding, event registry, and services discovery.
- The Ont4SRE enforces the loose coupling between the different models in the framework, so change in one model will not affect the others.
- The Ont4SRE is pluggable. Different ontologies for different domains (home, school, and hospital) can be used within the framework once the appropriate service providers are available.
- The Ont4SRE ontology can be easily updated to add new smart entities, sensors, actuators or services.

4.4.4 Ontology Mapping

Using a single, global ontology to represent all services is not realistic to support the tasks envisaged by a distributed environment like a SRE. Consequently, ontology mapping aims to provide a common layer from which several applications could access multiple ontologies and exchange semantic information. Developing such mappings has been the focus of a variety of work originating from diverse communities over a number of years. The interested reader is referred to [139] for a good survey of ontology mapping techniques, as an in-depth discussion is outside the scope of this thesis.

4.5 Services Accessibility (Context)

Referring to Figure 3.1, at the low level (physical & communication layers), many industrial approaches [140] have proposed a number of protocols to help integrate embedded devices into a distributed network. At this level, recent research [141-143] has focused mostly on improving communication, data coding efficiency and power consumption.



At a higher level (service layer), several works have proposed middlewares to address interoperability issues in SRE using technologies like OSGi [13, 59, 60, 144], WS-* web service [5-7, 41, 145, 146] and RESTful web service [9-11, 52, 147]. These are deployed on the top of the low-communication protocols to provide a service abstraction and programming interface, which standardizes the accessibility and helps data sharing. Several researches results like in [65, 67, 148] have proved that applying these technologies to embedded devices could be achieved and could improve them in spite of their hardware limitations in term of memory, code space and communication bandwidth.

Other efforts go in a different direction, to establish standards enabling the syntactic and semantic description of the services layer. Exploiting web services is difficult using only the syntactic descriptions. Semantic-based approaches like [14, 16, 96, 149] use semantic web languages like OWL-S, and SAWSDL to provide semantic awareness and add machine interpretable information to services content which simplifies the service discovery and filtering capabilities.

To *enable a consistent access to all components of a SRE,* the web services (WS-*) architectural style were adapted and a semantic description for each web service was provided in order to support low-level gathered data with a high-level of abstraction (see Figure 4.14).

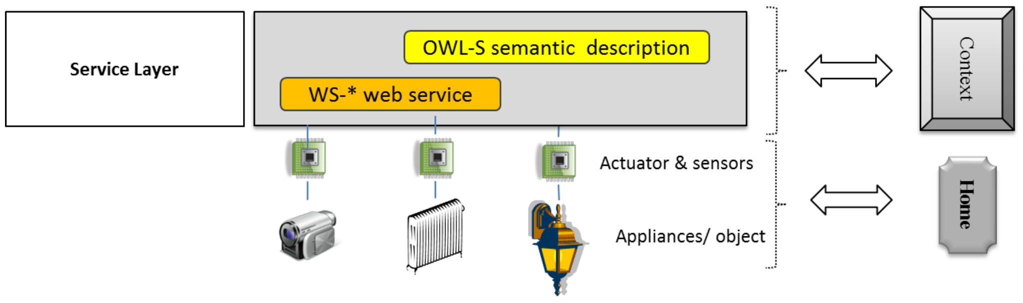


Figure 4.14: Service layer based on WS-* web service and OWL-S semantic description

4.5.1 Web Services (WS-*)

By integrating the WS-* web service into the GF4SRE framework, smart entities can publish their functionalities in the form of actions, queries and events via a web services description language (WSDL) interface. A web service may contain one or more operations to be performed. According to the Ont4SRE ontology model, two kinds of operation are distinguished, as depicted in Figure 4.15.

- `action` operation represents a web service operation to be performed by an actuator to modify the smart entity's state. In Example 3.1, to adjust the air conditioner temperature, a web service `setTemperature(parameter)` is used. It represents an action named `setTemperature`, which has one input called `parameter`.
- `query` operation represents a web service operation used to retrieve and read the current state of a smart entity via a sensor. The result of calling up this operation is the current state of the smart entity. For example, to get the room temperature, we can use an operation called `getTemperature`. This represents a query named `getTemperature` without any input parameters and which returns the room temperature.

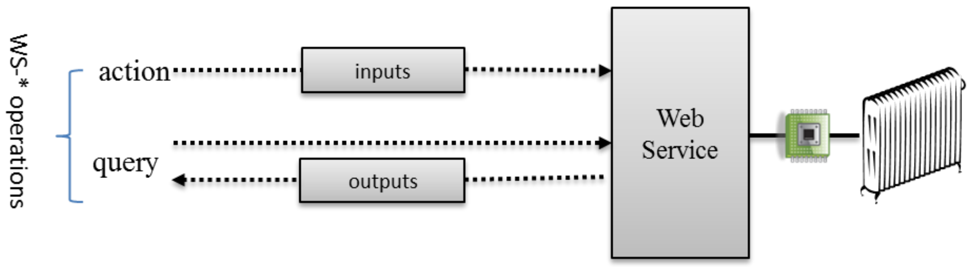


Figure 4.15: Operation types for a web service

4.5.1.1 Events Notification

An event represents a callback mechanism enabling a software system to be informed about changes in the environment rather than asking at specific intervals in time if any significant change has occurred. The change may be considered to have occurred only if a predefined threshold value has been crossed (e.g. after increase in a certain number of degrees in temperature). In the GF4SRE framework, the orchestration process subscribes to an event manager to be notified when a new event is produced. Each time a new event is produced, the corresponding sensor's publisher fires a new update to the event manager, which, in turn, receives the incoming message and notifies all the subscribers about the occurrence of the event. The received message may contain supplementary information such as the time when the event occurred, or the geographical location where the event was produced, etc.

As mentioned in Subsection 3.3.3.5, using event notifications raises scalability by minimizing the HTTP calls and reduces the energy consumption of the smart entities. Further, it reduces the dependencies between framework components and therefore increases reusability and flexibility. The main idea behind the event notification mechanism applied in the GF4SRE framework is shown in Figure 4.16, and explained below.

1. The **Event Publisher** is the event resource responsible for generating and sending events to the event manager.
2. The **Event Manager** component contains the following elements:
   - Definitions of interested events. Each event is defined with the **`event name`** and its related parameters and the data types exchanged via this event.
   - A declaration of the involved events publishers.
   - Remote Event Connection: There are different ways to publish an event to the Event Delivery Network (EDN) remotely (e.g. using Java API or WS-Eventing specification). The event manager should listen to each interested event. There are two ways for the event manager and the event publisher to discover each other.
     - *Manual configuration*: the event publisher may explicitly be pre-configured with the address to which it should publish the event.
     - *Dynamic discovery*: when the device joins the network, it announces its arrival via a multicast **`Probe`** message. Then, the event manager receives the multicast **`Hello`** messages, stores the information about the event publisher and start listening to it.
3. The **Event Subscriber** that is interested in a particular event should subscribe to being notified.

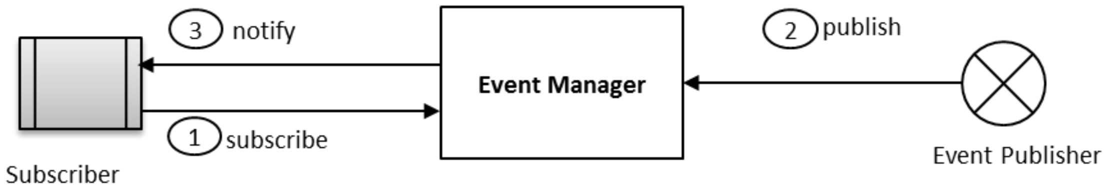


Figure 4.16: Event notification process

4.5.1.2 **Alternative solution for deploying WS-***

Because the computational memory and the communication bandwidth of actuators and sensors are limited, WS-* can be implemented as an additional attached hardware component (called a proxy) rather than implementing it directly on actuators and sensors.

4.5.1.3 **Benefits of using Big Web Services (WS-*)**

The WS-* choice is based on the followings arguments:

- WS-* web service is a well-known standard supported by a large research community.
- Research results have found that applying this technology on embedded systems is possible.
- WS-* provides a common interface written in a standard machine-readable format (WSDL, XSD), so applications can access, consume, and reuse these services in a uniform way.
- Further, WS-* standards support many features such as addressing, WS-Eventing, discovery and services composition.
- Many applications and development tools like Oracle SOA Suite [91] support using WS-* services with composition language standards like BPEL4WS.
- Many semantic web technologies have been developed to build a semantic layer on top of WS-* standards such as OWL-S, SAWSDL.
- Although, the RESTful web service appears as an alternative approach to design web services, with the later, everything is seen as a resource, identified by URL and has a uniform interface (DELETE, POST, GET, and PUT), whereas WS-* web services are written in a standard WSDL format to specify operations and their parameters which is useful for functionality modelling in SRE.

4.5.2 **Semantic Annotations of the Web services**

On top of the service layer, an additional level of abstraction is introduced. Instead of giving only a syntactic description of the services provided by the devices (sensors, actuators), each service (action, query, event) is marked up with a semantic description to describe its capabilities. The different service interfaces are mapped onto the Ont4SRE ontology to provide common semantic equivalences. The semantic description of a web service includes the following elements.

- A semantic meaning for each WSDL operation name corresponding to its name (action, query, or event) in the Ont4SRE.

- A semantic description of each input parameter and its data type as required by each web service operation as well as each output parameter with its data type, which is returned by calling a web service operation.
- Grounding details to be able to invoke the web service, the grounding include providing a URI address for each WSDL service and the necessary concrete details related to the transport protocol and message format.

The semantic proposal in the case study which will be introduced in Chapter 6, uses OWL-S technology, which has well-defined constructs for describing the properties and capabilities of web services. Each implemented web service provides the appropriate OWL-S documents. Another similar, technology like SAWSDL, can be implemented in the framework for the same proposal.

Example 4.2

Figure 4.17 shows an example of a mapping between a web service and its semantic descriptions. On the right side of the figure, a lamp controller provides concrete implementation of a web service with an operation named **`adjustLampBrightness`**, which has an input named **`parameter`** and returns an output **`return`** of type Boolean to confirm the modification of the lamp's brightness level. On the left side is the ontology instantiated for a given real lamp (**`lamp_1`**) with its actuator (**`LampController_1`**) and service (**`adjustBrithness_1`**) which provides the corresponding semantic description for each of the web service elements. At the bottom of the figure, the OWL-S construct is used to map the right and left sides (i.e. **`adjustLampBrightness`** vs. **`adjustBrightness`**, **`parameter`** vs. **`level`** and **`return`** vs. **`confirmation`**), and the necessary details about the web service binding (i.e. transport protocol, WSDL address, etc.). These relationships are expressed in OWL-S language and saved in an RDF format.

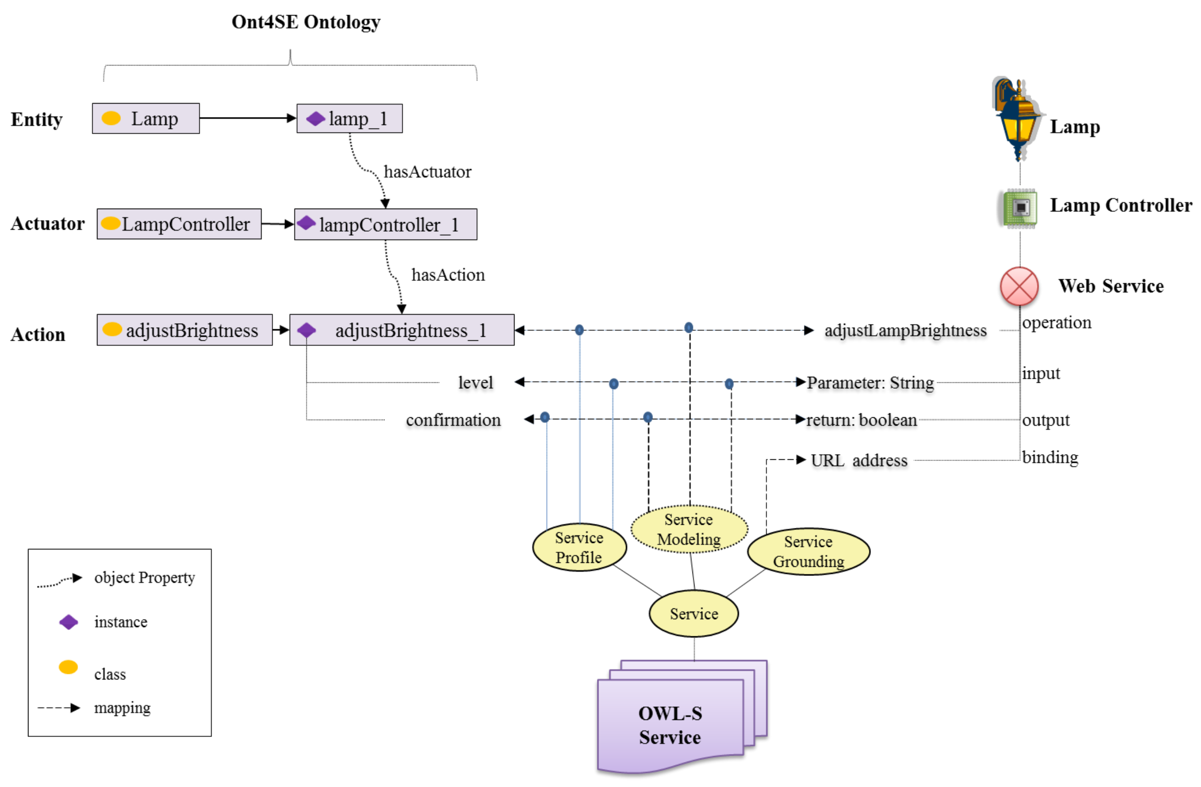


Figure 4.17: Ontology-based semantic annotation for web services

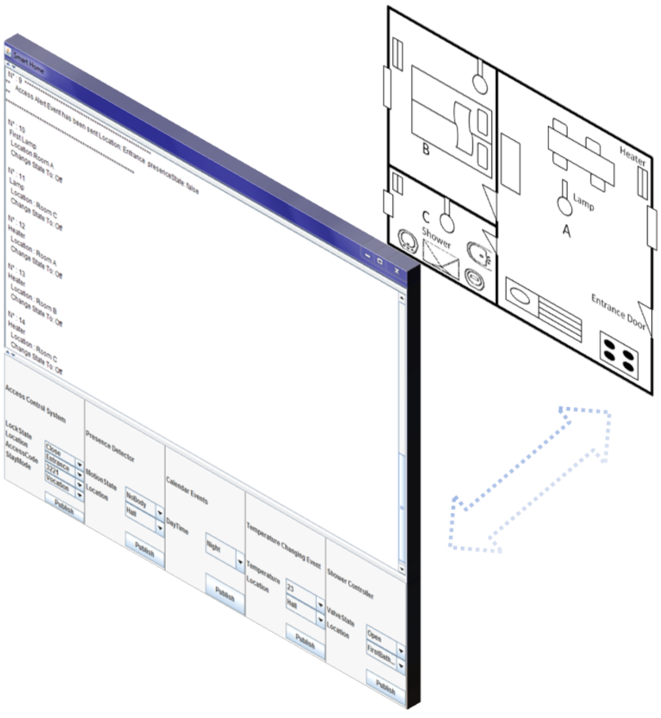
4.5.2.1 Key Benefits of Using Semantic Annotations

Using the semantic annotations, add to the GF4SRE framework the following features.

- Add machine interpretable information to services content in order to understand the capabilities of web services that have different original purposes.
- Give a unified presentation for similar web services and their properties and parameters.
- Distance the user from the complexity of device platforms by hiding the implementation details related to each web service - like its grounding details - which are transparently mapped.
- Help the discovery of the smart entities' capabilities by mapping and linking the different smart entities with their associated services.
- Simplify the completion of complex control scenarios using the web services.

4.5.3 Implementation of the Smart Residential Environment (Context)

The implementation of the smart residential environment consists of simulating the case study, which will be introduced in Chapter 6. It contains a set of smart entities occupying a smart home where each smart entity is simulated and associated with one or more virtual devices (actuators or sensors) with one real implemented web service interface that provides its services (queries, events and actions). The evolution of service invocations and the produced events are visualized in a graphical user interface as shown Figure 4.18. The reader is referred to Subsection 6.2.1 for more details.



```
N° : 9 ****************************************
** Access Alert Event has been sent Location: Entrance presenceState: false
****************************************
N° : 10
First Lamp
 Location Room A
 Change State To: Off
N° : 11
Lamp
 Location : Room C
 Change State To: Off
N° : 12
Heater
 Location : Room A
 Change State To: Off
N° : 13
Heater
 Location : Room B
 Change State To: Off
N° : 14
Heater
 Location : Room C
 Change State To: Off
```

Access Control System — LockState: Close; Location: Entrance; AccessCode: 3221; StayMode: Vocation; Publish

Presence Detector — MotionState: NoBody; Location: Hall; Publish

Calendar Events — DayTime: Night; Publish

Temperature Changing Event — Temperature: 23; Location: Hall; Publish

Shower Controller — ValveState: Open; Location: FirstBath...; Publish

Figure 4.18: Smart home simulation

4.6 Registry and Discovery Engine

With the framework components introduced in the previous sections, the smart environment was allowed to provide a service-based architectural layer with semantic descriptions and an ontology-based model for describing it. This section describes how to register, search, and find the relevant services via a registry & discovery mechanism to create - as will be shown latter - complex control scenarios. In the GF4SRE framework, this happens via the *Registry & Discovery Engine* component. Existing solutions like UDDI, UPnP and Jini have been proposed to enable the service discovery for devices and hardware components [150-152], but discovery techniques are still based on the syntactic details of services. They enable explicit services to be found without exploiting the benefits of semantic service discovery capabilities. Several semantic-based discovery approaches have shown that adding semantic information to syntactical service definitions through the use of an ontology can help interpret the purpose and usage of those services [33, 77]. For further reading, [111] and [153] give a comparative review of services discovery using semantic annotations techniques for ambient environments.

In the SRE context, searching for smart objects is different from searching for concrete services on the internet since each service is associated with a given smart entity offered by a given sensor or actuator in a specific location. In a SRE, services have effects on their providers, where they may change or monitor their states. The benefits for the service requestors, in this case, are implicitly acquired by adapting the environment to his needs. Therefore, it is important for the programmer and the computing systems to have a good understanding of each service provider (e.g. lamps, heaters, etc.) and their properties in order to understand the effects of each service on them and on the environment. As far as this thesis is concerned, the discovery process adapted here is based totally on this vision. The registry and discovery module proposed are based on the Ont4SRR ontology, where smart entities, locations, associated devices and offered services are modelled and linked together.

The Registry & Discovery Engine is composed of Ont4SRE used as a database store, a registry, and discovery unit, as shown in Figure 4.19. The registry component allows sensors and actuators to announce themselves and their services, while the discovery component allows applications (consumers) to search for specific smart entities and their services. In the following, the role and concepts of the two units are presented:

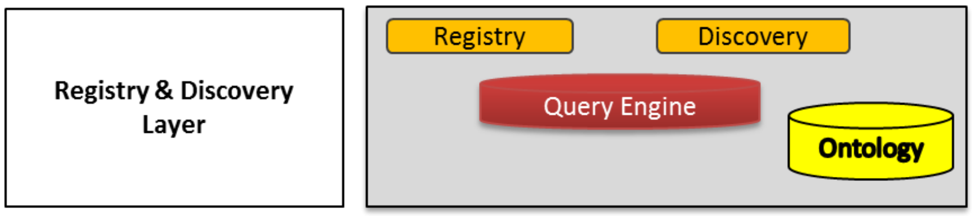


Figure 4.19: Registry and discovery layer and related components

4.6.1 Services Registry

The service registry is the process of specifying which and where smart entities and services are available in a particular SRE. This keeps track of what can be discovered and used in the SRE. The service registry process is performed in two steps as follows.

4.6.1.1 Ontology Instantiation (Step 1)

For a given smart building, the configuration of the environment corresponds to an instantiation of the ontology classes related to smart objects occupying the space. The instantiation is based on describing and representing the smart environments along six types of ontology-based required information, as shown in Figure 4.20.

- ***Smart Entities***: instantiate the ontology classes related to smart entities occupying the space.

- ***Locations***: instantiate the ontology classes related to different real locations. Each smart entity is linked to a location in the physical world.
- ***Sensors***: instantiate the ontology classes related to sensors for each corresponding smart entity in order to link the latter with its services and to perform queries via its sensors. The instantiation includes providing the **`sensorID`** value for each sensor.
- ***Actuators***: instantiate the ontology classes related to actuators for each corresponding smart entity in order to link the latter with its services and to perform actions via its actuators. The instantiation includes providing the **`actuatorID`** value for each actuator.
- ***Publishers***: instantiate the ontology classes related to publishers for each corresponding sensor belonging to a smart entity instance.
- ***Services***: instantiate the ontology classes related to different services (query, action, and event) for each corresponding sensor or actuator belonging to a smart entity instance. This gives a list of the services and their required parameters that a smart entity offers.

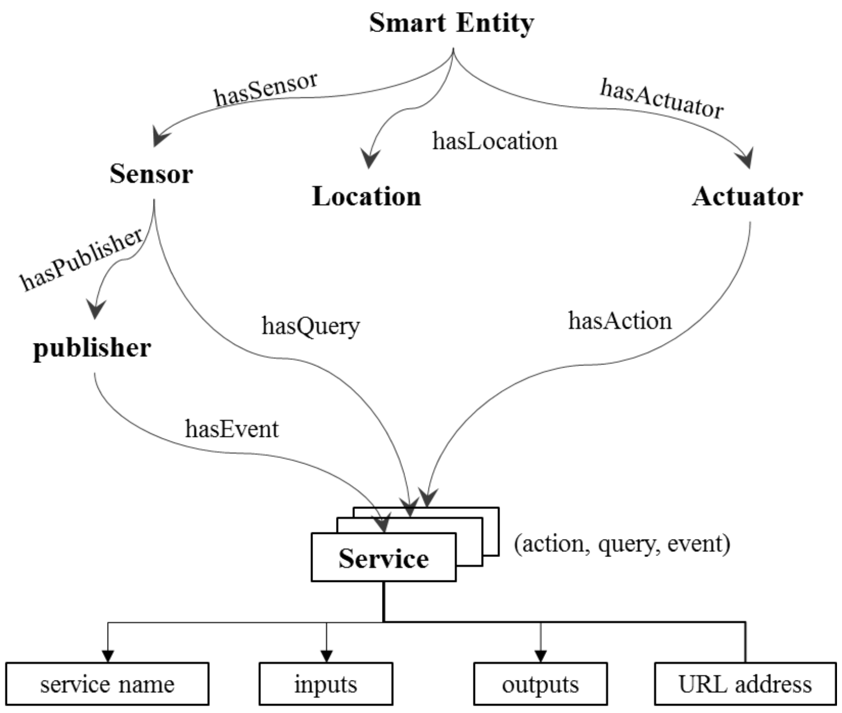


Figure 4.20: Ontology required information for the instantiation of each smart entity

Example 4.3

Listing 4.3 shows an extract of the Ont4SRE ontology instantiated within Example 4.1, which contains an instance **`lamp_1`** of the class **`Lamp`** and is located in **`room_1`**. The lamp has an actuator **`lampController_1`** and offers two actions **`switchOn`** and **`switchOff`** while the **`room_1`** has a sensor **`presenceSensor_1`**, associated with a publisher **`presencePublisher_1`** to detect if somebody enters the home in the form of an event **`PresenceEvent`**. The code is organized as follows.

- Lines 1-2 define that the instance **`room_1`**, which is a type of the class **`Room`**, and has an **`entityID`** equal to **`Room_A`**.
- Lines 3-4 define that the instance **`room_1`** has entity **`lamp_1`**.
- Lines 4-6 define that the instance **`lamp_1`** has an actuator instance **`lampController_1`**.
- Lines 7-12 define that the instance **`lampController_1`** has **`switchOff`** and **`switchOn`** actions.
- Lines 16-27 define that the instance **`room_1`** has a sensor instance **`presenceSensor_1`** with an event publisher **`presenseEventPublisher_1`**.

```
1    <diuf:Room rdf:ID="room_1">
2      <diuf:entityID rdf:datatype="#string">Room_A</diuf:entityID>
3      <diuf:hasEntity>
4        <diuf:Lamp rdf:ID="lamp_1">
5          <diuf:hasActuator>
6            <diuf:LampController rdf:ID="lampController_1">
7              <diuf:hasAction>
8                <diuf:switchOff rdf:ID="switchOff_1">...</diuf:switchOff>
9              </diuf:hasAction>
10             <diuf:hasAction>
11               <diuf:switchOn rdf:ID="switchOn_1">...</diuf:switchOn>
12             </diuf:hasAction>
13      </diuf:hasActuator>
14    </diuf:Lamp>
15    </diuf:hasEntity>
16    <diuf:hasSensor>
17        <diuf:PresenceSensor rdf:ID="presenceSensor_1">
18          <diuf:hasPublisher>
19            <diuf:PresenseEventPublisher rdf:ID="presenceEventPublisher_1">
20              <diuf:hasEvent>
21                <diuf:PresenceEvent rdf:ID="presenceEvent_1">...
22                </diuf:PresenceEvent>
23              </diuf:hasEvent>
24            </diuf:PresenceEventPublisher>
25          </diuf:hasPublisher>
26      </diuf:PresenceSensor>
27    </diuf:hasSensor>
28   </diuf:Room>
```

Listing 4.3: Code extract of a smart lamp instantiation and its corresponding services

4.6.1.2 Installation of Actuators and Sensors (Step 2)

After partially instantiating the ontology (without grounding), the final registry process is started by an actuator or a sensor associated with a smart entity. First, it sends a request to be added to the service providers. Then, the registry engine identifies the device instance using its unique identifier, which can be, for example, the physical address of the device (e.g. MAC address). The registry engine extracts and updates the received information to their corresponding device instances in the ontology, which includes a list of services, the semantic description document, and their corresponding URL addresses. There are two ways for sensors and actuators to find the registry and discovery engine the first time.

- *Manual configuration*: the sensor or actuator may explicitly be pre-configured with the address to which it should register itself and its services.
- *Dynamic discovery*: when the device joins the network, it announces its arrival via a multicast **`Probe`** message. The registry unit then receives the multicast **`Hello`** messages, and stores the information about the devices and their related services.

Example 4.4

An extract of an expected web service request sent by a **`Lamp Controller`** actuator is given in Listing 4.4. It shows that the actuator is identified by an **`actuatorID`** equal to **`4754`**, and registers a **`switchOn`** service.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body xmlns:ns1="http://ws/">
  <ns1:registre>
   <deviceID>4754</deviceID>
   <serviceName>switchOn</serviceName>
    <owlURL>http://diufpc10:8080/.../LivingRoomLamp.owl</owlURL>
    <wsdlURL>http://diufpc10:8080/.../LivingRoomLamp?WSDL</wsdlURL>
  </ns1:registre>
 </soap:Body>
</soap:Envelope>
```

Listing 4.4: SOAP message sent by a LampController actuator

4.6.2 Services Discovery

The discovery component assumes responsibility for offering a query interface for finding relevant smart entities. Once the ontology has been instantiated, all the sensors and actuators are installed, and services are registered, so the Ont4SRE ontology can be used to send queries about the smart entities. The queries in the discovery unit are written in the SPARQL language. The latter allows the formation of different queries similar to what the SQL language does. With the GF4SRE framework, queries are formulated based on the following given information.

- ***Smart Entity Type*** specifies the type of the requested smart entity (e.g. **`Lamp`**, **`Door`**).
- ***Smart Entity Identifier*** (optional) specifies the **`entityID`**.
- ***Location*** (optional) specifies where the requested smart entity should be located.

The three previous parameters determine the predicates of the query. Thanks to the power of the SPARQL language, and based on these predicates, the query could be further formulated in order to know which services and devices are linked to each smart entity.

In the GF4SRE framework, the discovery query interface provides four types of query when looking for existing smart entities.

1. **Full Query:** this query is performed to return a list of all the available smart entities and their related devices and services that match a specific type of smart entity (e.g. find all smart entities of type `Lamp` in the building). This can be useful, for example, when a control system decides to switch off all the lamps. In this case, the action `switchOff` can be performed by iterating over each lamp in the discovered list. Listing 4.5 is a full query example written in SPARQL to discover the smart lamps and their associated services. This extract of code is organized as follows:
   - Lines 1-3 define the PREFIX keywords which associate prefix labels with URI.
   - Line 4 defines the clause `SELECT` to return a result set of records (rows). Each row contains the smart entity, its location, actuators, sensors, publishers and their associated services: action, query and event.
   - Lines 8-36 declare the clause `WHERE` to filter the results to retrieve. The criteria are expressed in the form of predicates.
   - Line 9 finds the smart entities of type `Lamp`. The subject is *?entity*, the predicate is `rdf:type` and the object is `?entityType`.
   - Line 10 retrieves the name of the entity.
   - Line 11 uses method `Filter` to select only the smart entity of type `Lamp`.
   - Lines 12-13 determine the location of the entity.
   - Lines 14-35 use the UNION operator to combine the results of three condition sets.
     - In the first condition set, lines 14-20 specify that each entity instance must have an actuator and each actuator must have an `actuatorID`, and an action. Finally, each action has a WSDL and OWL-S URL address.
     - In the second condition set, lines 22-28 specify that each entity instance must have a sensor, `sensorID`, and a query. Finally, each query has a WSDL and OWL-S URL addresses.
     - In the third condition set, lines 30-35 specify that each entity instance must have a sensor, and a publisher. Each publisher must have an event.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX diuf: <http://www.unifr.ch/owl/2011/03/HomeOntology.owl#>
SELECT  ?entityName ?entityType ?locationID ?actuator ?actuatorID
        ?sensor ?sensorType ?sensorID ?publisher
        ?actionType ?actionWsdlAddress ?actionOwlsAddress
        ?queryType ?queryWsdlAddress?queryOwlsAddress ?eventType
WHERE {
  ?entity rdf:type ?entityType.
  ?entity diuf:entityName ?entityName.
  filter (?entityType = diuf:Lamp).
  ?entity diuf:hasLocation ?location.
  ?location diuf:locationID ?locationID
  {   ?entity diuf:hasActuator ?actuator.
      ?actuator diuf:actuatorID ?acutatorID.
      ?actuator diuf:hasAction ?action.
      ?action rdf:type ?actionType.
      ?action diuf:wsdlAddress ?actionWsdlAddress.
      ?action diuf:owlsAddress ?actionOwlsAddress.
  }
  UNION
  {   ?entity diuf:hasSensor ?sensor.
      ?sensor rdf:type ?sensorType.
      ?sensor diuf:hasQuery ?query.
      ?query rdf:type ?queryType.
      ?query diuf:wsdlAddress ?queryWsdlAddress.
      ?query diuf:owlsAddress ?queryOwlsAddress.
  }
  UNION
  {?entity diuf:hasSensor ?sensor.
   ?sensor diuf:hasPublisher ?publisher.
   ?sensor diuf:sensorID ?sensorID.
   ?publisher diuf:hasEvent ?event.
   ?event rdf:type ?eventType.
  }
}
order by (?entity)
```

Listing 4.5: A SPQRL Full Query to discover all the smart lamps in the space

2. **Location-based-Full Query**: this kind of query is used to return a list of all the available smart entities that exist in a given location. In a smart space, the context of the environment and the end user's needs are different from one place to another. For example, in a security scenario, if there is nobody at home, leaving some lights on during the night in some places is a good idea to give the appearance the home is occupied. Similarly, the heating for the pool or garage can be turned off when not being used in order to reduce the total energy consumption. The SPARQL query for this kind of request is similar to the Full query represented above with the following additional condition statement where the filtering is based on the `locationID` provided by the requestor (e.g. `Room_A`, or `Room_B`).

```
filter(?locationID = 'Room_A').
```

3. **EntityID-based Query**: this kind of query is used to return one smart entity with a known identifier which can be used, for example, to control a specific smart entity like an

entrance door or a room rather than a list of doors and rooms. In this case, the discovery engine will apply a Full query and return one smart entity from the list. The SPARQL query for this kind of request is similar to the Full query represented above with the following additional condition statements where the filtering is based on the **`entityID`** provided by the requestor (e.g. **`Lamp_A`**, or **`Lamp_B`**).

```
?entity diuf:entityID ?entityID.
filter(?entityID = 'Lamp_A').
```

4. **Minimum-Cardinality Query:** this kind of query is used to return one smart entity. A typical example is when the programmer knows that there is only one smart entity of a given type occupying a particular space like a **`FireAlarm`** or he needs only one smart entity of a given type to perform a given action like sending an SMS message via a **`Telephone`**. In this case, the discovery engine will apply a minimum-cardinality query and return one smart entity from the list.

Example 4.5

Consider the same home ontology instance presented in Example 4.1 consisting of four room instances, each containing a lamp instance which provides a **`switchOn`**, and **`switchOff`** service via a **`LampController`** actuator.

To get a list of all lamp instances in each room, the discovery process is performed in the following order.

1. First, the client (e.g. process) sends a request to the discovery engine asking for all the smart entities of type **`Lamp`**.
2. Then, the query engine component retrieves the request, initiates a full query using the SPARQL language, and executes it.
3. Finally, it returns a list of the discovered smart entities, their associated devices and services. The response is an XML message, as presented Listing 4.6.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<smartEntitiesHolder>
    <entityType>Lamp</entityType>
    <smartEntity xmlns:xsi="http:" xsi:type="smartEntity">
        <entityType>Lamp</entityType>
        <locationID>A</locationID>
        <entityID>lamp_4</entityID>
        <name>Lamp_4</name>
        <actuator xsi:type="actuator">
            <actuatorID>a316</actuatorID>
            <actuatorName>LampController_5</actuatorName>
            <action xsi:type="action">
              <OWLSAddress>http://.../SwitchOffHallLamp.owl</OWLSAddress>
              <WSDLAddress>http://.../SwitchOffHallLamp.wsdl</WSDLAddress>
              <name>switchOff</name>
            </action>
            <action xsi:type="action">
               <OWLSAddress>http://.../SwitchOnHallLamp.owl</OWLSAddress>
               <WSDLAddress>http://.../SwitchOnHallLamp.wsdl</WSDLAddress>
               <name>switchOn</name>
            </action>
        </actuator>
    </smartEntity>
    <smartEntity xmlns:xsi="http:" xsi:type="smartEntity">
        <entityType>Lamp</entityType>
        <locationID>B</locationID>
        <entityID>lamp_1</entityID>
        <name>Lamp_1</name>
        <actuator xsi:type="actuator">
           <actuatorID>a318</actuatorID>
           <actuatorName>LampController_2</actuatorName>
           <action xsi:type="action">
            <OWLSAddress>http://.../SwitchOffParentLamp.owl</OWLSAddress>
            <WSDLAddress>http://.../SwitchOffParentLamp.wsdl</WSDLAddress>
            <name>switchOff</name>
           </action>
           <action xsi:type="action">
             <OWLSAddress>http://.../SwitchOnParentLamp.owl</OWLSAddress>
             <WSDLAddress>http://.../SwitchOnParentLamp.wsdl</WSDLAddress>
              <name>switchOn</name>
           </action>
        </actuator>
    </smartEntity>
...
</smartEntitiesHolder>
```

Listing 4.6: A simplified result of SPARQL full query for the discovered smart lamps

4.6.3 Key Benefits of Discovery & Registry Engine

Using the Discovery and Registry engine adds the following features to the GF4SRE.

- Enabling smart entities to be registered and discovered.
- Using semantic-based discovery solves syntactic matching challenges in smart environments due to the unification of the services' semantic descriptions.
- Ontology Instantiation is done for one time and updating is done dynamically.
- The benefits of using SPARQL are:

- It enables the automatic discovery of smart entities.
- Its ability to take into consideration the Ont4SRE class relationships where smart environment concepts are defined and linked together. This enables each smart entity to be linked to its devices, location and services.
- SPARQL queries help the classifying and grouping of services, depending on their types or locations.
- SPARQL queries can be further formulated to provide more specific conditions and requirements for requested smart entities.

4.6.4 **Implementing the Registry and Discovery Engine**

The Registry & Discovery Engine is implemented using the Ont4SRE ontology, WS-* web service, and Java API. As shown in Figure 4.21, it consists of the following software components:

- The *Ontology Repository (Ont4SRE)* component serves as a database store for SRE instantiation and devices registry and discovery.
- The *Query Engine (SPARQL)* is used to query the Ont4SRE ontology.
- The *Registry Unit* is responsible for registering each sensor or actuator joining the network.
- The *Discovery Unit* is responsible for discovering the relevant smart entities requested by clients.
- The *Query Interface* is a set of web services available for requestor to query about smart entities or register new devices. Both the registry and discovery units communicate with their clients via the query interface.

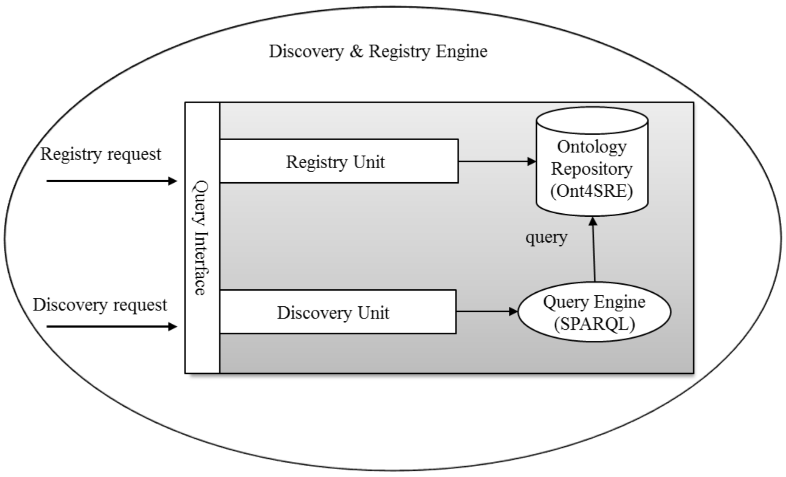


Figure 4.21: Service registry and discovery units: physical model

Two kinds of queries allow access to the Registry and Discovery Engine:

- ***Registry Enquires*** are used to update and store information in the Ont4SRE ontology. The following is a summary of the main implemented inquiries:

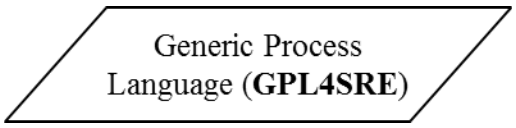
- A registry query to add a newly installed device. In this kind of query, the device sends a request to be added. The registry engine identifies the device instance using its identifier and adds it to the Ont4SRE.
- A query to update an existing device. This kind of query can be used to update the device state (i.e. online or offline).

- ***Discovery Enquires*** are used to retrieve information from the Ont4SRE ontology. The implemented queries are related to the four queries introduced in Section 4.6.2 for the discovery and selection of the smart entities.

4.6.4.1 Smart Residential Environment Visualization

A graphical interface has been developed to visualize and present in real time the registered smart entities with their associated devices and services graphically. In order to create the smart entities tree, as represented in Figure 6.4, the SPARQL query was used to select all distinct smart entities with one or more actuators or/and sensors and all their related properties such as their location, and services. For more details see Subsection 6.2.3.

4.7 The GPL4SRE Language: Overview



The Ont4SRE ontology, Services Accessibility (Context), and Registry & Discovery Engine components introduced in the previous sections allowed the modelling, access and searches for smart entities and their services in a given smart space. In this section, developers will be able to create simple and complex control scenarios on top of the smart objects. Whatever the technique used for managing SRE (manual, automatic or semi-automatic), it needs a programming language to express the composition algorithm. A number of tools and languages have been proposed by the software industry to support services composition in the smart environment context [106-108]. However, creating business processes with these tools and languages through an XML editor is complex and error prone. [154] proposes a macro language that extends Java in order to help developers create context-aware applications. [155] proposes an agent-based domain specific language for controlling appliances and providing a single interface for each device. However, it does not efficiently support the management of events.

Compared to existing composition languages like BPEL4WS [18] and BPMN [89], the proposed domain specific language (DSL) based on BPEL4WS is designed to be used for the orchestration of the SRE. It supports a high degree of abstraction and allows users to define processes in a more compact and comprehensible way. It provides interaction ability with final users and generic (declarative) definition of the involved smart entities and their services using semantic-based service description, rather than using WSDL description interfaces. The discovery of services and the event registry are completely automatic. The GPL4SRE language is the link between the different elements of the GF4SRE framework infrastructure (i.e., Ont4SRE, Context, Registry & Discovery Engine, and the execution environment).

4.7.1 GPL4SRE Program Structure

The GPL4SRE is a workflow-based language characterized by an imperative programming style. Diverse control scenarios can be simply written with GPL4SRE language. It is possible to discover and interact with different smart entities, create loops, declare variables, copy and assign values, as well as register and wait for events from sensors' publishers. Scenario execution can be sequential or parallel. The language syntax is explained in detail in Chapter 5. This section gives a description of the concepts and structure of a program written in the GPL4SRE language. As shown in Figure 4.22, a program written in GPL4SRE has two main sections: a declaration section and an execution section.

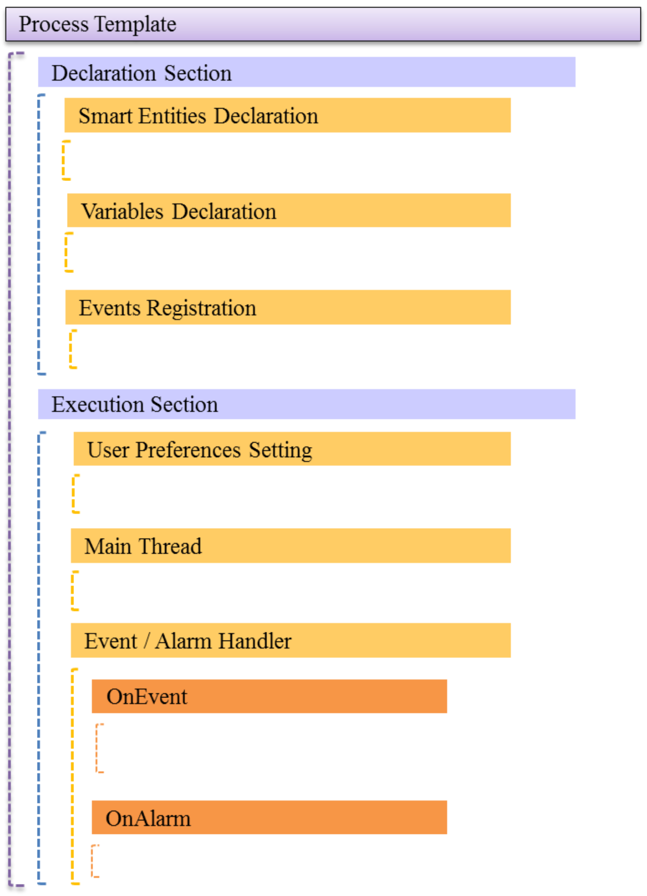


Figure 4.22: GPL4SRE elements structure

4.7.1.1 Declaration Section

The declaration section deals with the declaration of the smart entities, the registry of the events and the declaration of variables.

- ***Smart Entities Declaration Part***: allows for the declaration of the smart entities that will interact with the process. Smart entities can be declared individually or as a list of similar entities. Once a smart entity is declared, it can be controlled through its associated services. The smart entities are declared implicitly in regard only to their type and location (optional), while the discovery and services grounding are fully hidden and automated by sending a request using the **`discover`** keyword. Once the request is received by the discovery engine, it is automatically translated into a SPARQL query to find and send back a list of the discovered smart entities.
- ***Events Registration Part*** allows subscribing the process to events that it is interesting in. The event subscription is performed in implicit manner with regard only to the event type (e.g. **`MotionEvent`**, **`TemperatureChangingEvent`**), while subscribing the process to the Event Manager is fully automatic using **`subscribe`** as the keyword.
- ***Variables Declaration Part***: allows the declaring of variables in order to receive manipulate and send data between the smart entities and the process. Variable can be global or local. The programmer is allowed to declare primitive data types such as Int, Float, and Boolean. It is possible also to declare complex variables in a Map collection form (i.e. a list of (name, value) pairs).

4.7.1.2 Execution Section

The execution section specifies the parts that will be executed when the process is running. It includes the following parts.

- ***User Preferences Setting Part***: this plays the role of the initializer to customize the process according to the final user requirements. The process provides **`set`** construct to assign user preferences to variables, a **`select`** construct to select, according to the final user, which smart entity should be involved in the scenario, and a **`print`** construct to display information to the user through a simple interface.
- ***Main Thread Part***: specifies the first part of the process that runs when the process starts after the initialization of user preferences. This part can be used for the following proposals.
  - Initialize variables.
  - Perform a combination of actions at regular intervals (time-driven fashion).
  - Perform a combination of action when an event is produced (event-driven fashion), especially when a process reacts to an event over a short period of time.

- Determine the lifetime of the process. The process terminates when this part completes its execution.

- ***Event / Alarm Handlers Part*** are associated with an enclosed scope. This allows the process to respond to events or alarms (an expiration of a timeout) which occur independently of the execution of the main thread and are produced at any moment during the process's lifetime. The keyword **`OnEvent`** is used to declare the beginning of an event handler branch, and **`OnAlarm`** for the beginning of an alarm handler branch.

4.7.2 Main Constructs

After the introduction of the main structure of a GPL4SRE program, this subsection presents some of the main constructs used within the process execution parts.

- ***Control flow Constructs*** define the program's procedural logic. The important ones are **`for`**, **`while`** and **`if`**.
- ***Execution Constructs***: the workflow execution can be sequential or parallel by using the keywords **`sequence`** and **`flow`**, respectively. Other execution constructs, inspired by BPEL4WS, are also provided, such as the **`pick`** construct to give a set of choices of possible sets of actions.
- ***Service Interaction Constructs*** allow interacting with the smart entities. There are three types:
  1. An **`action`** construct is used to perform an action to modify the state of a smart entity. Three elements are required to perform an action: the target smart entity, the action's name and the input message to be sent which contains the new value of the smart entity state. The action name must have the same name as specified in the Ont4SRE ontology.
  2. A **`query`** construct is used to perform a query, which retrieves the current state of the corresponding smart entity. Two elements are required to perform a query: the smart entity, and the name of the query. The query name must have the same name as specified in the Ont4SRE ontology. The returned result is in a **`Map`** collection form (i.e. a list of (name, value) pairs).
  3. An **`OnEvent`** construct allows the process to wait for an event from an event publisher. This kind of interaction is declared either with a **`pick`** construct or with an event handler part (**`OnEvent`**). Two elements need to be specified: the event name from where the message is expected to arrive, for example **`TemperatureEvent;`** and a variable to hold the received message. The event name must have the same name as specified in the Ont4SRE ontology.

Example 4.6

The scenario introduced in Example 3.5 of a simplified process to control an air conditioner based on the room temperature is used again. A temperature sensor is able to publish an event about temperature change. Each time the process receives such an event, it checks if the temperature is greater than **30°**; if yes, it switches the air conditioner on.

According to the Ont4SRE concepts, the room involved, the air conditioner, and associated devices are introduced as follows:

- There is an air conditioner instance of the class `Airconditioner` located in a room instantiated from the class `Room,` with a `entityID` equal, for example, to `AC-1` The air conditioner has an `AirconditionerController` actuator which offers two actions `switchOn` and `switchOff`
- The room has a `TemperatureSensor` sensor, which hosts a `TemperatureEvent Publisher` publisher to send a `TemperatureEvent,` if any change in the temperature occurs.

Listing 4.7 shows the scenario implemented with the GPL4SRE language and the subsequent points describing it:

- *Smart Entities Declarations*: line 2 specifies the involved smart entities. In this example, a single air conditioner is declared. The programmer specified the entity type, which is `Airconditioner,` according to its definition in the Ont4SRE ontology and its ID which is supposed to be `AC-1`.
- *Events Registration*: line 3 registers the process with `TemperatureChangingEvent` event.
- *The main Thread*: lines 4-11 specify the main body of the process.
- *Event Handlers*: line 6 specifies an `OnEvent` handler to wait for the temperature event.
- *Control flow and Execution Constructs*: lines 4, 5, and 7 give an example of the use of `sequence`, `while`, and `if` constructs, respectively.
- *Service Interaction Constructs*: line 8 shows how the `action` construct is used to invoke the `switchOn` service of the `Airconditioner`.
- *Map Data Structures*: in line 6, one `Map` data structure `inputVariable` is declared. It holds the received message from the temperature event. In line 7, the value of the `temperature` is accessed through a getter method.

```
Process temperatureControlScenario1{
  Airconditioner ac = discover(Airconditioner,"AC-1");
  subscribe TemperatureEvent;
  sequence {
    while (true){
      onEvent(TemperatureEvent, Map inputVariable);
        if(inputVariable.get("temperature") > 30){
          action (ac, "switchOn");
        }
    }
  }
}
```

Listing 4.7: Room temperature control scenario

In the above example, the scenario is to control the temperature of only one room if it exceeds 30°. The code to apply the same scenario to other rooms containing a temperature sensor and an air conditioner similar to those in the previous room is shown in Listing 4.8. Comparing this scenario with the previous one distinguishes the following:

- *Smart Entities Declarations*: line 2 specifies the involved smart entities. In this case, a list of air conditioners is declared. To do this with the BPEL4WS language, the programmer needs to specify each service involved manually and the bindings between services need to be known a priori. For example, for each air conditioner, the associated services **`switchOn`** and **`switchOff`** should be declared and their WSDL descriptions should be known.
- *User Preferences setting*: lines 5-12 set the user preferences. The final user can express the temperature she considers hot and selects which air conditioner can be involved in the temperature control process.
- *For Construct*: lines 16-23 give an example of the use of **`for`** construct and show how it is used to iterate over the air conditioner list in order to switch on each air conditioner in the list.

```
Process temperatureControlScenario2{
  List acList = discover(Airconditioner);
  float temperature;
  subscribe TemperatureEvent;
  preferences{
     print("Give your prefered temperature");
     set(temperature);
     for (Airconditioner ac: acList){
        print ("Would you like to use this air conditioner?");
        select (ac);
    }
  }
  sequence {
     while(true){
        onEvent ( TemperatureEvent, Map inputVariable);
           for (Airconditioner ac: acList){
              if(inputVariable.get("temperature") > temperature){
                 if(inputVariable.get("locationID") ==
                             ac.getLocation().get("locationID")){
                    action (ac, "switchOn");
                 }
              }
           }
        }
     }
}
```

Listing 4.8: Multi-rooms temperature control scenario

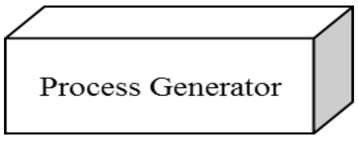
4.7.3 Key Benefits of the GPL4SRE Language

Some key features of the GPL4SRE language are its ability to:

- **Interact with smart entities:** The decision-making process nature of the SRE context often depends on acquiring the actual state of the environment and reacting to new changes by providing a set of actions to achieve the final goal such as saving energy.The GPL4SRE is able to interact in the form of action and query with declared smart entities to change or read their states. Each smart entity is related to a specific physical component that characterizes it and offers a specific number of services.
- ***Automate the discovery process***: programmers are not forced to make tedious efforts in finding and understanding details about the available smart entities and link them with their associated services and locations. Thanks to GPL4SRE and the Ont4SRE ontology, smart entities and their services are declared in implicit manner only according to their types and locations while linking, discovering, and service grounding is fully automatic, a capability that traditional composition languages like BPEL4WS lack.
- ***Facilitate the applications development and the user interaction*:** GPL4SRE expresses the smart environment's nature, and enables the programmer to easily use the environment expressions and concepts. It reduces the complexity and the size of the source code as well as increasing the level of abstraction of the composing services by hiding the low-level implementation details and facilitating coding.

- ***Iterate over a list of smart entities***: GPL4SRE offers a **`for`** loop construct to iterate over a smart entities list. This kind of iteration enables the manipulation of all the entities and services sets at once, a capability that traditional composition languages, like BPEL4WS, lack.
- ***Involve the end-user***: GPL4SRE offers a distinct execution part called **`preferences`** to enable the final user to reconfigure and customize the processes according to her needs.
- ***React to the produced events***: with event handler parts, the GPL4SRE process is able to handle events that occur independently of, and asynchronously to, the execution of the main thread at any moment during the process' lifetime.
- ***Automate the subscription to events*****:** the subscription to events is performed automatically, a capability that traditional composition languages like BPEL4WS lack, as the subscription must be programmed manually.
- ***Handle data***: it is possible to use variables to store, reformat, manipulate, and transfer messages. A programmer is allowed to declare simple, complex, global, or local variables.
- **Control flow:** the GPL4SRE enable to control the flow of the program using conditional statements like **`while`**, and **`if`**.
- ***Enable parallel and sequential executions*****:** scenario execution can be sequential or parallel.

4.8 Process Generator



The aim of the Process Generator component is to compile and convert the GPL4SRE program into a runnable process. The compiler reads the templates written in the GPL4SRE language and produces BPEL4WS executable statements. As shown in Figure 4.23, the operations performed by the compiler include:

- ***Lexical Analysis:*** the input text (process template) written in GPL4SRE is read and divided into tokens, which represents the language's symbols (e.g. *int*, *float*, *for*, *while*).
- ***Syntax Analysis***: this stage consists of building a tree-structure from the tokens list produced by lexical analysis. The tree structure corresponds to the structure of the program.
- ***Type Checking***: in this stage, the parser checks the tree structure to determine if any violation of the grammar requirements has occurred (syntax errors) or semantic errors, for example, if a variable is declared twice, or used but not declared. It determines as well whether the declared smart entities, the performed services (actions, queries, and events), and input and output parameters correspond to their definition in the Ont4SRE ontology.

- ***Code Generation***: in the final stage, the process template is translated into BPEL4WS executable code statements with their associated links, and XML files.

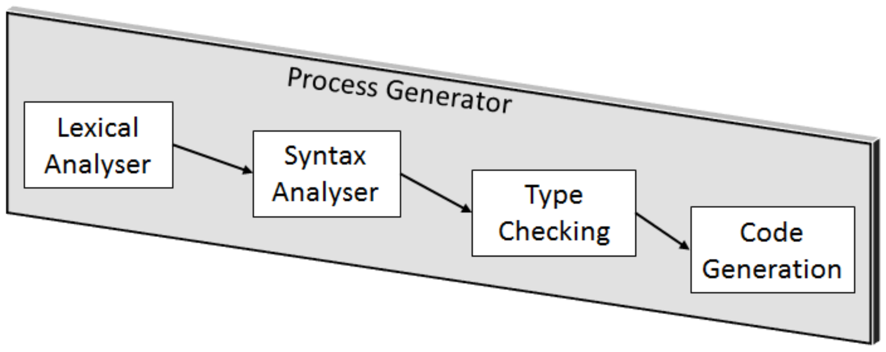


Figure 4.23: Recognizer operations

4.8.1 Benefits of Using the Process Generator

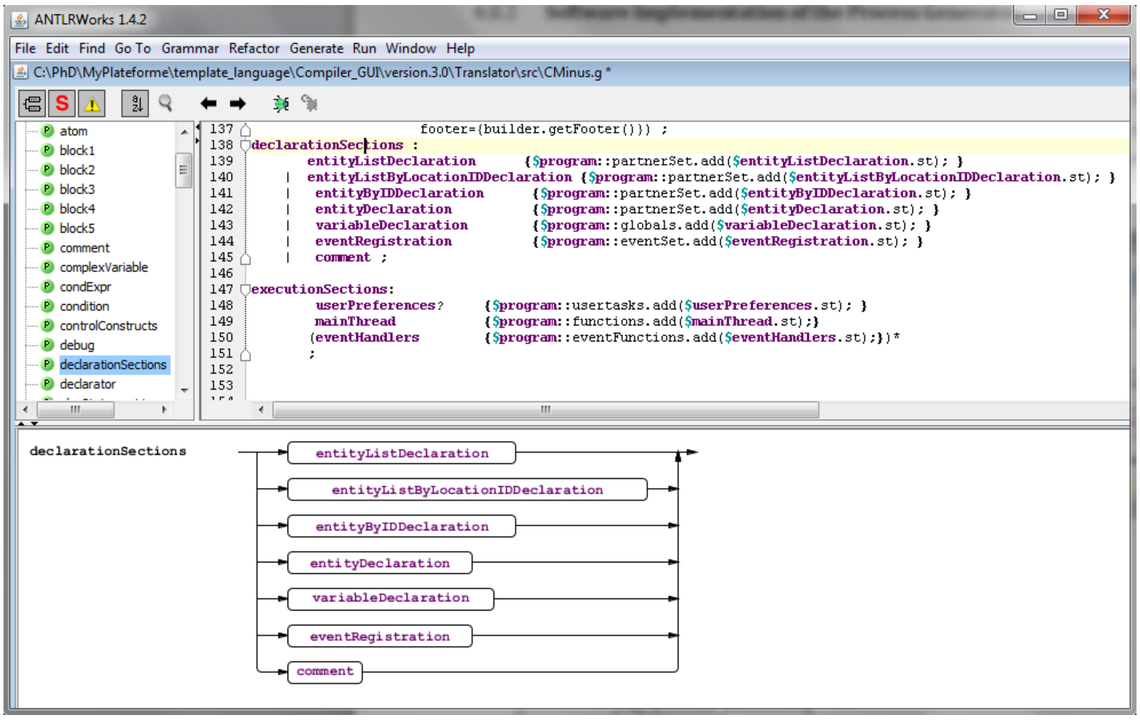
1. It generates a runnable process. This includes generating the BPEL4WS process and all its related packages.
2. It automatically checks the information about the inputs and outputs of each invoked service to assure that all the necessary data are provided, all operations can be executed, and all links are respected.
3. It detects syntax and semantic errors at the compiling stage. Listing 4.9 shows an example of error messages generated during the compilation of a GPL4SRE program. The error detection includes:
    - Checking if any violation in the language grammar requirements has occurred. This includes declaring false constructs, missing breaks or semi comas.
    - Checking if a variable or a smart entity is declared twice.
    - Checking if a variable or a smart entity is used but not declared.
    - Checking if the declared smart entities, the performed services (actions, queries, and events) and their inputs and outputs parameters correspond to their definitions in the ontology.

```
Compile-single:
Line 16:12 An error has occurred: variable x is not declared
Line 17:19 An error has occurred: entity room is not declared
Line 25:9 An error has occurred: event MotdionEvent is not found in the
          ontology
```

Listing 4.9: Error messages generated by compiling a GPL4SRE program

4.8.2 Software Implementation of the Process Generator

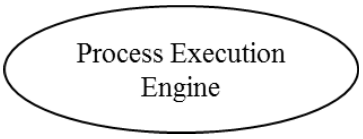
Based on the compilation phases defined above, a recognizer (recursive-descent parser) is created to read scenario templates written in GPL4SRE and to produce BPEL4WS executable statements. As shown in Figure 4.24, the recognizer is generated using the ANTLR tool (Another Tool for Language Recognition) [156]. ANTRL takes as input a grammar (see Appendix C), which defines the GPL4SRE language and generates a recognizer for it.



```
                    footer={builder.getFooter()}) ;
declarationSections :
        entityListDeclaration      {$program::partnerSet.add($entityListDeclaration.st); }
    |   entityListByLocationIDDeclaration {$program::partnerSet.add($entityListByLocationIDDeclaration.st); }
    |    entityByIDDeclaration      {$program::partnerSet.add($entityByIDDeclaration.st); }
    |    entityDeclaration          {$program::partnerSet.add($entityDeclaration.st); }
    |    variableDeclaration        {$program::globals.add($variableDeclaration.st); }
    |    eventRegistration          {$program::eventSet.add($eventRegistration.st); }
    |    comment ;

executionSections:
         userPreferences?     {$program::usertasks.add($userPreferences.st); }
         mainThread           {$program::functions.add($mainThread.st);}
        (eventHandlers        {$program::eventFunctions.add($eventHandlers.st);})*
        ;
```

Figure 4.24: Grammar development environment for building the GPL4SRE translator

4.9 Process Execution Engine



Once the executable process and its associated package are generated, the Process Execution Engine component is responsible for the life cycle of the process. The execution engine has not been developed from scratch, but is based on Oracle SOA Suite [91]. This is an open-source engine to manage and run the SOA environment. The main features of using Oracle SOA platform are:

- ***Supporting processes programmed with BPEL4WS***: It provides a comprehensive and easy-to-use infrastructure for deploying and running BPEL4WS processes. To adapt the platform to the GF4SRE framework needs, the BPEL4SE was extended, with specific parts especially for enabling the final user to express her requirements via a simple dynamic graphical interface, and perform and automate the discovery and events registry process. Providing a lightweight execution engine adapted for the SRE needs will be a good alternative in the future.

- ***Providing an Event Driven Network (EDN) for publishing and subscribing Events***: the Oracle SOA suite provides the necessary infrastructure for supporting eventing. EDN enables event subscribers and publishers to communicate via an intermediary called an event manager (broker), which performs the broadcast (notification) and the subscription tasks.

4.9.1 The Process Lifecycle

The process lifecycle is evaluated in the terms of its initialization, interaction, and execution as follows:

1. First, the process interacts with the discovery engine in order to find smart entities, which have been declared.
2. Secondly, variables are initialized.
3. If the process needs to be notified when a particular event is produced, it must register in the EDN as a new subscriber to be informed about a particular event.
4. The process starts the first execution part (*User Preferences Setting*) by setting some variables and filters the entities list (optional) according to user preferences via the *Client User Interface*.
5. Then, the process executes the main thread.
6. During the execution of the main thread, the process is able to respond to events in parallel via the declared event handlers.

4.9.2 Installing the Process Execution Engine

The process execution engine is implemented by using the Oracle SOA Suite. As shown in Figure 4.25, the following main components are installed:

1. *Database* for the SOA Suite deployments. It contains a collection of schemas used by the SOA components.
2. *Oracle WebLogic Server* is the main component of the SOA Suite to deploy and run applications and manage SOA environments. It embodies the oracle BPEL engine to execute standard BPEL processes. The WebLogic Server also provides an administration server to configure and manage all the resources such as the deployed BPEL processes and web services.

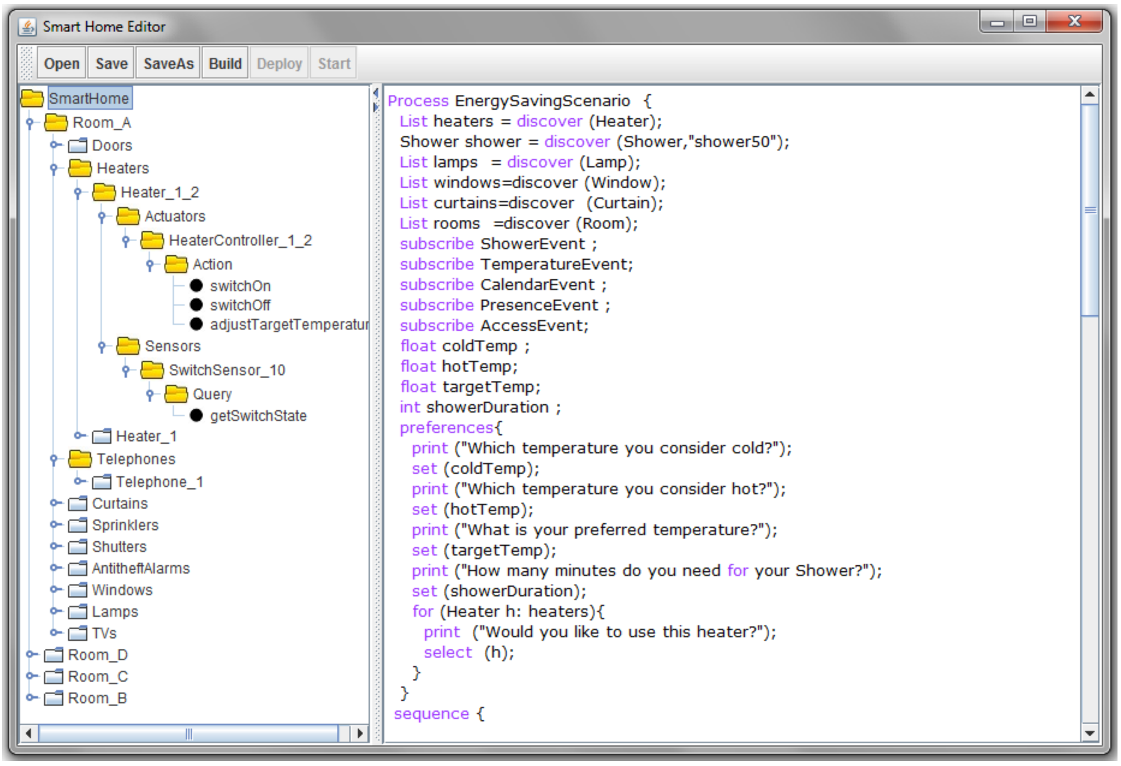
3. *Oracle Service Bus (OSB)* is a lightweight enterprise service bus (ESB). It connects and manages the interaction between heterogeneous services and legacy systems across the platform.

Database
WebLogic Server
OSB

Figure 4.25: Oracle SOA Suite main installed components

4.10 Process Template Tool

An editor is created to help the programmer to realise GPL4SRE-based control scenarios and to save, compile, deploy and run them. Figure 4.26 shows a screenshot of the realised process template tool. It is built with Java Swing components and provides a graphical user interface that consists of three parts: The first shows a graphical representation of all the smart entities presented in a SRE; the second is the text editor where the programmer writes the GPL4SRE program; the third part is the toolbar, which consists of several buttons to save, build, deploy, and run a GPL4SRE program.



```
Process EnergySavingScenario  {
  List heaters = discover (Heater);
  Shower shower = discover (Shower,"shower50");
  List lamps  = discover (Lamp);
  List windows=discover (Window);
  List curtains=discover  (Curtain);
  List rooms  =discover (Room);
  subscribe ShowerEvent ;
  subscribe TemperatureEvent;
  subscribe CalendarEvent ;
  subscribe PresenceEvent ;
  subscribe AccessEvent;
  float coldTemp ;
  float hotTemp;
  float targetTemp;
  int showerDuration ;
  preferences{
    print ("Which temperature you consider cold?");
    set (coldTemp);
    print ("Which temperature you consider hot?");
    set (hotTemp);
    print ("What is your preferred temperature?");
    set (targetTemp);
    print ("How many minutes do you need for your Shower?");
    set (showerDuration);
    for (Heater h: heaters){
      print  ("Would you like to use this heater?");
      select  (h);
    }
  }
 sequence {
```

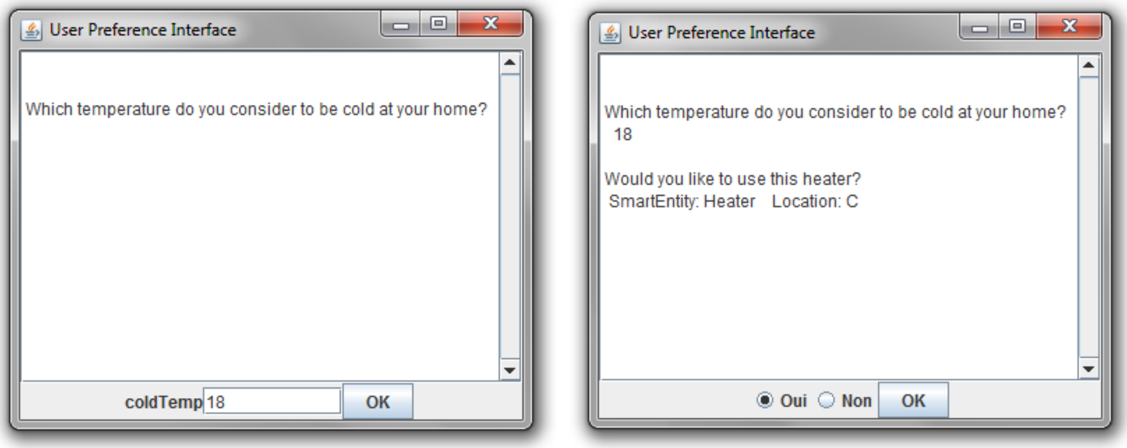
Figure 4.26: Process Template Tool

4.11 Client User Interface

Client User Interface

A simple dynamic graphic user interface has been developed to handle the communication between the end-user and the process built with Java Swing components. Figure 4.27 shows a snapshot of this interface. The bottom part allows for inserting user preference and the upper part shows messages sent by the process. According to user preferences constructs presented in Subsection 4.7.1.2, there are three possible interactions between the final user and the control process:

1. Using `print` construct to display a new message on the upper part.
2. Using `set` construct to insert data by the final user to send to the process.
3. Using `select` construct to decide if a given smart entity should be involved in a particular decision.


User Preference Interface

Which temperature do you consider to be cold at your home?

coldTemp 18 OK

User Preference Interface

Which temperature do you consider to be cold at your home?
18

Would you like to use this heater?
SmartEntity: Heater Location: C

[✓ Oui] Non OK

Figure 4.27: Client User Interface

4.12 Discussion and Summary

This chapter proposed a framework and its software architecture to facilitate modelling and controlling SRE. It explained in detail the concept, role, implementation, and benefits of each component in the GF4SRE framework, summarised as follows:

- The **Ont4SRE** ontology is proposed for modelling and describing the SRE. By using the Ont4SRE ontology, control systems and programmers can clearly answer questions about the smart entities present in a smart space.
- **Accessibility & interoperability of services in SRE (Context):** How sensors and actuators networks can be implemented using web service-oriented technology like WS-* to represent the smart entities functionalities and enable a consistent access to all the

components of an SRE was outlined. A standard description of the different services of physical components through web service uniform interfaces is provided. Further, semantic descriptions were proposed based on the Ont4SRE ontology to enable searching for smart entities. Although, there is a strong tendency in the web of Things to move towards RESTful web services, the GF4SRE does not question either the proposed language or the fact that making a model for describing the SRE - should at least question how WoT services are to be delivered and adapted to the GF4SRE framework, which could be a new challenge to explore.

- **Services registry and discovery:** A registry and discovery software component is represented to deal with the registry and discovery requirements in the SRE context. Thanks to the Ont4SRE ontology and the SPARQL language, an appropriate discovery process is proposed in order to link and interpret the purpose of both smart entities and their associated services.
- The **GPL4SRE language for controlling the SRE:** a domain specific language is proposed to create diverse control scenarios. It enables the discovery and integration of smart entities as part of the control process.

Thanks to these components, the GF4SRE framework offers the following features:

- *It is composed of pluggable software modules*. The GF4SRE is designed to achieve a loose coupling and a clean separation of the involved components and thereby enhance reusability and flexibility. The technologies are cleanly integrated for modelling and controlling different SRE (home, school, hospital, etc.) without increasing the complexity and spending tedious effort. The GF4SRE framework can be used for different buildings with the same principles and architecture.
- *It is a generic software solution* where discovering and using smart objects in control process scenarios is made in a transparent manner. System architecture minimizes the complexity through the separation of tasks using modularization and abstraction. This includes:
  - *Access transparency*: resources and services are clearly represented to users. Users declare them rather than specify where they are.
  - *Discovery transparency*: programmers do not need to determine the exact number of smart entities and services that should be involved. Smart entities and their services are declared in an implicit and generic manner by providing their type rather than the syntax of their implementation.
  - *Registry & discovery transparency*: the discovery of smart entities and event registry is completely automatic.
  - *Interaction transparency*: the information is introduced to users in an understandable fashion with fully meaning phrases.

- *Context awareness:* The proposed framework is implicitly context-aware. This includes:
  - Physical environment context: different physical properties can be considered such as location, time, temperature, rainfall, light level to handle different situations depending on the information received from sensors.
  - Human context: users are allowed to interact with the processes in terms of preferences and task requirements.
  - Virtual environment context: the GF4SRE framework enables applications to be aware of the available physical objects and their services in the SRE.
- To show the feasibility of the proposed approach, the different framework components are implemented and an interactive Smart Home prototype was developed. In Chapter 6, the Case Study of a complex control scenario using the proposed framework is introduced to illustrate its usefulness and features.

5

The GPL4SRE Domain Specific Language

In the previous chapter, the structural and conceptual models of the GPL4SRE language were introduced to set out its role played in the framework. This chapter focuses on the actual content of GPL4SRE and describes it. The basic syntax elements of a program written in GPL4SRE are introduced.

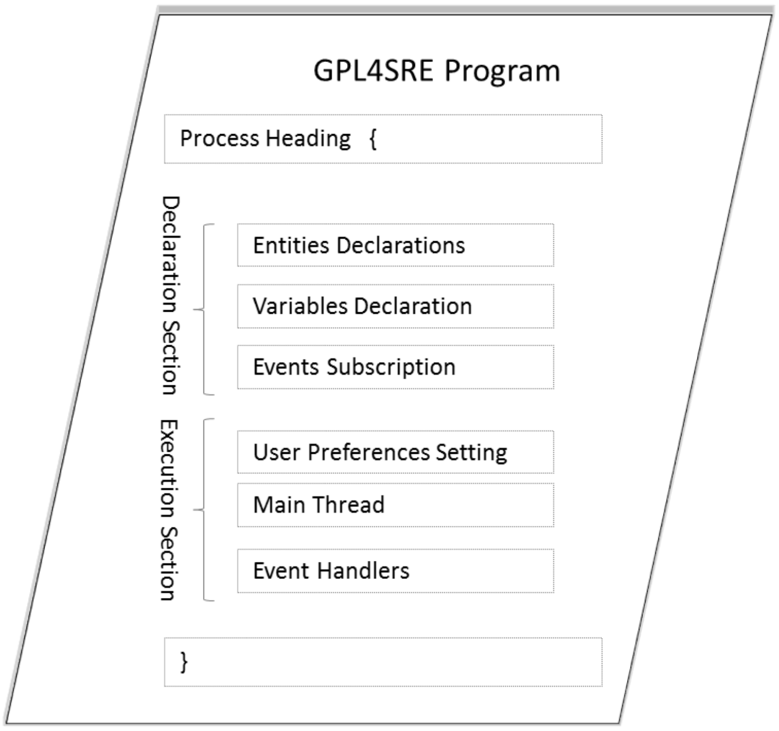


Figure 5.1: GPL4SRE Structure

5.1 GPL4SRE Program

As shown in Figure 5.1, a GPL4SRE program consists of a heading and two essential sections: *Declaration Section*, with entities and variables declarations, as well as event registration; and *Execution Section*, which is divided into three parts: user preference setting, main thread, and event handlers. The GPL4SRE program syntax is represented as follows[1].

```
program: processHeading '{' declarationSection* executionSection '}'
```

More details of the syntax of each section of the program follow below with helpful examples.

[1] In this chapter, the syntax elements are presented with the EBNF standard form in appendix C. Appendix D contains the complete syntax diagram of the GP4SRE language

5.2 The Process Heading

The process heading encloses the entire template definition and gives the program a name. The syntax of a process heading declaration is as follows.

```
processHeading: 'Process' processIdentifier
```

Example 5.1

```
Process EnergySavingScenario {
  // The process body
}
```

The **`Process`** keyword is the root element, while the word **`EnergySavingScenario`** denotes the process name (i.e. **`processIdentifier`**). It represents a sequence of characters, which normally uses the standard naming conventions.

5.3 Declaration Section

The declaration section enables the declaration of the smart entities, variables and the registration of events as follows.

```
declarationSection: entityDeclaration     | entityByIdDeclaration |
                    entityListDeclaration |
                    entityListByLocationIdDeclaration|
                    eventRegistration | varaibleDeclaration
```

5.3.1 Entities Declaration Parts

GPL4SRE offers four predefined methods to discover the smart entities. The smart entities declarations correspond to the four queries supported by the discovery engine presented in Subsection 4.6.2. Smart entities can be declared individually or as a list of entities of the same type. The syntax declaration of the fours methods is as follows.

```
entityDeclaration     : entityType entityIdentifier '=' 'discover'
                        '(' entityType ')' ';'
entityListDeclaration : 'List' entityListIdentifier '=' 'discover'
                        '(' entityType ')' ';'
entityByIdDeclaration : entityType entityIdentifier '=' 'discover'
                        '(' entityType ',' entityIdName ')' ';'
entityListByLocationIdDeclaration : 'List' entityListIdentifier '='
                        '(' entityType ',' locationIdName ')' ';'
```

Example 5.2

The declaration of a single smart entity is written as follows.

```
Room room_X = discover (Room);
```

In this example, a single entity of type **`Room`** is declared. The smart entity is randomly selected from the available entities of the same type. The elements used in calling this method, are in the following order:

- **`Room`** specifies the discovered entity type returned in calling the method **`discover`**. Other examples can be **`Door`**, **`Lamp`**, etc.
- **`room_X`** specifies the entity identifier (variable) that will hold the returned result.
- **`discover`** is the method's name to be called. To execute this method, the process interacts with the discovery engine, as explained in Section 4.6.2.
- **`Room`** is an input parameter which specifies the type of entity to be discovered. The entity's type is declared using entities name defined in the Ont4SRE ontology.

Example 5.3

The declaration of a list of smart entities is written as follows.

```
List rooms = discover (Room);
```

The difference, compared to the previous example, is that the returned result here is a list of all the available smart entities of the type **`Room`** occupying the space, rather than a single room.

Example 5.4

The declaration of a smart entity based on its identifier (**`entityIdName`**) is written as follows.

```
    Room room_X = discover (Room, "room_1");
```

In this example, a single entity of type **`Room`** is declared. The entity selection is based on the **`entityIdName`**. For this example, the result will be a single smart entity having an **`entityIdName`** equal to "**`room_1`**".

Example 5.5

```
    List heaters = discover (Heater, "room_1");
```

In this example, a list of entity of type **`Heater`** is declared. The entity filtering is based on the **`locationIdName`**. The result will be a list of heaters having a **`locationIdName`** equal to "**`room_1`**".

5.3.2 Event Registration Part

The event registration part enables the process to subscribe to a specific event. The syntax is as follows.

```
  eventRegistration : 'subscribe' eventType ';'
```

Example 5.6

```
    subscribe MotionEvent;
```

In this example, the keyword **`subscribe`** is used for the subscription and the **`MotionEvent`** denotes the type of the event that the process is interested in and should be subscribed to.

5.3.3 Variable Declaration Part

Variables are used to hold data during the run time. They must be declared before being used. Each variable declaration consists of a variable name and its data type. Data type defines which set of values a variable can hold. Variables can have a primitive (i.e. simple variable) and a complex type (i.e. complex variable).

```
  variableDeclaration : simpleVariable | complexVariable
```

5.3.3.1 Simple Variable

Simple variable declarations are made with the same syntax as for primitive variables in Java language.

```
simpleVariable : dataType variableIdentifier ('=' expression)?
```

Example 5.7

```
boolean x;
float y = 23.3;
```

GPL4SRE supports simple data types such as **`integer`**, **`float`**, and **`String`**. Once a variable is declared, any expression can be assigned to it.

5.3.3.2 Complex Variable

A complex variable denotes a **`Map`** collection (i.e. a list of (key, value) pairs). It maps keys to values and behaves like the **`Map`** collection in Java programming. It is used, for example, to pass messages between the processes and services (i.e. **`action`**, **`query`**, and **`event`**).

```
complexVariable: 'Map' variableIdentifier ('=' '{' mapPair (','
                 mapPair)* '}')? ';'
mapPair        : '(' parameterKeyName ',' (IntegerNumber |
                 DoubleNumber | STRING_LITERAL | variableIdentifier |
                 'true' | 'false') ')'
```

Example 5.8

```
Map input1={("phoneNumber", "0764596458"),("message", "Hello")};
```

In this example, **`input1`** is a complex variable, which contains two keys: "**`phoneNumber`**" with an assigned value equal to **`"0764596458"`** and a "**`message`**" with an assigned value equal to **`"Hello"`**.

5.3.3.3 Variable's Scopes

Variable declarations can appear directly within the *Declaration Section*, which means that they are visible to all GPL4SRE constructs (global variables), or can be declared under a scope, which means they are only visible to the children of that scope (local variables). The *Main Thread*, *Event Handlers* (**`onEvent`**, **`onAlarm`**), control flow constructs (**`if`**, **`else`**, **`while`**, **`for`**), and execution constructs (**`pick`**, **`flow`**, **`sequence`**) are each considered to be an enclosed scope.

5.4 Execution Section

The *Execution Section* specifies the parts that will be executed when the process is running. It includes the user preferences setting, the main thread, and event handler parts. As noted in Subsection 4.9.1, the process executes firstly the user preferences setting part then both the main thread and event handlers parts are executed in parallel.

```
executionSection : userPreferences? mainThread eventHandlers*
```

5.4.1 User Preferences Setting Part

This part customizes the process according to final user requirements. It provides some constructs to display information to the user and to assign user preferences to variables through a simple user interface. Three main constructs are used to perform these tasks: **`print`**, **`set`** and **`select`** constructs.

```
userPreferences : 'preferences' preferenceBlock
preferenceBlock : '{'(print | set | select | ifStatement4Preference |
                  forStatement4Preference)* '}'
```

The child statements of **`if`** and **`for`** constructs in the user preference setting part, are limited to **`select`**, **`print`**, **`set`**, **`if`** and **`for`** constructs.

5.4.1.1 Print Statement

`print` statement is used to display messages sent by the process to the final user through the client user interface.

```
print : 'print' '(' STRING_LITERAL ')' ';'
```

Example 5.9

```
preferences {
  print ("Hello World");
}
```

5.4.1.2 Set Statement

`set` statement is used to assign user preferences to a global variable.

```
set : 'set' '(' variableIdentifier ')' ';'
```

Example 5.10

```
float targetTemperature;
preferences {
  print ("which temperature do you like in your home");
  set(targetTemperature);
}
```

In this example, the construct **`set`** enables the user to assign a digital value to the variable `targetTemperature`.

5.4.1.3 **Select Statement**

`select` statement is used to enable the final user to select which smart entities can be involved in a given situation. Discovered entities are selected by default as long as the final user did not eliminate them.

```
select : 'select' '(' entityIdentifier ')' ';'
```

Example 5.11

```
Heater _heater = discover(Heater);
preferences {
 print ("do you want to use this heater");
 select(_heater);
}
```

5.4.2 **The Main Thread Part**

The main thread specifies the part of the process that runs when the process starts and after setting the user preferences. All constructs from Section 5.5 to Section 5.9 can be used in this part (e.g. control constructs, executions constructs, interaction constructs, other constructs, and variable assignments).

```
mainThread : 'sequence' block
block      : '{' (controlConstructs | executionConstructs |
               interactionConstructs | otherConstructs |
               assignStatement)* '}'
```

Example 5.12

```
sequence {
  int x=5;
}
```

The main thread starts with a **`sequence`** statement, which encloses all the child activities of the main thread. In the latter, it is possible to interact with different smart entities, create loops, declare variables, copy, and assign values, as well as wait for events and perform sequential or parallel executions.

5.4.3 Event Handlers Part

An event handler allows the process to react to an event or alarm that occurs in parallel while the process executes the main thread. All constructs from Section 5.5 to Section 5.9 can be used in the event handler block. The difference between the main thread and the event handlers is that the first is executed automatically and immediately when the process starts while the latters are executed each time the corresponding event is produced as long as the process is alive.

```
eventHandlers : 'onEvent' '(' eventType ',' 'Map' variableIdentifier
                ')' block |
                'onAlarm' '(' (('for' '(' duration')') |('repeatEvery'
                '(' duration ')')) ')' block
```

Example 5.13

```
onEvent (PresenceEvent, Map inputMessage){
  boolean presenceState = inputMessage.get("presenceState");
  String locationId=inputMessage.get("locationID");
}
```

This example specifies a presence event handler. The event type (**`PresenceEvent`**) corresponds to the event name defined in the Ont4SRE ontology. The child activities are executed each time a presence event publisher fires an event. A **`Map`** variable named **`inputMessage`** is declared to hold the message received from the presence event. According to the **`PresenceEvent`** definition in the Ont4SRE, it should contain two pairs one with the key "**`presenceState`**" and the other with the key "**`locationID`**" with their assigned values.

Example 5.14

```
onAlarm (for(12H-30M-20S)){
  ...
}
```

This example defines a timeout event using the **`onAlarm`** handler. The keyword **`for`** determines the duration after which the **`onAlarm`** will be executed. The time counting for the duration begins at the point in time when the process starts. For this example, the **`onAlarm`** block will be executed after 12 hours, 30 minutes and 20 seconds from when the process starts running. Another alternative is **`repeatEvery.`**It is set to fire the **`onAlarm`** repeatedly each time an interval time expires. For example, **`repeatEvery(1D5H14M20S)`** means an **`onAlarm`** handler is executed repeatedly after each 1 day 5 hours 14 minutes and 20 seconds.

5.5 Control Flow Constructs

Control flow constructs define the program's procedural logic. The important ones are **`for`**, **`while`** and **`if`**.

```
controlConstructs : ifStatement | whileStatement | forStatement
```

5.5.1 Conditional Statement (`If`)

A conditional **`if`** statement is used to control the program flow, i.e. if a specified condition is fulfilled certain statements are executed, if not, other statements are executed.

```
ifStatement   : 'if' '(' conditionExpression ')' block elseStatement?
elseStatement : 'else' block
```

Example 5.15

```
int x=4;
if (x < 5){...}
else {...}
```

5.5.2 Repetitive Execution Statement (`While`)

The **`while`** activity performs its child activities repeatedly until the specified Boolean condition no longer evaluates to be true.

```
whileStatement : 'while' '(' conditionExpression ')' block
```

Example 5.16

```
int x=1;
while (x < 10)
 {...}
```

5.5.3 Repetitive Execution Statement (`For`)

The construct **`for`** is used to iterate over a list of smart entities. The child activities of **`for`** statement are limited only to interaction constructs (**`action`** and **`query`**), assignment statements and control constructs (**`if`**, **`for`** and **`while`**). Its syntax is as follows:

```
forStatement :'for' '(' entityType entityIdentifier ':'
                 entityListIdentifier ')' partial_block

partial_block :'{' interactionConstructs | assignStatement
               | forStatement
               | ('if' '(' conditionExpression ')' partial_block)
               | ('while' '(' conditionExpression ')' partial_block)
               '}'
```

Example 5.17

```
List heaters=discover(Heater);
for (Heater h: heaters){
   if( temperature > 28){
     action (h, "switchOff");
   }
}
```

In this example, an action **`switchOff`** is sent sequentially for each heater in the heaters list if the **`temperature`** is greater than 28°.

5.6 Execution Constructs

This kind of construct is used to execute the process activities either in sequential or in parallel by using the keywords **`sequence`** and **`flow`**, respectively. Other execution constructs, inspired from BPEL, are also provided, such as the **`pick`** construct.

```
executionConstructs : sequenceStatement | flowStatement |
                      pickStatement
```

5.6.1 Sequence Execution (`sequence`)

This activity consists of one or more activities executed in sequential order. The **`sequence`** is completed by the execution of the final activity.

```
sequenceStatement : 'sequence' block
```

Example 5.18

```
sequence {
  int t=0;
  if (t <30){...}
  while (t<4){...}
}
```

In this example, the **`sequence`** activity encloses two activities **`if`** and **`while`** statements. The two activities **`if`** and **`while`** are executed sequentially.

5.6.2 Parallel Execution (`flow`)

The **`flow`** construct consists of one or more **`sequence`** statements to be performed in parallel. A **`flow`** activity is completed when all parallel activities in the flow have finished their execution. This construct is useful for example, to get results from different resources by invoking different services concurrently.

```
flowStatement : 'flow' '{' ('sequence' block)+ '}'
```

Example 5.19

```
flow{
  sequence {...}
  sequence {...}
}
```

In this example, two **`sequence`** statements are executed in parallel and simultaneously.

5.6.3 Pick Statement

The **`pick`** construct can specify a set of choices of possible blocks to be executed. Each block starts with an event handler **`OnEvent`** or **`onAlarm`** handler. Only the event handler, that is produced first, will run its procedure, and the **`pick`** block will complete once that event handler's activities have been executed.

```
pickStatement : 'pick' '{' (onEventStatement | onAlarmStatement)+ '}'

onEventStatement : 'onEvent' '(' eventType ',' 'Map'
                   variableIdentifier ')' block

onAlarmStatement : 'onAlaram' '(' 'for' '(' duration ')' ')' block
```

Example 5.20

```
pick {
  onEvent(MotionEvent, Map input1){
    ...
  }
  onAlarm (for(11H-30M-20S)){
    ...
  }
}
```

In this example, two event handlers are declared with the **`pick`** statement a **`MotionEvent`** and an **`onAlarm`**. When the program execution reaches the **`pick`** statement, it will wait either for the **`MotionEvent`** to be produced or for the **`onAlarm`** duration time to expire. The first of two handlers fired will execute its child activities. Time counting for the duration of the **`onAlarm`** begins at the point in time when the program execution reaches the **`pick`** statement. The time expression defined for the **`onAlarm's`** example specifies a duration of 11 hours, 30 minutes and 20 seconds. This means an alarm will be produced after 11 hours, 30 minutes and 20 seconds from when counting started.

5.7 Services Interaction Constructs

Service interaction constructs allow interaction with the smart entities via their devices (sensors and actuators). There are three kinds of interaction: **`action`**, **`query`** and **`onEvent`**.

```
interactionConstructs : action | query | onEvent
```

5.7.1 Action Construct

The **`action`** construct is used to modify the state of a given smart entity.

```
action : 'action' '(' 'this.'? entityIdentifier ',' serviceName ((','
         'this.'?) variableIdentifier)? ')' ';'
```

Example 5.21

```
Heater heater = discover(Heater);
Map inputVariable = {("temperature",25.0)};
action(heater, "adjustTargetTemperature", inputVariable);
```

With the **`action`** statement, three parameters are needed: the entity and action names, and a **`Map`** variable, which holds the name of the entity state to be modified and its new value. In this example, the **`heater`** is the smart entity, the "**`adjustTargetTemperature`**" is the action name, and the **`inputVariable`** is used to hold the entity state "**`temperature`**" with an assigned value equal to **`25.0`**. The keyword **`this`** is used to distinguish between the smart entity declared globally in the *Declaration Section* (e.g. **`this.heater`**) and the one declared locally inside the scope.

5.7.2 Query Construct

The **`query`** construct is used to read the current state of a given smart entity.

```
query : 'Map' variableIdentifier '=' 'query' '(' ('this.')?
        entityIdentifier ',' serviceName (',' ('this.')?
        variableIdentifier)? ')' ';'
```

Example 5.22

```
Room room = discover(Room);
Map result = query (room, "getTemperature");
```

The **`query`** statement takes the smart entity and the name of the query as input parameters and returns the requested current state. In this example, **`room`** is the entity identifier, "**`getTemperature`**" is the name of the query, and **`result`** is the variable used to hold the returned **`temperature`**. Then, **`result.get("temperature")`** is used to get the temperature.

5.7.3 onEvent Construct

The **`onEvent`** construct can be declared inside the main thread or any event handler without needing a **`pick`** statement to wait for a specific event.

```
onEvent :'onEvent' '(' eventType ',' 'Map' variableIdentifier ')' ';'
```

Example 5.23

```
X=5;
onEvent (TemperatureEvent, Map input1);
if(input1.get("temperature") < 18){
  ...
}
```

In this example, the **`onEvent`** statement is declared to wait for a **`TemperatureEvent`**. The part of the program declared after the **`onEvent`** statement will not be executed until the event is fired. A map data structure named **`input1`** is declared to hold the message received from the event. In this example, the program first assigns the number 5 to the variable **`x`**, then it waits for the temperature event to be produced. Once the event is fired, the next activity (**`if statement`**) is executed.

5.8 Other Useful Constructs

Other useful constructs are available with GPL4SRE language like **`wait`**, **`terminate`**, and **`empty`**.

```
otherConstructs : waitStatement | terminateStatement | emptyStatement
```

5.8.1 Wait Statement

The `wait` activity enables the process to wait for a determined time interval. The syntax is as follows.

```
waitStatement : 'wait' '(' expression ')' ';'
```

Example 5.24

```
wait(60*3); // wait for 3 minutes
```

5.8.2 Terminate Statement

The **`terminate`** constructs allows the stop process instance to be executed immediately. The syntax is as follows.

```
terminateStatement : 'terminate' '(' ')' ';'
```

Example 5.25

```
terminate();
```

5.8.3 Empty Statement

This activity has no operation associated with it. It does nothing. The syntax is as follows.

```
emptyStatement :  'empty' '(' ')' ';'
```

Example 5.26

```
empty();
```

5.9 Assignment Statement

The assignment statement is used to manipulate and assign data from one variable to another, as well as to construct and insert new data. Table 5.1 gives some examples of possible assignments including those for simple and complex variables.

```
assignStatement : assignSimpleVariable | assignComplexVariable |
                  setMapVarParameter | getEntityById |
                  getEntityByLocation;
```

Table 5.1: Example of variables Assignments

Description	**Assignment statement**	**Description**
Assign a constant to a simple variable.	`x = 100;`	`x` and `y` are simple variables. 100 is a constant.
Assign a simple variable value to another simple variable.	`y = x;`	
Assign a key's value from a complex variable to a simple variable.	`x = b.get("location");`	`b` and `d` are complex variables. `location` is a key. `RoomA` is a string.
Assign a constant to a complex variable key.	`d.set("location","RoomA");`	
Assign a simple variable value to a complex variable key.	`d.set("location",c);`	`location` and `x` are keys. `d` is a complex variable. `C` is a simple variable.
Assign a key's value from a complex variable to a complex variable key.	`d.set ("x",b.get("d"));`	
Get an entity by its Id name.	`Entity h=getEntityById (heaters,"h1");`	`h` is an entity. `heaters` is an entity list. `h1` is the entity identifier name.
Get an entity by its location.	`Entity h=getEntityById (heaters,"RoomA");`	

6 Case Study: Smart Home Scenarios

This chapter introduces a case study based on the proposed framework for the smart home environment. The case study includes the development of two scenarios revolving around energy saving, security and safety.

6.1 Smart Home Environment

The smart home introduced in this chapter is similar to Jane's home introduced in Section 3.7.1. Figure 6.1 describes this particular home, which has four rooms: A, B, C, D, and a garden, E. These places are populated by the set of smart entities presented in Table 6.1.

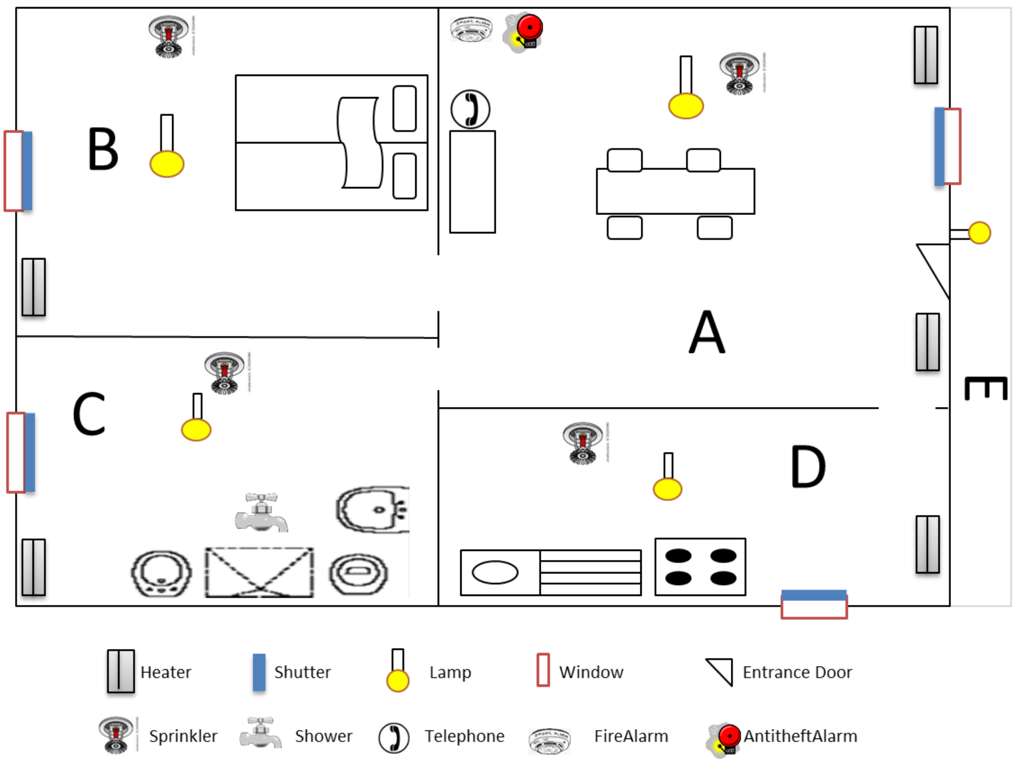


Figure 6.1: Smart home plan

Table 6.1: Smart Entities occupying the smart home

SmartEntity	Smart Location					Total
	Room A	Room B	Room C	Room D	Garden E	
Heater	2	1	1	1		5
Window	1	1	1	1		4
Lamp	1	1	1	1	1	5
Shutter	1	1	1	1		4
Door	1					1
Sprinkler	1	1	1	1		4
Telephone	1					1
AntitheftAlarm	1					1
FireAlarm	1					1
Shower			1			1

Each entity presented in Table 6.1 is associated with actuator(s) and/or sensor(s) which offer the services presented in Table 6.2. The actuators and services' names correspond to their definition in the Ont4SRE ontology.

Table 6.2: Smart entities, their devices, and services according to the Ont4SRE definitions

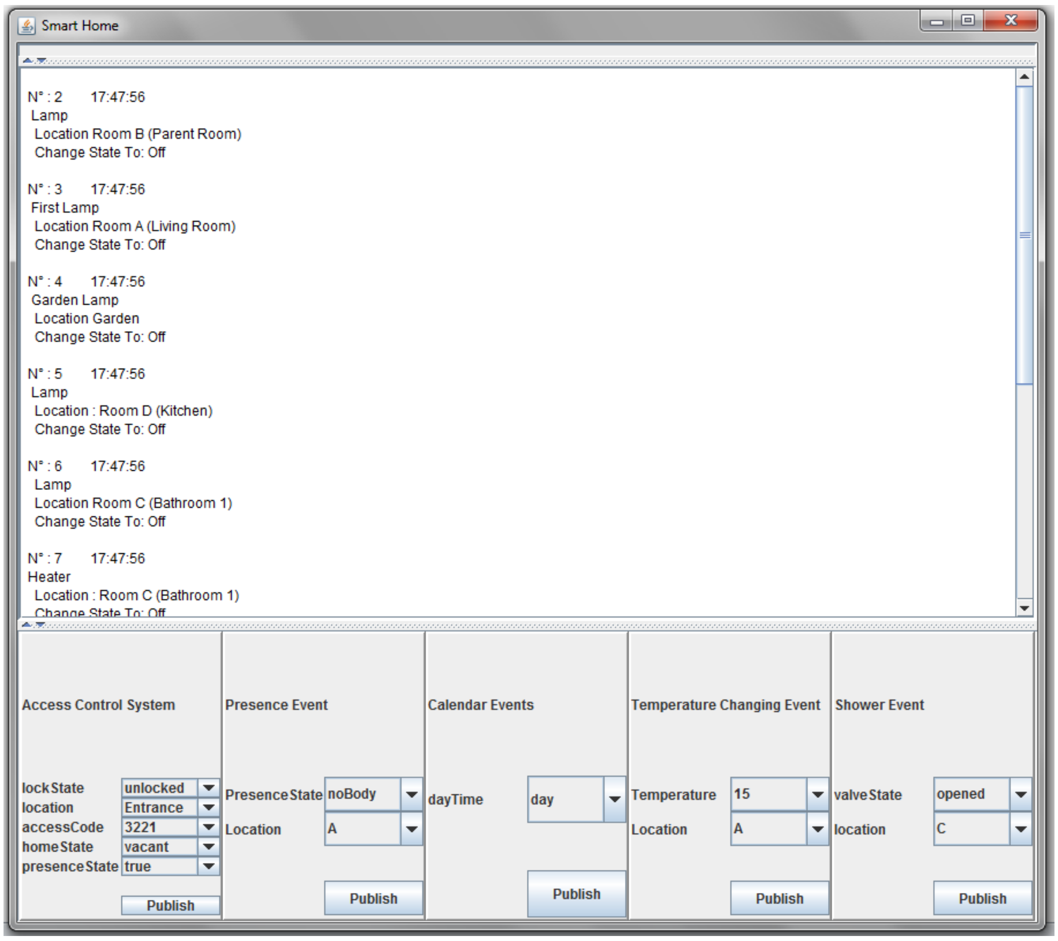
Entity	**Sensors & Actuators**	**Concerned Property**	**Services(actions, queries, events)**
Room	FireSensor	dangerLevel	FireEvent
	PresenceSensor	presenceState	PresenceEvent getPresenceState
	TemperatureSensor	temperature	TemperatureEvent getTemperature
	MotionSensor	motionState	MotionEvent
	WaterLeakSensor	leakLevel	WaterLeakEvent
Door	AccessControlSystem	lockState, homeState, doorState, accessCode	open, close, lock, unlock, getHomeState, getLockState, getAccessCode
	DoorBrokenSensor	dangerLevel	DoorBrokenEvent
Window	WindowController	lockState, windowState	lock, unlock, open, close
	WindowBrokenSensor	dangerLevel	WindowBrokenEvent
Shutter	ShutterController	lockState, positionState	open, close
Lamp	LampController	switchState, brightnessLevel	switchOn, switchOff, getSwitchState, adjustLight
Heater	HeaterController	switchState, targetTemperature	switchOn, switchOff, getSwitchState, adjustTargetTemperature
Shower	ShowerController	faucetState	open, close
Telephone	TelephoneStation	Message, phoneNumber	eCall, sendSMS
AntitheftAlarm	AlarmController	switchState	switchOn, switchOff
FireAlarm	FireAlarmController	switchState	switchOn, switchOff
Sprinkler	SprinklerController	valveState	open, close
Calendar	DigitalCalendar	daytime, nighttime, date	CalendarEvent, getDaytime, getNighttime
WeatherStation	DigitalWeather Station	dangerLevel	WindstormEvent

6.2 Making-Ready the Smart Home

Three steps are necessary to make the smart home ready for the execution of different home control scenarios: (i) making available the smart entities and their services; (ii) instantiating the home ontology; (iii) making ready the discovery and registry engine.

6.2.1 Making-Ready the Smart Entities

This step consists of building the smart home and making available each smart entity and its related services through the network. As indicated in Section 4.8, for the proposed smart home, the services provided by actuators, and sensors are simulated and described in web service interfaces. Each device (actuator or sensor) has one interface that provides its services (queries, events, and actions) corresponding to their definition in Table 6.2. The web services associated with a smart home web application are deployed on a web server. The evaluation of service invocations and the produced events are visualized in a graphical user interface, as in Figure 6.2 below. The lower section allows for artificially publishing various events such as **`MotionEvent`** or **`ShowerEvent`** while the upper section shows the reaction of the system (Smart Home) according to the action scenarios.



```
N° : 2      17:47:56
 Lamp
  Location Room B (Parent Room)
  Change State To: Off

N° : 3      17:47:56
 First Lamp
  Location Room A (Living Room)
  Change State To: Off

N° : 4      17:47:56
 Garden Lamp
  Location Garden
  Change State To: Off

N° : 5      17:47:56
 Lamp
  Location : Room D (Kitchen)
  Change State To: Off

N° : 6      17:47:56
  Lamp
  Location Room C (Bathroom 1)
  Change State To: Off

N° : 7      17:47:56
Heater
  Location : Room C (Bathroom 1)
  Change State To: Off
```

Access Control System		Presence Event		Calendar Events		Temperature Changing Event		Shower Event	
lockState	unlocked	PresenceState	noBody	dayTime	day	Temperature	15	valveState	opened
location	Entrance	Location	A			Location	A	location	C
accessCode	3221								
homeState	vacant								
presenceState	true								

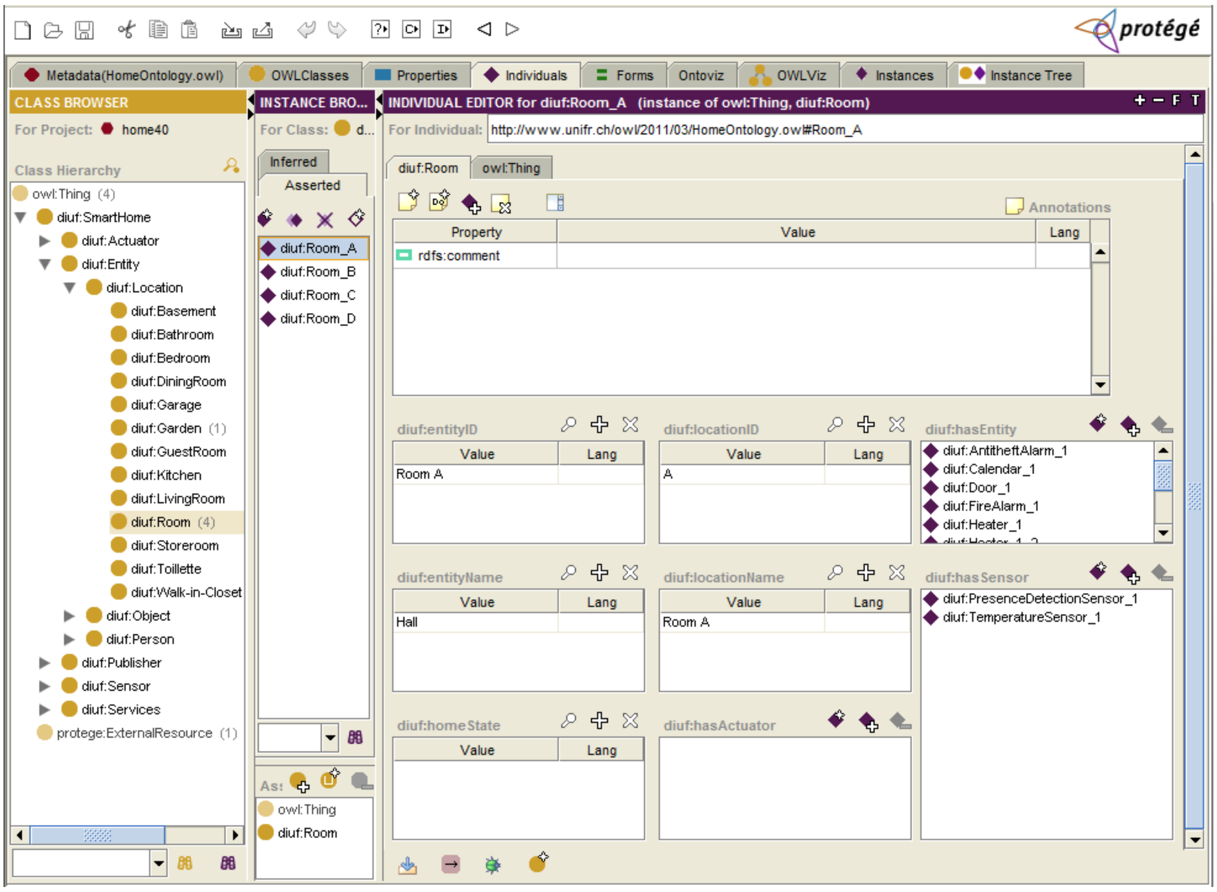
Figure 6.2: Smart home simulator

6.2.2 Instantiating the Smart Home Ontology

Conforming to the ontology instantiation steps presented in Subsection 4.6.1.1, the present smart home ontology was completely instantiated, which included:

- The instantiation of the five locations (i.e. room A, B, C, D and garden E).
- The instantiation of each smart entity located in each room, given in Table 6.1.
- The instantiation of each actuator and sensor associated with each smart entity, given in Table 6.1.
- The instantiation of each service (action, query, event) provided by each actuator and sensor presented in Table 6.2.

The ontology instantiation is relatively simple using a protégé tool. Figure 6.3 shows an example of how a room is instantiated using the protégé tool.



INDIVIDUAL EDITOR for diuf:Room_A (instance of owl:Thing, diuf:Room)

For Project: home40

For Class: d...

For Individual: http://www.unifr.ch/owl/2011/03/HomeOntology.owl#Room_A

Class Hierarchy
- owl:Thing (4)
  - diuf:SmartHome
    - diuf:Actuator
    - diuf:Entity
      - diuf:Location
        - diuf:Basement
        - diuf:Bathroom
        - diuf:Bedroom
        - diuf:DiningRoom
        - diuf:Garage
        - diuf:Garden (1)
        - diuf:GuestRoom
        - diuf:Kitchen
        - diuf:LivingRoom
        - diuf:Room (4)
        - diuf:Storeroom
        - diuf:Toillette
        - diuf:Walk-in-Closet
      - diuf:Object
      - diuf:Person
    - diuf:Publisher
    - diuf:Sensor
    - diuf:Services
  - protege:ExternalResource (1)

Instances: diuf:Room_A, diuf:Room_B, diuf:Room_C, diuf:Room_D

Asserted types: owl:Thing, diuf:Room

Tabs: diuf:Room, owl:Thing

Property	Value	Lang
rdfs:comment		

diuf:entityID

Value	Lang
Room A	

diuf:locationID

Value	Lang
A	

diuf:hasEntity: diuf:AntitheftAlarm_1, diuf:Calendar_1, diuf:Door_1, diuf:FireAlarm_1, diuf:Heater_1, diuf:Heater_1_2

diuf:entityName

Value	Lang
Hall	

diuf:locationName

Value	Lang
Room A	

diuf:hasSensor: diuf:PresenceDetectionSensor_1, diuf:TemperatureSensor_1

diuf:homeState

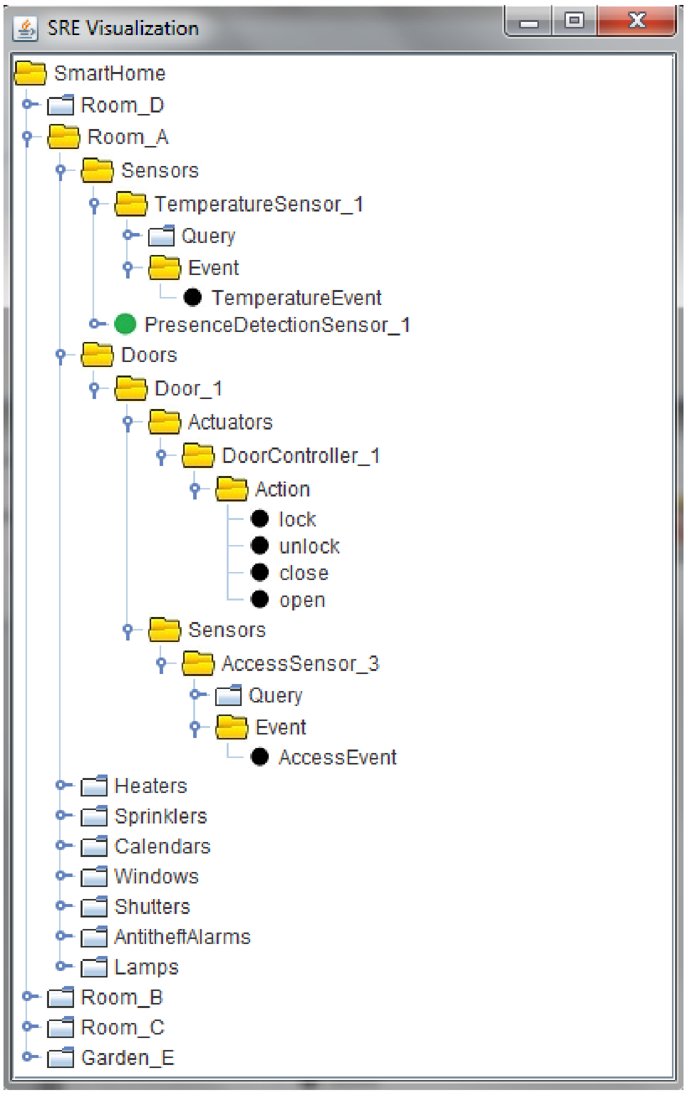
Value	Lang

diuf:hasActuator

Figure 6.3: Smart home instantiation using a protégé tool

6.2.3 Making-Ready the Registry and Discovery Engine

After the ontology instantiation step, the ontology was saved in the registry and discovery engine as a database store for the smart home for the registry and discovery proposals. The registry process performed conformed to the steps introduced in Subsection 4.6.1.2. When actuators and sensors are installed (i.e. deployed within the smart home simulator), they send a registry request similar to Listing 4.4, to the register engine to provide the necessary grounding information and to enable the smart entities to become part of the services providers in the instantiated ontology. By finishing the registry process, the discovery engine can respond to queries concerning the discovery of the smart entities and services. Figure 6.4 shows a screenshot of the registered smart entities and their associated devices and services.



```
SmartHome
  Room_D
  Room_A
    Sensors
      TemperatureSensor_1
        Query
        Event
          TemperatureEvent
      PresenceDetectionSensor_1
    Doors
      Door_1
        Actuators
          DoorController_1
            Action
              lock
              unlock
              close
              open
        Sensors
          AccessSensor_3
            Query
            Event
              AccessEvent
    Heaters
    Sprinklers
    Calendars
    Windows
    Shutters
    AntitheftAlarms
    Lamps
  Room_B
  Room_C
  Garden_E
```

Figure 6.4: Visualization of the smart entities occupying a SRE

6.3 Example Scenarios

Once the ontology has been instantiated and all the services registered, diverse scenarios can be simply written with the GPL4SRE language. In the remainder of this section, two scenarios are demonstrated: energy saving, and security and safety.

6.3.1 Energy Saving Scenario

The energy consumption of homes depends mainly on the usage of the HVAC, lighting, and electrical devices like TV, washing machine, etc. If the final user wants to reduce energy consumption during the winter season, it can be reduced by the following strategies:

▪ **Case I: When the house's occupant leaves or enters the home**

  The home can be set in vacant mode when the inhabitant leaves the home. In this situation and to save the energy, the following actions are taken:

  - Switch off all the heaters and lamps.
  - Close all the windows.

  When the inhabitant enters the home, the following actions are taken:

  - Switch on the heater in each room which has a temperature lower than the minimum.

▪ **Case II: when the house's occupants are at home**

  When a user is at home, the HVAC and lighting systems should be turned off or their consumption can be minimized when it is not needed. Concerning occupied rooms', managing the temperature and light is based on presence detection, the comfort level needed and the time of sunrise or sunset and the room's function. The following actions can be taken:

  - Switch off the lights in rooms when they are not occupied or during daylight hours.
  - Open all the shutters to allow the light to enter the rooms.
  - Switch on the light when somebody enters a given room during the night hours.
  - Switch on the heater if the temperature is lower than the minimum level and off if the temperature is greater than the maximum level required.
  - Close windows if heaters are switched on.
  - Adjust the heater to an economical level during the hours of sleep at night.
  - Limit hot water consumption during showers.

6.3.1.1 Programming methodology

As in any other language, there are many ways to implement the scenario using GPL4SRE. By analyzing the scenario and from the energy saving strategies explained above, the following points arise:

- The necessary queries and actions depend on the environment context changing. For example, switching a lamp on or off depends on the presence detection (`PresenceEvent`) while switching a heater on or off depends on temperature change (`TemperatureEvent`). Therefore, the decisions should be grouped according to each type of event and its effects.
- Additional contextual information sometimes needs to be gathered to determine the right decision (e.g. if a change in the temperature occurs in a given space, we need to know if that space is occupied by somebody, and if it is day or night time). Therefore, to be sure of having the correct information about the current state of the environment, queries can be made about the states in question.

These two criteria serve as guidelines for programming the two introduced scenarios in a clear way. A redraft of the energy saving scenario description based on these criteria is given in Listing 6.1, providing a high-level description (pseudo-code) of the necessary queries and actions, and the involved smart entities grouped under the following five event handlers.

1. **AccessEvent:** an access event (lines 1-10) is produced when the inhabitant enters or leaves the home. It informs of any change in the `homeState` (i.e. `occupied` value if the inhabitant enters and `vacant` value if he leaves).
2. **PresenceEvent**: a presence event (lines 11-20) is produced when the inhabitant enters or leaves a given location. It informs of any change in the `presenceState` (i.e. `true` value if the inhabitant enters and `false` value if he leaves), and the location where this change has occurred.
3. **TemperatureEvent:** a temperature event (lines 21-29) is produced when the temperature changes. It informs about the new `temperature`, and the location where this change has occurred.
4. **ShowerEvent:** a shower event (lines 30-33) is produced when the inhabitant starts taking her shower. It informs about the change in the `faucetState` (i.e. `opened` value if the faucet is opened and a `closed` value if it is closed), and the location where this change has occurred.
5. **CalendarEvent:** a calendar event (lines 34-45) is produced when day or night begins. It informs of the change from e.g. `daytime` to `nighttime` (i.e. `daytime` has `true` value if it is day, while `nighttime` has `true` value if it is night).
6. **OnAlarm Event:** an `onAlarm` event (lines 46-48) is produced after a timeout expires. For this scenario, an alarm is produced each time the clock reaches midnight.

```
When an AccessEvent is produced
  if the homeState equals vacant
    switch off Heaters and Lamps
  if the homeState equals occupied
    for each room
      query the current temperature in the Room
      if the current temperature is equal to or less than the minumim
        switch Heater on
        adjust Heaters' targetTemperature to comfort level
        close Window in the room concerned

When a PresenceEvent is produced
  if the presenceState is false
    wait for 3 minutes
    query the presenceState in the Room in which the event was produced
    if the presenceState is still false
      switch off the Lamps in the given room
  else
    query the Calendar whether it is daytime or nighttime
    if it is nighttime
      switch on the Lamp in the given room

When a TemperatureEvent is produced
  query the homeState from the EntranceDoor
  if the homeState is occupied
    if the change in temperature is less than the minimum required
      switch on the Heater in the room concerned
      adjust the Heater's targetTemperature to comfort level
      close the Window in the room concerned
    If the temperature is greater than the maximum needed
      switch off the Heater in the given location

When a ShowerEvent is produced
 if the faucetState is opened
   wait for x minutes (user determines the shower duration)
   close the Shower

When a CalendarEvent is produced
  if it is daytime
    open the Shutters
    switch the Lamps off
    for each Heater
      adjust the Heater' targetTemperature to confort level
  if it is nighttime
    close the Shutters
    for each Room
      query the presenceState in the Room
      if the presenceState equals true
        switch on the Lamp in the room concerned

When onAlarm on midnight is produced
  for each Heater
    adjust the Heater's targetTemperature to an economical level
```

Listing 6.1: Pseudo-code concerning the energy saving scenario

6.3.1.2 Energy Saving Scenario Implementation

Listing 6.2 shows how the energy saving scenario is written in a GPL4SRE. Hereafter, the code lines are commented on from a programming methodology viewpoint since the language syntax and constructs were introduced in Section 4.7 and Chapter 5. Decisions and actions are grouped under the five event handlers, as explained above, and according to the following order sequence:

- The main thread (lines 34-69) consists of an `AccessEvent`. In general, where activities are less often performed or where the aim is to initialize a process, these are declared within the main thread. When the process starts, it queries the `homeState`. Depending on the query's response, if the home is in vacant mode, it switches all the heaters and lamps off. Otherwise (`occupied` mode) other queries concerning the temperature of each room are performed and depending on the result of those queries, some or all heaters could be switched on. This is the home initialization then the process waits for an `AccessEvent` (line 66) to be produced to repeat the previous steps.
- The other five event handlers (`PresenceEvent, TemperatureEvent, ShowerEvent` and `CalendarEvent,` and `OnAlarm` event) are declared as separate event handlers in parallel with the main thread. Some contain a `wait` statement. If they are declared inside a `pick` statement with other events handlers, this will prevent the process from executing the next activity until it has finished executing the `wait` activity, which could take a period of time (i.e. a few minutes, hours or days).
- Lines 94-121 specify a `TemperatureEvent` handler which is programmed as follows.
  - Line 95 queries the `homeState` from the entrance door.
  - Lines 96-97 assign the `homeState` received from the query and the `locationID` received from the event message into two variables named `homeState` and `location`
  - Lines 98-121 perform a set of conditions and actions including:
    - Line 98 checks if the home is occupied. If that is true, line 99 is executed to check if the temperature is considered cold. If that is true as well, lines 100-113 are executed. The heater located where the temperature changed to low, is switched on and the window located in the same place is closed.
    - If the condition in line 99 is evaluated to be false, then the temperature is not considered cold and another `if` statement in line 114 is performed to check if the temperature exceeds the maximum needed temperature. If line 114 is evaluated to be true, then lines 115-121 are executed. The heater located where the event was produced, is switched off.

```
Process EnergySavingScenario {

  List heaters  = discover(Heater);
  Shower shower = discover(Shower,"shower_C");
  List lamps    = discover(Lamp);
  List windows  = discover(Window);
  List shutters = discover (Shutter);
  List rooms    = discover(Room);
  Door entranceDoor = discover(Door,"EntranceDoor_1");
  Calendar calendar =discover (Calendar);

  subscribe ShowerEvent;
  subscribe TemperatureEvent;
  subscribe CalendarEvent;
  subscribe PresenceEvent;
  subscribe AccessEvent;

  float coldTemp;
  float hotTemp;
  float targetTemp;
  float econmicTemp = 18;
  int showerDuration;

  preferences {
    print("Which temperature do you consider to be cold?");
    set(coldTemp);
    print("Which temperature do you consider to be hot?");
    set(hotTemp);
    print("What is your preferred temperature?");
    set(targetTemp);
    print("How many minutes do you need for your Shower?");
    set(showerDuration);
    for(Heater h: heaters){
      print ("Would you like to use this heater?");
      select (h);
    }
  }

  sequence{
    Map result = query(entranceDoor,"getHomeState");
    String homeState = result.get("homeState");
    while(true){
      if(homeState == "vacant"){
        for (Lamp lamp : lamps){
          action (lamp, "switchOff");
        }
        for (Heater heater : heaters){
          action(heater,"switchOff");
        }
      }
      if(homeState == "occupied"){
        for(Room room: rooms){
          Map temp = query(room, "getTemperature");
          String location = room.getLocation().get("locationID");
          if(temp.get("temperature") < coldTemp){
            for(Heater h:heaters){
              if(h.getLocation().get("locationID") == location){
                action(h,"switchOn");
                Map input1 = {("temperature",targetTemp)};
                action(h,"adjustTargetTemperature",input1);
              }
            }
            for(Window w: windows){
              if(w.getLocation().get("locationID") == location){
                action(w,"close");
              }
            }
          }
        }
      }
      onEvent(AccessEvent, Map input2);
      homeState=input2.get("homeState");
    }
  }

  onEvent(PresenceEvent, Map input5){
    String location = input5.get("locationID");
    if (input5.get("presenceState") == false){
      wait(3*60);
      Room room = getEntityByLocation(rooms,location);
      Map input6 = query(room,"getPresenceState");
      if(input6.get("presenceState") == false ){
        for(Lamp l: lamps){
          if(l.getLocation().get("locationID") == location){
            action(l,"switchOff");
          }
        }
      }
    }else{
      Map input7 = query(calendar, "getDaytime");
      if(input7.get("dayTime") ==  false){
        for(Lamp l: lamps){
          if(l.getLocation().get("locationID") == location){
            action(l,"switchOn");
          }
        }
      }
    }
  }

  onEvent(TemperatureEvent, Map input1){
   Map result1 = query(entranceDoor,"getHomeState");
   String homeState = result1.get("homeState");
   String location = input1.get("locationID");
   if(homeState == "occupied"){
     if(input1.get("temperature") <coldTemp) {
       for (Heater h: heaters){
         if(h.getLocation().get("locationID") == location){
            action(h,"switchOn");
            Map input2 = {("temperature",targetTemp)};
            action(h,"adjustTargetTemperature",input2);
         }
       }
       for (Window w: windows){
         if(w.getLocation().get("locationID") == location){
           action(w,"close");
         }
       }
     }
   }
   if(input1.get("temperature") > hotTemp){
       for (Heater h: heaters){
         if(h.getLocation().get("locationID") == location){
           action(h,"switchOff");
         }
       }
    }
  }

  onEvent(ShowerEvent, Map input3){
   if(input3.get("faucetState") == "opened"){
     wait(showerDuration * 60);
     action (shower, "close");
   }
  }
```

```
128    onEvent(CalendarEvent, Map input4){
129      if(input4.get("dayTime") == true){
130        for(Shutter s: shutters){
131          action(s,"open");
132        }
133        for(Lamp l: lamps){
134          action(l,"switchOff");
135        }
136      }
137      else{
138        for(Shutter s: shutters){
139          action(s,"close");
140        }
141        for(Room room: rooms){
142          Map p = query(room, "getPresenceState");
143          String location = room.getLocation().get("locationID");
144          if(p.get("presenceState") == true){
145            for(Lamp l: lamps){
146              if(l.getLocation().get("locationID") == location){
147                action(l,"switchOn");
148              }
149            }
150          }
151        }
152      }
153    }

154    onAlarm(repeatEvery(23:59:59,1D-1S)){
155      for(Heater h:heaters){
156        Map input4 = {("temperature",econmicTemp)};
157        action(h,"adjustTargetTemperature",input4);
158      }
159    }
160  }
```

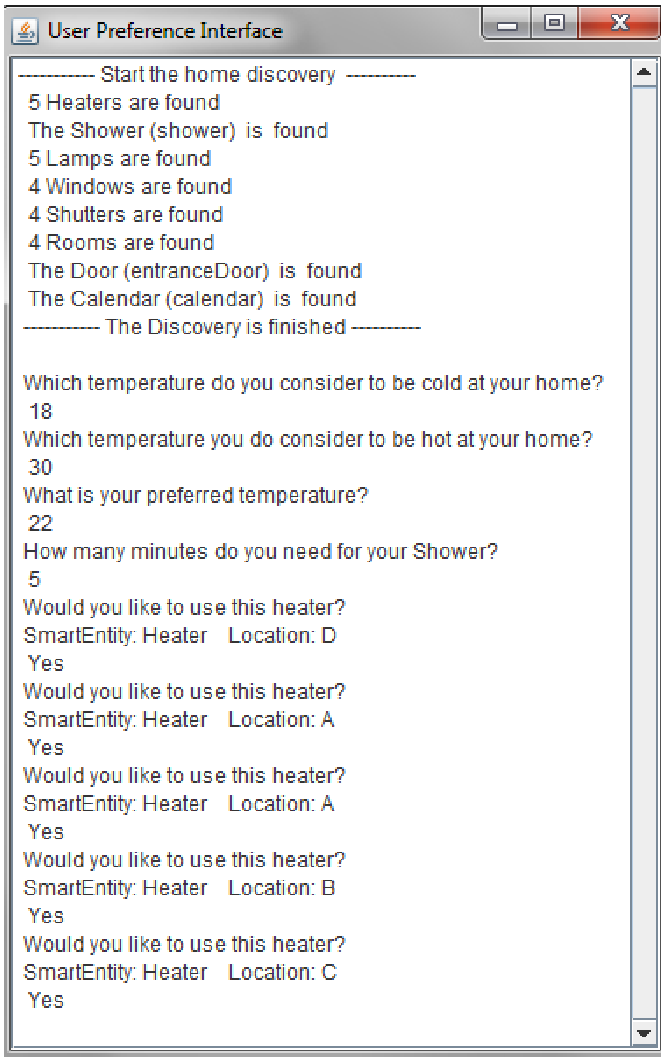
Listing 6.2: The energy saving program written in GPL4SRE

6.3.1.3 Scenario Execution Monitoring

Figure 6.5 and Table 6.3 monitor the energy saving scenario's execution history. According to the lifecycle process introduced in Subsection 4.9.1, the energy saving program is executed in the following order:

1. First, the process interacts with the discovery engine in order to find the smart entities declared in lines 2-9. As shown in Figure 6.5, 25 smart entities matching the requested criteria can be found.
2. Then, the process registers in the EDN as a new subscriber to be informed about all the events declared in lines 10-14.
3. Then, the user preferences are set (lines 20-33), by asking the user which temperature she considers cold and hot in the rooms as well as asking about the duration of the shower and selecting which heater will be involved, as illustrated in Figure 6.5.
4. The process then executes the main thread (lines 34-69) during which the process is able to respond to the five events in parallel through the event handlers' blocks (lines 70-159).

5. Table 6.3 shows an extract of the execution history of the saving energy scenario (i.e. main thread and events' handler parts). As can be noted, the running scenario reacts to the different events. Each time, an event is produced, the appropriate actions and queries are performed according to their order in the program. For example, on 13:30:33, an access event was produced informing the process that somebody has entered the home. The process first queried the current temperature of each room. Because the temperature of each room was less than 18°, all the heaters were switched on and all windows were closed.



```
----------- Start the home discovery  ----------
 5 Heaters are found
 The Shower (shower)  is  found
 5 Lamps are found
 4 Windows are found
 4 Shutters are found
 4 Rooms are found
 The Door (entranceDoor)  is  found
 The Calendar (calendar)  is  found
----------- The Discovery is finished ----------

Which temperature do you consider to be cold at your home?
 18
Which temperature you do consider to be hot at your home?
 30
What is your preferred temperature?
 22
How many minutes do you need for your Shower?
 5
Would you like to use this heater?
SmartEntity: Heater    Location: D
 Yes
Would you like to use this heater?
SmartEntity: Heater    Location: A
 Yes
Would you like to use this heater?
SmartEntity: Heater    Location: A
 Yes
Would you like to use this heater?
SmartEntity: Heater    Location: B
 Yes
Would you like to use this heater?
SmartEntity: Heater    Location: C
 Yes
```

Figure 6.5: User preferences setting in energy saving scenario

Table 6.3: An extract of the execution history of the saving energy scenario

Time	Event	Received data					Room	Query				SmartEntity	Changed state
		daytime	homeState	temperature	presence	faucetState		homeState	daytime	Temperature	Presence		
13:22:09	Process Starting	-	-	-	-	-	A	vacant	-	-	-	Lamps Heaters	off off
				-	-	-	B			-	-	Lamp Heater	off off
				-	-	-	C			-	-	Lamp Heater	off off
				-	-	-	D			-	-	Lamp Heater	off off
13:30:33	Access Event	-	occupied	-	-	-	A	-	-	15	-	Window Heater	closed on
				-	-	-	B			15	-	Window Heater	closed on
				-	-	-	C			15	-	Window Heater	closed on
				-	-	-	D			15	-	Window Heater	closed on
13:35:54	Presence Event	-	-	-	true	-	A	-	day	-	-	-	-
13:41:00	Presence Event	-	-	-	false	-	A	-	-	-	false	Lamp	wait 3 min off
13:45:00	Presence Event	-	-	-	true	-	C	-	day	-	-		-
13:50:09	Shower Event	-	-	-	-	opened	C	-	-	-	-	Shower	wait 10 min closed
14:10:04	Presence Event	-	-	-	false	-	C	-	-	-	false	Lamp	wait 3 min off
14:10:50	Presence Event	-	-	-	true	-	A	-	day	-	-	-	-
18:23:19	Calendar Event	night	-	-	-	-	A	-	-	-	true	Lamp Shutter	on closed
				-	-	-	B			-	false	Shutter	closed
				-	-	-	C			-	false	Shutter	closed
				-	-	-	D			-	false	Shutter	closed
18:00:21	Presence Event	-	-	-	false	-	C	-	-	-	false	Lamp	off
18:00:30	Presence Event	-	-	-	true	-	A	-	day	-	-	-	-
06:11:05	Calendar Event	day	-	-	-	-	A	-	-	-	-	Lamp Shutter	off opened
				-	-	-	B			-	-	Lamp Shutter	off opened
				-	-	-	C			-	-	Lamp Shutter	off opened
				-	-	-	D			-	-	Lamp Shutter	off opened

6.3.2 Security & Safety Scenario

The second scenario consists of controlling the home's security and safety, which mainly depends on controlling access to the home, detecting intrusion, fire, and inclement weather effects, and water leaks. It can be managed by the following strategies, some of which are proposed in [157]:

▪ **Case I: Control of Access**

- The entry door and windows should be closed and locked when the homeowner leaves home.
- Notify the homeowner via an SMS message if anyone enters the empty home.
- Activate the outside lights during the night to give the impression that the home is occupied.

▪ **Case II: Intrusion Detection**

When the user is away from home, he should be informed about any intrusion detected, whether in the form of movement or a broken door or window. The following actions can be taken:

- Alert the home owner or police if any window or door is broken or movement is detected.
- Activate the anti-theft alarm if any window or door is broken or movement is detected.

▪ **Case III: Weather Storm and Wind**

- Close the windows and shutters.
- Alert the homeowner.

▪ **Case IV: Fire Safety**

- Alert the homeowner and the fire department if a fire is detected.
- Switch the fire alarm on.
- Turn on all the anti-fire water system (fire sprinklers*)* in fire detected accidents.
- Turn on all room lighting.
- Unlock doors and windows.

▪ **Case V: Leak Detection**

- Alert the home owner if any water leak is detected. Sensors in the garage and kitchen can detect a water leak from the washing machine, dishwasher or water heater.

6.3.2.1 **Programming methodology**

By following the programming methodology applied in the energy saving scenario, the security and safety scenario can be redrafted, as shown in Listing 6.3, which gives a high-level description (pseudo-code) of the necessary queries and actions, with involved smart entities grouped under the following eight event handlers.

1. **MotionEvent:** a motion event (lines 1-6) is produced when any movement is detected around the motion sensor. The motion event informs about the change in the `motionState` (i.e. a `true` value if a movement is detected), and the location where this change has occurred.
2. **DoorBrokenEvent:** a door broken event (lines 7-10) is produced when a door is broken. It informs about the `dangerLevel` (i.e. a `high` value if a danger is detected), and the location where this danger has been detected.
3. **WindowBrokenEvent:** a window broken event (lines 7-10) is similar to a door broken event.
4. **AccessControl:** an access event (lines 11-17) is similar to the one introduced in the energy saving scenario.
5. **Windstorm Event:** a digital weather station informs the control system through a windstorm event (lines 18-21) when a windstorm is coming. It informs about the `dangerLevel` (i.e. a `high` value if the windstorm considered violent).
6. **FireEvent:** a fire event (lines 22-28) is produced when smoking or a fire is detected. It informs about the `dangerLevel` (i.e. a `high` value if a fire is detected), and the location where the fire has been detected.
7. **WaterLeakEvent:** a water leak event (lines 29-30) is produced when a water leak in the pipes is detected. It informs about the `waterleak` (i.e. a `true` value if a lack is detected) and the location where the lack has been detected.
8. **CalendarEvent:** a calendar event (lines 31-35) is similar to the one introduced in the energy saving scenario.

```
When a MotionEvent is produced
  query the homeState from the EntranceDoor
  if the home is in vacant mode
    notify homeowner by sending an SMS message via the Telephone
    call the police via the Telephone
    switch AntitheftAlarm on

When a Window or a DoorbrokenEvent is produced
  notify the homeowner through an SMS message using the Telephone
  call the police using the Telephone
  switch the AntitheftAlarm on

When an AccessEvent is produced
  if the inhabitant leaves home
    close and lock the EntranceDoor
    close and lock the windows
  if the inhabitant enters the home
    if the access code is not equal to the homeowner's access code
      notify the homeowner via an SMS message using the Telephone

When a WindstormEvent is produced
  close the Shutters
  close and lock the Windows
  alert the user via an SMS message using the Telephone

When a FireEvent is produced
  switch the FireAlarm on.
  open all the Sprinklers
  switch on all the Lamps
  unlock all the Doors and Windows
  alert the home owner via an SMS message using the Telephone
  alert the fire department via a Telephone call

When a WaterLeakEvent is produced
  notify the homeowner via an SMS message using the Telephone

When a CalendarEvent is produced
  if it is nighttime
    switch garden Lamp on
  else
    switch garden Lamp off
```

Listing 6.3: Pseudo-code concerning the security and safety scenario

6.3.2.2 Security Scenario Implementation

Listing 6.4 shows how the security and safety scenario is written in a GPL4SRE. Decisions and actions are grouped according to the eight event handlers, as follows.

1. The main thread (lines 29-54) consists of a **`pick`** statement which contains three events handlers: **`MotionEvent`**, **`DoorBrokenEvent`** and **`WindowBrokenEvent`**. As can be noted, the three events lead to the same actions and decisions, which consist of alerting the homeowner and calling the police. To avoid writing the code concerning sending an alert message and making a call in each event handler, the three events handlers are declared in the **`pick`** statement while the two actions are executed sequentially after the **`pick`** block.

2. The remainder event handlers (**`AccessEvent`**, **`WindstormEven`**, **`FireEvent`**, **`WaterLeakEvent`**, and **`CalendarEvent`**) are declared in parallel (lines 55-115) to the

main thread. Lines 55-71 specify an **`AccessEvent`** handler, which is executed when an access event is detected in the following order.

- Line 56 verifies whether the home is occupied or empty.
- If the home is empty, lines 57-62 are executed consisting of closing and locking the entrance door and the four windows.
- If the home is occupied, lines 65-69 are executed consisting of checking that the **`accessCode`** received by the event message is equal to the homeowner **`accessCode`**. If not, the homeowner is alerted via an SMS message.

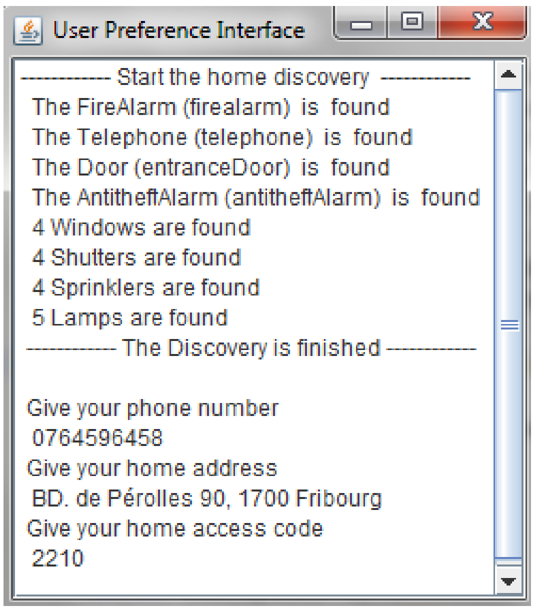
```
Process  SecurityandSafetyScenario {

  FireAlarm firealarm  = discover(FireAlarm);
  Telephone telephone  = discover(Telephone);
  Door entranceDoor    = discover(Door,"EntranceDoor_1");
  AntitheftAlarm antitheftAlarm = discover(AntitheftAlarm);

  List windows    = discover(Window);
  List shutters   = discover(Shutter);
  List sprinklers = discover(Sprinkler);
  List lamps      = discover(Lamp);

  subscribe MotionEvent;
  subscribe DoorBrokenEvent;
  subscribe WindowBrokenEvent;
  subscribe AccessEvent;
  subscribe WindStormEvent;
  subscribe WaterLeakEvent;
  subscribe FireEvent;
  subscribe CalendarEvent;

  String phoneNumber = "";
  String accessCode = "";
  String homeAddress = "";

  preferences{
    print("Give your phone number");
    set(phoneNumber);
    print("Give your home address");
    set(homeAddress);
    print("Give your home access code");
    set(accessCode);
  }

  sequence {
    while (true){
      String alertMessage = "";
      pick{
        onEvent(MotionEvent, Map input1){
          Map result = query(entranceDoor, "getHomeState");
          if(result.get("homeState") == "vacant"){
            alertMessage = "Somebody entered the following address "
                          +homeAddress;
          }
        }
        onEvent(DoorBrokenEvent, Map input2){
          alertMessage = "a door is broken in the following address "
                        +homeAddress;
        }
        onEvent(WindowBrokenEvent, Map input3){
          alertMessage = "a window is broken in the following address "
                         +homeAddress;
        }
      }
      if(alertMessage != ""){
        action(antitheftAlarm, "switchOn");
        Map input2 = {("message",alertMessage),("phoneNumber",
                       phoneNumber)};
        action(telephone,"sendSMS",input2);
        Map input3 = {("message",alertMessage),("phoneNumber","117")};
        action(telephone,"eCall",input3);
      }
    }
  }

  onEvent(AccessEvent, Map input4){
    if(input4.get("homeState") == "vacant"){
      action(entranceDoor,"close");
      action(entranceDoor,"lock");
      for (Window w: windows){
        action(w,"close");
        action(w,"lock");
      }
    }
    if(input4.get("homeState") == "occupied"){
      if (input4.get("accessCode") != accessCode){
        Map message = {("message","some body entered your home"),
                   ("phoneNumber",phoneNumber)};
        action(telephone,"sendSMS",message);
      }
    }
  }

  onEvent(WindStormEvent, Map input5){
    for(Shutter s : shutters){
      action(s,"close");
    }
    for(Window w : windows){
      action(w,"close");
      action(w,"lock");
    }
    Map input6 = {("message","a windstorm is coming"),
                ("phoneNumber", phoneNumber)};
    action(telephone,"sendSMS",input6);
  }

  onEvent(FireEvent, Map input7){
    action(firealarm,"switchOn");
    for (Sprinkler sp : sprinklers){
      action(sp,"open");
    }
    for (Lamp lamp : lamps){
      action(lamp,"switchOn");
    }
    for (Window w : windows){
      action(w,"unlock");
    }
    action(entranceDoor,"unlock");
    Map input8 = {("message","a fire is detected in your home"),
                  ("phoneNumber",phoneNumber)};
    action(telephone,"sendSMS",input8);
    String alertMessage = "A fire is detected at "+homeAddress;
    Map input9 = {("message",alertMessage),("phoneNumber","118")};
    action(telephone,"eCall",input9);
  }

  onEvent(WaterLeakEvent, Map input10){
    String alertMessage = "Water Leak is detected at "
                          +input10.get("locationID");
    Map input11 = {("message",alertMessage),("phoneNumber",phoneNumber)};
    action(telephone,"sendSMS",input11);
  }

  onEvent(CalendarEvent, Map input12){
    Lamp lamp = getEntityByID(lamps,"GardenLamp_1");
    if(input12.get("dayTime") == true){
      action(lamp,"switchOff");
    }else{
      action(lamp,"switchOn");
    }
  }
}
```

Listing 6.4: The security and safety scenario written in GPL4SRE

6.3.2.3 **Scenario Execution Monitoring**

The execution lifecycle of the security and safety scenario is similar to the energy saving scenario.

1. First, the process interacted with the discovery engine to find declared smart entities. According to the result shown in Figure 6.6, 21 smart entities involved in the scenario were found.
2. Then, the process registered in the EDN as a new subscriber to be informed about all events declared in the security scenario (lines 10-17).
3. Then the user preferences setting part started (lines 21-28) by asking the user his phone number, address and home access code (see Figure 6.6).
4. The process then executed the main thread. No initial actions are preformed; the main thread is able to respond to three kinds of events (i.e. **`MotionEvent`**, **`DoorBrokenEvent`**, or **`WindowBrokenEvent`**).
5. In parallel to the execution of the main thread, the process was able to respond to other events through the event handlers. Table 6.4 shows an extract of the execution history of the current scenario. The running scenario reacted as expected to the different events. Each time, an event was produced, the appropriate actions and queries were performed according to their order in the program. For example at 08:05:20, an **`AccessEvent`** was produced to inform the control system the home was in a **`vacant`** mode. According to the new environment context, the entrance door, and four windows were closed and locked.



```
------------ Start the home discovery ------------
 The FireAlarm (firealarm) is found
 The Telephone (telephone) is found
 The Door (entranceDoor) is found
 The AntitheftAlarm (antitheftAlarm) is found
 4 Windows are found
 4 Shutters are found
 4 Sprinklers are found
 5 Lamps are found
------------ The Discovery is finished ------------

Give your phone number
 0764596458
Give your home address
 BD. de Pérolles 90, 1700 Fribourg
Give your home access code
 2210
```

Figure 6.6: User preferences setting in security and safety scenario

Table 6.4: An extract of the execution history of security and safety scenario

Time	Event	Received data				Room	Query	SmartEntity	Changed State
		homeState	dangerState	presenceState	daytime		homeState		
08.00:00	Process Starting	-	-	-	-	A	-	-	-
				-	-	B		-	-
				-	-	C		-	-
				-	-	D		-	-
08.05:20	AccessEvent	vacant	-	-	-	A	-	Entrance Door	closed, locked
								Window	closed, locked
				-	-	B		Window	closed, locked
				-	-	C		Window	closed, locked
				-	-	D		Window	closed, locked
18:22:22	CalendarEvent	-	-	-	night	E	-	Lamp	on
06:15:00	CalendarEvent	-	-	-	day	E	-	Lamp	off
08:20:00	MotionEvent	-	-	true		A	occupied	AntitheftAlarm	On
								Telephone	eCall
11:32:00	DoorBroken Event	-	high	-	-	C	-	AntitheftAlarm	On
								Telephone	eCall
11:33:50	WindowBroken Event	-	high	-	-	B	-	AntitheftAlarm	On
								Telephone	eCall, sendSMS
14:08:12	AccessEvent	occupied	-	-	-	D	-	Telephone	sendSMS
18:07:00	Windstorm Event	-	high	-	-	A	-	Entrance Door	closed, locked
								Window	closed, locked
				-	-	B		Window	closed, locked
				-	-	C		Window	closed, locked
					-	D		Window	closed, locked
20:00:00	FireEvent	-	high	-	-	A	-	FireAlarm	on
								Telephone	eCall, sendSMS
								Entrance Door	unlocked
								Window	unlocked
				-	-	B		Window	unlocked
				-	-	C		Window	unlocked
					-	D		Window	unlocked

6.4 Discussion and Evaluation

This section discusses the utility and benefits of using the GF4SRE framework and the GPL4SRE language by evaluating the two scenarios implemented. Further, a discussion of good programming style when using GPL4SRE is outlined.

6.4.1 **Scenarios Evaluation**

The evaluation is discussed in term of the capabilities, abstraction and transparency, expressiveness, and reusability criteria compared to other languages like BPLE4WS. Some evaluation criteria are similar to those in the evaluation carried out by [158].

6.4.1.1 **Capabilities**

The introduced scenarios demonstrate that to control SREs, the decision-making process requires co-ordination between the smart entities populating the environment. The control system should have the ability to acquire the current context of the environment, react to new changes, and provide a set of actions in order to achieve the final goal such as home security or saving energy. The two scenarios revealed how GPL4SRE capabilities are designed to fit these requirements. Table 6.5 summarises the characteristics and capabilities of GPL4SRE compared with a traditional composition language like BPEL4WS. The following GPL4SRE characteristics can be identified:

- It can be seen in the energy saving scenario (lines 2-14), and in the scenario of security (lines 2-17), that GPL4SRE enables the automatic discovery of smart entities as well as the subscription to different events.
- The energy saving scenario (lines 20-33) and scenario of security (lines 21-28) show how to involve the final user in the process through an ask-response dialogue for data, and the choices available to satisfy user needs.
- It can be noted in the energy saving scenario (lines 37, 38, 46) and in the security scenario (lines 30, 35, 56) that with the GPL4SRE, it is possible to create loops, declare and assign variables, as well as offer control flow constructs (**`while`**, **`if`**, etc.) thereby enabling the creation of complex control scenarios.
- As can be noted in the energy saving scenario (lines 35, 40, 48, 53), and in the security scenario (lines 34, 47, 51, 61), GPL4SRE enables interactions with different smart entities, either in the form of actions or queries. This allows the control process to collect surrounding information from the environment to deduce the current environment situation and take suitable actions to change the appropriate states.
- As noted in the energy saving scenario (lines 70-93, 94-121, 122-127, 128-153 and 154-159) in the security scenario (lines 55-71, 72-83, 84-101, 102-107, 108-115), GPL4SRE allows different produced events to be handled in parallel. This enables the control process to be informed about changes in the environment rather than having to frequently ask if any significant change has occurred.
- As shown in the energy saving scenario (line 39-41), and the security scenario (lines 59-62), GPL4SRE enables the processing of individual smart entities and the services returned by service discovery queries using a **`for`** statement. This kind of cursor allows

the whole service sets to be manipulated at the same time, a capability that traditional composition languages - like BPEL - lack.

- Although, fault handlers are not directly supported in this version of GPL4SRE, the following precautions are considered appropriate:
  - An automatic fault handling in the situation where an unavailable smart entity is invoked. If a query fails to be performed, the returned result is considered to be null.
  - Any declared list is considered empty if no smart entities are found.
  - An automatic check made for each entity if it is not set to `null` before performing an action or query.

Table 6.5: A comparison of constructs and syntax of GPL4SRE and BPEL4WS language

Metric	**GPL4SRE**	**BPEL4WS**
Automatic discovery	✓	×
Event registry	✓	✓[a]
Automatic event registry	✓	×
User preferences	✓	×
Variables declaration	✓	✓[b]
Control flow	✓	✓
Entities declaration	semantic-based	syntactic-based
Services invocation	✓	✓[a]
Event handlers	✓	✓
Iteration over a list of smart entities	✓	×
Fault handlers	×	✓

✓[a] Programmer must manually bind each service and each event.

✓[b] Variables in BPEL are basically XML data.

6.4.1.2 **Abstraction**

The abstraction aims to introduce the environmental conceptual model into the GPL4SRE programs by linking the domain knowledge to the source code. This helps the programmer to program in the same way as the environment is modeled and facilitates the writing, understanding, and maintenance of the GPL4SRE scenarios, which would be harder using general-purpose languages like BPEL4WS. The abstractions carried out by the GF4SRE framework include:

1. ***Smart entity declarations***: As illustrated in the energy saving scenario, Listing 6.2 (lines 2-9), and the security scenario, Listing 6.4 (lines 2-9), the smart entity declarations correspond to their definition in the Ont4SRE ontology. The notions of **`Heater`**, **`Room`** and so on are used rather than the syntax implantation notions like **`partnerLink`**.
2. ***Services declarations***: In the energy saving scenario (lines 35, 43, 53), and the security scenario (lines 34, 47, 87) the notions of **`getHomeState`**, **`switchOn`** and so on are used for the name of the invoked service, which corresponds to their definitions in the Ont4SRE ontology rather than using their implementation syntax names.
3. ***Smart entities properties***: In the energy saving scenario (lines 36, 38, 67) and security scenario (lines 35, 48, 96) the entities properties (i.e. **`homeState`**, **`phoneNumber`**, **`vacant`**, etc.) are declared based on their definition in the Ont4SRE ontology rather than using service input implementation syntax names.

6.4.1.3 **Transparency**

The transparency discussed here means that any physical or software component should hide its implementation nature from its users, appearing and being used in a uniform and individual way. The transparency carried out by the GF4SRE framework includes:

1. *Discovery transparency*: The discovery of smart entities and event registry is completely automatic and performed in a transparent manner. This includes hiding the implementation syntax and grounding details of each smart entity. As illustrated in the energy saving (lines 2-9) and security (lines 2-9) scenarios, the **`discover`** method is used to find the different smart entities by specifying only their types (i.e. **`Room`**, **`Door`**, **`Telephone`**) and locations (optional), while discovering and linking the smart entities with their services is fully hidden, as the programmer should not have to be aware of that.
2. *Access and interaction transparency*: smart entities and their associated services are clearly represented to the programmer. Users declare them rather than specify where they are (i.e. WSDL URLs) or how they are implemented. With a traditional composition language like BPEL4WS, the programmer needs to specify all the grounding details related to each service. Listing 6.5 compares the two programs' syntax to switch a heater on extracted from the energy saving scenario written in GPL4SRE and BPEL4WS. The number of implementation details that the programmer must know and specify a priori in order to perform a **`switchOn`** action for a heater using BPEL4WS can be seen. The WSDL **`URL`** address (line 3), the **`portType`** (line 8) and the operation implementation parameters (line 9) should be known a priori, while with GPL4SRE, thanks to Ont4SRE, declaration and invocation are semantically annotated and automatically discovered in regard only to the entity type (line 11) and the semantic description of the service (line 12) according to the Ont4SRE definitions.

```
 #a- BPEL4WS
<wsdl:import namespace="http://hall.ws/"
  location="http://diufpc10.unifr.ch:8080/.../ParentHeater?wsdl"/>
<partnerLink name="Heater"
    partnerLinkType="http://parent.room.ws:ParentHeater_PL"
    partnerRole="ParentHeater_Role"/>
<invoke name="setTemperature" partnerLink="Heater"
   portType="http://parent.room.ws/ParentHeater"
   operation="switchOnHeater" inputVariable="inputVariable"/>

# b- GPL4SRE
Heater heater=discover(Heater, "Heater-B");
action(heater, "adjustTargetTemperature", inputVariable);
```

Listing 6.5: A syntax comparison of service invocation using GPL4SRE and BPEL4WS

6.4.1.4 Expressiveness

Ranging from the SRE formulization via the Ont4SRE's concepts and definitions for the GPL4SRE syntax, there is continuity and consistency in concepts, terms and expressions used. The relations between the different components that form the SRE overall are clearly defined in the same way through the different scenario implementation stages. This way of designing outlines the way in which a developer should program. A GPL4SRE programmer should keep in mind that there are smart entities that can be discovered, their actual states can be changed or read via action and query services and changes in their states can be detected using event handlers to realize any control scenario. Table 6.6 illustrates how the implemented energy saving and security scenarios are expressive in themselves. In the two scenarios, 48 smart entities (e.g. **`Window`**, **`Door`**) keywords were used, the notion of **`action`** and **`query`** used 45 times and **`discovery`** and **`subscribe`** keywords used 29 times. Further, each individual statement and code line has a meaning and is understandable by itself. Approximately 47% of the words and terms used in the two scenarios are either ontology-based or belong to the GPL4SRE keywords.

6.4.1.5 Reusability

The reader can note that in the two scenarios, adding a new smart entity like heater, window, or a motion detector in any room will have no negative influence or impact on the program and these new smart entities will be discovered and involved in the scenario once the scenario is rerun. Going further, the two scenarios can be implemented in any home occupied by similar smart entities and services without configuring any device or providing any new implementation details. These features are due to three reasons.

1. The using of semantic-based declarations for each smart entity and its associated properties and services rather than declaring the syntax of their implementations.
2. The Ont4SRE links all the smart entities and their associated components, enabling automatic discovery and registry.
3. The different services are completely loosely coupled.

Table 6.6: An evaluation of the number of keywords used in the two scenarios

Keywords Excluding the following symbols ;{}:() = 0…9.		**Times**			
		Scenario I (446 words)		**Scenario II** (345 words)	
Ontology-based	entity type (e.g. `Window, Door`)	29	6.50%	19	5.51%
	entity state (e.g. `temperature, lockState`)	30	6.73%	14	4.06%
	service name (e.g. `close, getTemperature`)	21	4.71%	23	6.67%
	event type (e.g. `MotionEvent`)	10	2.24%	16	4.64%
Language symbols	datatype `(e.g. float, Map, List)`	25	5.61%	23	6.67%
	`discover, subscribe, preferences`	13	2.91%	16	4.64%
	`action`	16	3.59%	22	6.38%
	`query`	6	1.35%	1	0.29%
	`onEvent`	5	1.12%	10	2.90%
	`if, for, while`	38	8.52%	12	3.48%
	`pick, flow, sequence, wait, print, set etc.`	14	3.14%	9	2.61%
Total		207	46.41%	165	47.83%

6.4.2 Lessons learnt and Good Programming Styles for GPL4SRE

This section reviews some good programming styles from the two presented scenarios, which can serve as recipes or a cookbook of how to program and build good control scenarios.

Using an event-driven approach: The control system intervention is often related to a particular situational change in the environment. Unlike the scheduling control style, the system does not intervene according to a predefined time schedule (i.e. control every 5 minutes if the temperature changes). Instead, the system can be informed automatically about any change that might occur using a combination of events generated by a set of sensors. This style of programming is called an event-driven approach. In this kind of programming, appropriate decisions and actions are grouped under event handlers depending on how each event effects the environment.

Querying the actual state of the environment: sometimes being informed about any change occurring in the environment is not enough to have a complete understanding about the actions and decisions which need to be taken. For example, when the temperature decreases in a given room, knowing whether the room is occupied or not is crucial in order to decide the level of comfort required. To ensure information about the current state of the environment is available during the decision making process, the programmer should query and gather information about the required states and concerning the decision process. In the temperaturedecreasing example, the control system should perform a query `getPresenceState` to find out if the room is occupied or not before adjusting the heater's temperature.

Avoiding code duplication: an environment characterised by dynamic changes in its situation means that different situations could lead to the same actions and decisions taken by the control system. For example, if a window or a door is broken, in both cases, the control system should alert the homeowner by sending an alert message. To avoid duplication of the code, similar actions can be shared by different events. For example, to avoid using `sendSMSmessage` action twice, the programmer can involve the two events handlers in a `pick` statement while the action `sendSMSmessage` can be executed sequentially after the `pick` block, as illustrated in Figure 6.7.

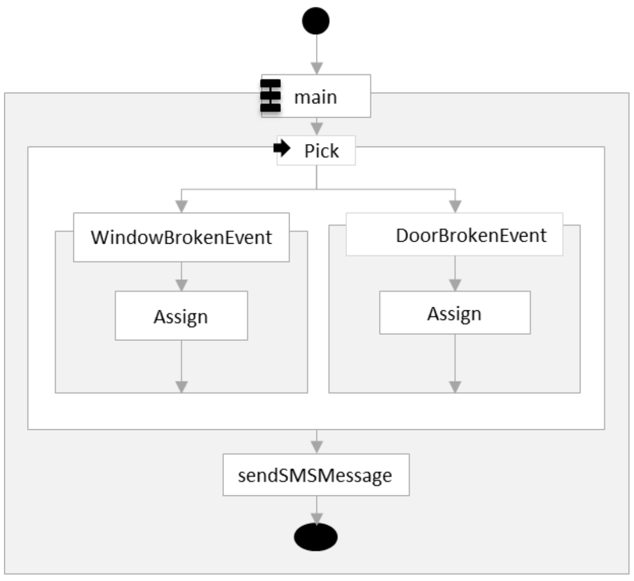


Figure 6.7: An example of using pick construct to avoid code duplication

The effects of using a `wait` statement: the `wait` construct is used to force the process to wait a specified amount of time. Listing 6.2, line 124 shows an example of using `wait` to make the process of waiting a predefined period before closing the shower. All activities declared after this construct are executed after the `wait` statement has been executed (i.e. the expiry of a predefined amount of time). The programmer should be aware of the consequences of deploying this construct. Consider the scenario when the system is programmed to react to a temperature event during the execution of the `wait` statement. To do this, the `TemperatureEvent` handler must be declared in parallel to the block that contains the `wait` statement.

Using predefined methods: GPL4SRE provides a set of predefined methods that could reduce the program code's size and facilitate programming. For example, the `getEntityByLocation` method is introduced to get a specific entity in a given location rather than iterating over a list of smart entities to find it. This method takes as input parameters, a list of the smart entities and the location name and returns the required smart entity. Other methods introduced in Appendix C can be used for similar proposal. Additional methods are the subject of future work. Consider, for example, a room occupied by many temperature sensors and we want to calculate the average temperature of these sensors. Programming this task with the GPL4SRE requires many code lines. One solution is to have a predefined method such as the following:

```
AVG(query(rooms, "getTemperature"), "temperature").
```

It takes as input parameters the list of smart entities, the target service and the variable from which the average should be calculated. Other methods like **`SUM`**, **`isAllTrue`** and so on can be proposed to facilitate GPL4SRE programming.

Initializing the process: sometimes initial actions need to be taken when the program starts. Such actions are usually declared in the main thread of the program.

Programming generic and specific scenarios: there are two possible contexts in which the programmer can write a GPL4SRE scenario.

- The first context is when the programmer makes a control scenario for a specific home. She knows exactly the kind of smart entities and their services occupying the space. This is similar to the two previous scenarios. Therefore, an ontology instance is available and can be used to specify the possible queries, actions, and decisions. The programmer has the ability to find a particular smart entity and know which queries and actions can be performed according to the available smart entities in each location. Further, there is no risk that the smart entities involved cannot be found in the space.
- The second context is when the programmer makes a control scenario for a SRE and she has no idea about what exists in the space, but guessed which kind of smart entities occupy the space without specifying their number. In this case, the programmer will not be able to discover a particular smart entity or distinguish the different locations. One solution is to program in a generic way by using the notion of a list. Each time an event is supposed to be produced in a given location, the program will receive the **`locationID`** of that location. With this key element, the programmer is then able to determine all the smart entities that occupy that location and perform the suitable actions and queries. As mentioned in Subsection 6.4.1.1 concerning the fault handling issues, with GPL4SRE when a smart entity list is declared but not found, it is considered empty and no iteration over that list can be performed. Consider, for example, the programmer's intention is to switch off all the lamps in each room when it is unoccupied. To do this, the programmer declares a list of smart entities of type **`Lamp`** and a **`PresenceEvent`** handler. When an inhabitant leaves a given room, a **`PresenceEvent`** is supposed to be produced. Once the program is notified about this change and the **`locationID`** is given, all the lamps located in the specified location can be switched off. Listing 6.6 shows how to program for such a scenario. During the execution of this piece of program for a given home, it might be there

is no presence detection sensor installed or the lamps are not connected via the network. The impact of these two problems on the program is as follows:

1. Concerning the unavailability of the **`PresenceEvent`** this means that the entire event handler block (lines 2-10) will never be executed, since such an event is impossible.
2. Concerning the lamps unavailability, there are two possibilities. Either all the lamps are not connected, in which case, lines 4-8 will not be executed because the list of the lamps is empty, or the lamps do not exist in the location where the event has been produced so line 6 will never be executed because the **`if`** statement will never be evaluated as **`true`**.

```
1    List lamps = discover(Lamp);
2    onEvent(PresenceEvent, Map input1){
3      if (input5.get("presenceState") == false){
4        for(Lamp lamp: lamps){
5          if( lamp.getLocation.get("ID") == input1.get("locationID")){
6            action (lamp, "switchOff");
7          }
8        }
9      }
10   }
```

Listing 6.6: A generic scenario to control the light

6.5 Summary

This chapter has demonstrated the feasibility and utilities of the proposed framework and its related components through a case study in a smart home. The different steps for realizing and developing the case study were illustrated. The implemented scenarios brought out the following elements which must be considered as added value:

- Defining a conceptual model for describing the SRE and transforming it into the Ont4SRE ontology facilitates discovery and control in the proposed framework. To use the framework, two reasonable conditions are required.
  1. The existence of an appropriate ontology for the chosen environment (home, hospital, school, etc.), and
  2. The existence of a declarative semantic description using the exact same terminology as defined in the Ont4SRE ontology for each web service provided by the concrete actuators and sensors.
- The power of the framework and its GPL4SRE language is in reducing the complexity of controlling SREs through reducing the program code size, increasing expressivity and the abstraction, and automating the discovery of the smart entities.

- Choosing a good programming style to realize control scenarios guarantees the avoidance of many concurrent programming challenges and duplications of code in the SRE characterized by context change and concurrently produced events.

7 Conclusion and Future Work

This thesis has described the design and implementation of a framework for controlling a smart residential environment. First, the necessary definitions, characteristics, and the utilities of the smart environment were clearly outlined. Then a survey of the use of emerging technologies and the identification of the main problems, especially those related to modelling and controlling the SRE, were provided.

The remainder of this chapter presents a summary of the contribution of this thesis and discusses the remaining challenges and future tasks.

7.1 Contributions

The core of this thesis is the proposal of the GF4SRE framework and its pluggable software architecture to support the integration of physical objects into the digital world using proven technologies to facilitate the definition and execution of complex control scenarios in the context of SREs. This involves reducing the complexity of the SRE infrastructure based on the separation of concerns, modularization, and facility for changing and assembling modules. A brief overview of the benefits of the GF4SRE components is summarized as follows.

- The Ont4SRE ontology is proposed to model and share a common presentation of the SRE, including the smart entities, locations, associated embedded devices (sensors, actuators), services (action, query, and event) and the relationships between them. This provides a significant basis for defining and finding the smart entities occupying the space and how it is used in term of services. In fact, it shows the whole potential of a standard query language such as SPARQL when introduced into the discovery and control of SRE. Describing a SRE using the Ont4SRE ontology increased the level of transparency and abstraction for both the discovery and control of different SREs - home, school, hospital.
- To address the accessibility and interoperability issues in the SRE context, the proposed framework architecture is based on actual state of the arts standards: the services proposed

by the various sensors and actuators are described using proven technologies like WS-* and OWL-S. Integrating these technologies brings the following advantages:

- Considering the smart entities and their devices as services providers and exposing their capabilities as web services correspond to how the smart environment is defined and modeled using the Ont4SRE ontology and validates (enforces) the idea of applying the service oriented paradigm to a heterogeneous dynamic environment like SREs.
- Smart objects can publish their functionalities in the form of actions, queries and events via a web services description language (WSDL) interface. This forces the loose coupling and a clean separation between the framework modules.
- Adding machine interpretable semantic annotations to web service content helps to understand their capabilities and distances the programmer from the complexity of the implementation details related to each web service and simplifies the completion of complex control scenarios.

▪ The registration and discovery components proposed take into account the particular context of SRE. The Ont4SRE is used as a database store for modeling and describing the SRE. The lookup infrastructure allows the running of the search queries implemented using the SPARQL language to find relevant smart entities and their services. The discovery and registration queries are performed automatically and presented to the user in an abstract and transparence manner.

▪ Control scenarios in the form of templates to manage the SRE are proposed. The templates are workflow-based and written by an original process oriented language named GPL4SRE**.** Further, a process execution engine, which can execute these scenarios, is provided.

▪ Another fundamental component is the GPL4SRE language, presented in Chapter 5, to write the control templates. This language has a well-defined general structure which is divided into two distinct parts. First, the declaration section uses the ontology vocabulary to discover transparently the entities in the smart environment and their associated services. Secondly, the execution section offers, in a simplified manner, all the essential control and execution structures of a well-established process control language such as BPEL4WS. The power of the GPL4SRE language is that it is designed to bridge the gap between natural human thinking and the syntax and expression of the control programs. The control scenarios are designed and written in an understandable fashion, with fully meaningful phrases. It offers considerable gains in expressiveness and simplicity compared with other process-oriented language like BPLE4WS.

To validate the proposed approach, it was implemented and tested in term of feasibility, capabilities and benefits. Two implemented control scenarios illustrated the different development stages: modelling the SRE using the Ont4SRE ontology, discovery, and scenario running lifecycle. Overall, the results demonstrate that the GF4SRE framework capabilities are designed to fit the SRE requirements in term of accessibility, discovery, context-awareness, and ability to combine events and actions to create complex control applications. The GF4SRE architecture significantly simplifies the development of control applications for the SRE by increasing abstraction, transparency, expressivity and reusability.

7.2 Open Challenges and Future Work

This thesis proposed a holistic framework to deal with challenges related to describing and controlling the SRE, but also emphasized other important challenges and perspectives that could be pursued.

Conflict of Interest between Multi Scenarios: By comparing the two scenarios introduced in Chapter 6, it can be noted in some cases, that opposite actions can be performed in the two scenarios - energy saving and security - although they relate to similar reasons. For example, in the energy saving scenario, the control system should turn all the lamps off if nobody is at home, while in the security scenario, the control system should turn some lamps on at night if nobody is at home, to give the impression that the home is occupied. This kind of conflict can be either detected by the programmer and one solution is to merge the two scenarios into one, or other solutions can be proposed like in [159, 160] where the notion of priority and preferences are introduced. However, in this thesis, this issue was not considered as part of the investigated challenges. It is not recommended to run two scenarios at the same time since this may cause a conflict between them.

Conflict of Interest between Multi-users: Another conflict concerning multi-users and shared spaces was explored in [161, 162]. This kind of conflict is due to the different preferences and needs of the users that share a given space. [163, 164] give more details about the nature of such conflicts in ambient intelligence systems since an in-depth analysis of this topic is outside the scope of this work. As far as this thesis is concerned, users can be attached to RFID tags to be automatically identified by the control system. Consider a presence detection sensor that is not only able to detect if anybody enters the room, but also is able to read the RFID tags, which lead it to know who entered the room. Once the inhabitant is identified, it is up to the programmer to decide how to perform the appropriate action to meet the identified user's needs.

Failure and Fault Handlers: The process execution engine, network, and devices can fail for several reasons, such as when a shutdown occurs or a battery loses power. In this work, three kinds of failure can be identified during the execution of a control scenario.

1. The first failure is when the control system is completely shut down. One of the common solutions to such problem is based on redundancy, to increase the reliability of the system, for example, by having an automatic failover control system. By using automatic detection, an error in the primary control system can be detected and the redundant

control system will take over control automatically. The interested reader is referred to [165] for a thorough discussion of this issue.

2. The second failure consists of the errors, especially those thrown inside the process itself and which cause the entire process to crash. Providing error recovery to catch such errors is a commonly adopted solution, like that implemented within BPEL4WS [166]. It is just a question of implementation to integrate similar mechanism into GPL4SRE.

3. The third failure is related to the sensors and actuators being unavailable for different reasons like device damage, malfunctions or cable break. The first challenge related to this failure is how to alert the discovery engine and the involved processes about the devices' unavailability to avoid errors like network timeouts and faults in the invoked services. The second problem is how the process should react in this circumstance. [69, 167-169] proposed different ideas to deal with these problems like:

   - The use of a number of small sensors elements in order to form a single sensor which has intrinsic fault tolerance.
   - Implementing a hidden fault handler for each single invoke operation (action or query). The fault handler program tries to invoke the target several times (with a maximum number of attempts). If this does not solve the problem, the services may be substituted by an equivalent one.
   - A dynamic discovery can solve the problem of removing and adding sensors and actuators to the networks. The WS4D initiative proposed for embedded devices provides such capabilities, where clients and devices are able to exchange a Hello-Bye message each time a device joins or leaves the network.

Appendices

A Abbreviations

This appendix gathers the abbreviations used in this thesis and their corresponding meanings.

Abbreviation	**Meaning**
API	Application Programming Interface
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
BPMN	Business Process Model and Notation
CORBA	Common Object Request Broker Architecture
CRUD	Create, Read, Update and Delete
DAML	DARPA Agent Markup Language
DPWS	Devices Profile for Web Services
DSL	Domain Specific Language
EDN	Event Delivery Network
FTP	File Transport Protocol
GDL4WSAC	Goal Description Language for Semantic WS Automatic Composition
GENA	Generic Event Notification Architecture
GF4SRE	Generic Framework for Smart Residential Environments
GPL4SRE	Generic Process Language for Smart Residential Environment
GPS	Global Positioning System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HVAC	Heating, Ventilation, and Air Conditioning
HUD	Head-up Display
IoT	Internet of Things
ISO	Open Systems Interconnection
JLS	Jini Lookup Service
JVM	Java Virtual Machine
JSON	JavaScript Object Notation
LAN	Local Area Network
LJS	Jini Lookup Service
Ont4SRE	Ontology for Smart Residential Environment
OSGi	Open Service Gateway initiative
OWL	Web Ontology Language
OWL-S	Semantic Mark-up for Web Services
PDA	Personal Digital Assistant
PDDL	Planning Domain Definition Language

QoS	Quality of Service
RDF	Resource Description Framework
REST	Representational State Transfer (REST)
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAWSDL	Semantic Annotations for WSDL
SE	Smart Environment
SLP	Service Location Protocol
SMTP	Simple Mail Transfer Protocol
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
SRE	Smart Residential Environment
SSDP	Simple Service Discovery Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
UDDI	Universal Description, Discovery, and Integration
UPnP	Universal Plug and Play
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WLAN	Wireless Local Area Network
WoT	Web of Things
WS-*	Big Web Service
WS4D	Web Services for Devices
WSDL	Web Service Description Language
WSMO	Web Service Modelling Ontology
XML	Extensible Mark-up Language
XSLT	Extensible Stylesheet Language Transformations

B GPL4SRE: Types, Reserved Words, Operators, Methods

B.1 Types

`int, float, double, boolean, String, Map, List.`

B.2 Reserved Words

`process, discover, subscribe, float, int, double, boolean, String, Map, List, preferences, print, select, set, for, if, true, false, IF, if, while, pick, sequence, flow, action, query, onEvent, onAlarm, wait, terminate, empty.`

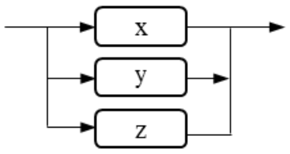
B.3 Operators Precedence in Expressions

`/ *`	multiplying operators
`+ -`	adding operators
`== < > =< >= !=`	relational operators
`&& ||`	logical operators

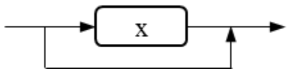
B.4 Predefined Methods

```
discover(entityType):Entity
discover(entityType, entityIdName):Entity
discover(entityType):List
discover(entityType, locationIdName):List
getEntityByID(entityListIdentifier, entityIdName) :Entity
getEntityByLocation(entityListIdentifier, locationIdName):Entity
```

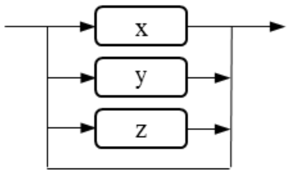
C GPL4SRE: EBNF Syntax



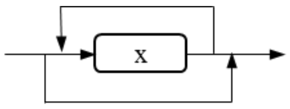
(«X»|«Y»|«Z»)
Match any alternative within the sub-rule exactly once.



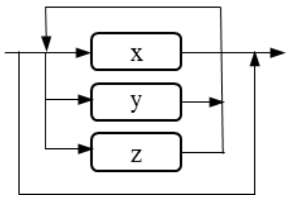
x?
Element x is optional.



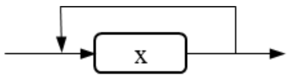
(«X»|«Y»|«Z»)?
Match nothing or any alternative within sub-rule.



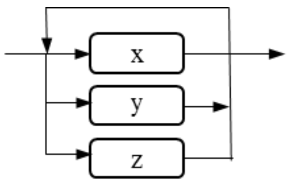
x*
Match element x zero or more times.



(«X»|«Y»|«Z»)*
Match an alternative within subrule zero or more times.



x+
Match element x one or more times.



(«X»|«Y»|«Z»)+

Match an alternative within subrule one or more times.

Figure C.1: EBNF grammar sub-rules (taken from [156])

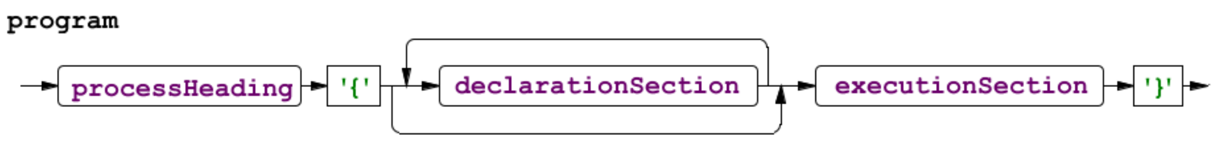
C.1 EBNF Syntax

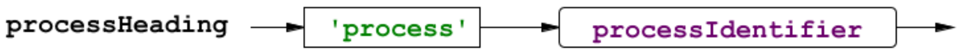
```
program                  : processHeading '{' declarationSection* executionSection '}' ;
processHeading           : 'process' processIdentifier ;
declarationSection       : entityDeclaration | entityByIdDeclaration | entityListDeclaration |
                           entityListByLocationIdDeclaration | eventRegistration | variableDeclaration;
entityDeclaration        : entityType entityIdentifier '=' 'discover' '(' entityType ')' ';' ;
entityByIdDeclaration    : entityType entityIdentifier '=' 'discover' '(' entityType ',' entityIdName ')' ';'
entityListDeclaration    : 'List' entityListIdentifier '=' 'discover' '(' entityType ')' ';' ;
entityListByLocationIdDeclaration :
                           'List' entityListIdentifier '=' 'discover' '(' entityType ','
                           locationIdName ')' ';' ;
eventRegistration        : 'subscribe' eventType ';' ;
variableDeclaration      : simpleVariable | complexVariable ;
simpleVariable           : dataType variableIdentifier ('=' expression)? ';' ;
complexVariable          : 'Map' variableIdentifier ('=' '{' mapPair (',' mapPair)* '}')? ';' ;
executionSection         : userPreferences? mainThread eventHandlers* ;
userPreferences          : 'preferences' preferenceBlock ;
preferenceBlock          : '{' ( print | set | select | ifStatement4Preference |
                           forStatement4Preference)* '}' ;
print                    : 'print' '(' STRING_LITERAL ')' ';' ;
set                      : 'set' '(' variableIdentifier ')' ';' ;
select                   : 'select' '(' entityIdentifier ')' ';' ;
ifStatement4Preference   : IF '(' conditionExpression ')' preferenceBlock elseStatement4Preference? ;
elseStatement4Preference : 'else' preferenceBlock ;
forStatement4Preference  : 'for' '(' entityType entityIdentifier ':' entityListIdentifier ')'
                           preferenceBlock ;
mainThread               : 'sequence' block ;
eventHandlers            : 'onEvent' '(' eventType ',' 'Map' variableIdentifier ')' block |
                           'onAlarm' '(' (('for' '(' duration')' | (
                           'repeatEvery' '(' duration ')')) ')' block ;
block                    : '{' (controlConstructs | executionConstructs | interactionConstructs |
                           otherConstructs | assignStatement )* '}' ;
controlConstructs        : ifStatement | whileStatement | forStatement ;
ifStatement              : IF '(' conditionExpression ')' block elseStatement? ;
elseStatement            : 'else' block ;
whileStatement           : 'while' '(' conditionExpression ')' block ;
forStatement             : 'for' '(' entityType entityIdentifier ':' entityListIdentifier ')'
```

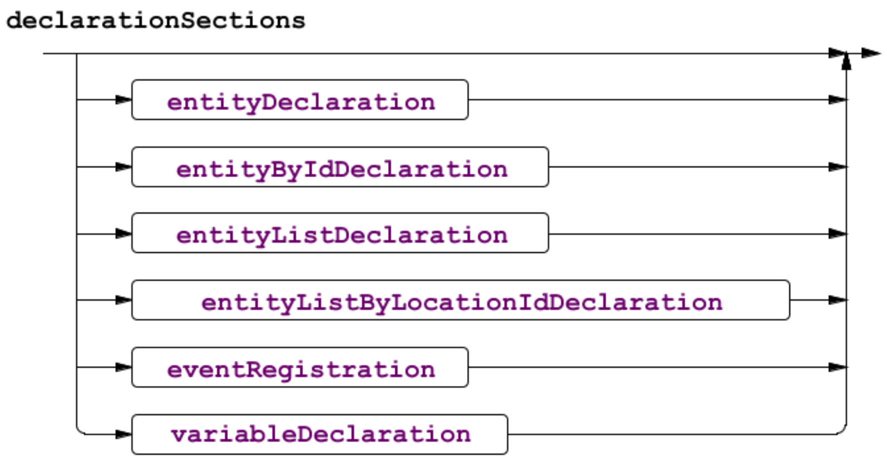
```
                          partial_block ;
partial_block           : '{' interactionConstructs | assignStatement | forStatement |
                          (IF '(' conditionExpression ')' partial_block) |
                          ('while' '(' conditionExpression ')' partial_block) '}' ;
executionConstructs     : sequenceStatement | flowStatement | pickStatement ;
sequenceStatement       : 'sequence' block ;
flowStatement           : 'flow' '{' ('sequence' block)+ '}' ;
pickStatement           : 'pick' '{' (onEventStatement | onAlarmStatement)+ '}' ;
onEventStatement        : 'onEvent' '(' eventType ',' 'Map' variableIdentifier ')' block ;
onAlarmStatement        : 'onAlarm' '(' 'for' '(' duration ')' ')' block ;
interactionConstructs   : action | query | onEvent ;
action                  : 'action' '(' 'this.'? entityIdentifier ',' serviceName ((','
                          'this.'?) variableIdentifier)? ')' ';' ;
query                   : 'Map' variableIdentifier '=' 'query' '(' ('this.')? entityIdentifier
                          ',' serviceName (',' ('this.')? variableIdentifier)? ')' ';' ;
onEvent                 : 'onEvent' '(' eventType ',' 'Map' variableIdentifier ')' ';' ;
otherConstructs         : waitStatement | terminateStatement | emptyStatement ;
waitStatement           : 'wait' '(' expression ')' ';' ;
terminateStatement      : 'terminate' '(' ')' ';' ;
emptyStatement          : 'empty' '(' ')' ';' ;
assignStatement         : assignSimpleVariable | assignComplexVariable | setMapVarParameter |
                          getEntityById | getEntityByLocation;
assignSimpleVariable    : (dataType | 'this.')? variableIdentifier ('=' expression)? ;
assignComplexVariable   : 'Map' variableIdentifier ('=' '{' mapPair (',' mapPair)* '}')? ';' ;
setMapVarParameter      : 'this.'? variableIdentifier '.' 'set' '('
                          parameterKeyName ',' expression ')' ';' ;
getEntityById           : entityType? entityIdentifier '=' 'getEntityByID' '('
                          entityListIdentifier ',' entityIdName ')' ';' ;
getEntityByLocation     : entityType? entityIdentifier '=' 'getEntityByLocation' '('
                          entityListIdentifier ',' locationIdName ')' ';' ;
conditionExpression     : relationalCondition (( '&&' relationalCondition) | ('||' relationalCondition))* ;
relationalCondition     : expression ( '==' expression | '!=' expression | '<' expression | '>'
                          expression | '<=' expression | '>=' expression)* ;
expression              : atom (('+' atom) | ('-' atom) | ('*' atom) | ('/' atom))* ;
atom                    : ('this.')? variableIdentifier | STRING_LITERAL | getParameterValue |
                          getEntityLocation | IntegerNumber | DoubleNumber |
                          'true' | 'false' | ('(' expression ')') ;
```

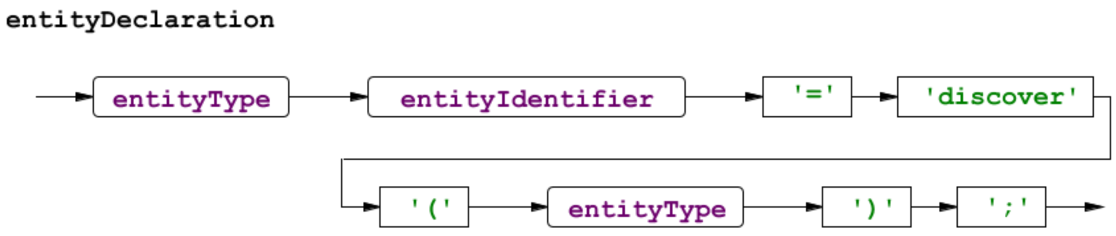
```
getParameterValue       : 'this.'? variableIdentifier '.' 'get' '(' parameterKeyName')' ;
getEntityLocation       : entityIdentifier '.' 'getLocation' '(' ')' '.' 'get' '('
                          'locationID' ')' ';' ;
IF                      : 'if' | 'IF' ;
mapPair                 : '(' parameterKeyName',' (DoubleNumber | IntegerNumber |
                          STRING_LITERAL | variableIdentifier | 'true' | 'false') ')' ;
duration                : (IntegerNumber 'Y' '-')? (IntegerNumber 'M' '-')?
                          (IntegerNumber 'D' '-')?( IntegerNumber 'H' '-')?
                          (IntegerNumber 'M' '-')?  IntegerNumber 'S' ;
dataType                : 'int' | 'boolean' | 'String' | 'long' | 'float' | 'double';
eventType               : ID ;
entityType              : ID ;
entityIdentifier        : ID ;
entityListIdentifier    : ID ;
processIdentifier       : ID ;
variableIdentifier      : ID ;
locationIdName          : STRING_LITERAL;
serviceName             : STRING_LITERAL;
entityIdName            : STRING_LITERAL;
parameterKeyName        : STRING_LITERAL;
ID                      : ('a'..'z' | 'A'..'Z' | '_')('a'..'z' | 'A'..'Z' | '0'..'9' | '_')* ;
DoubleNumber            : '0'..'9'+ '.' '0'..'9'+ ;
IntegerNumber           : ('0'..'9')+ ;
STRING_LITERAL          : '"' (~('"' | '\\' | '\n' | '\r' ))* '"' ;
```

D GPL4SRE: Syntax Diagram

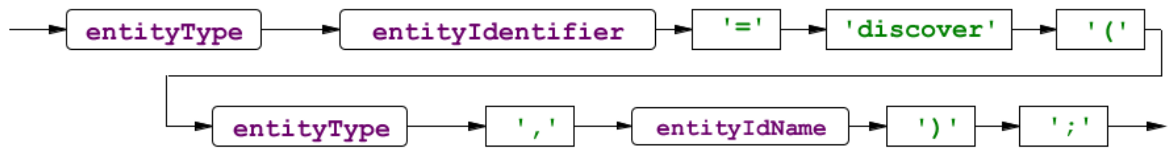




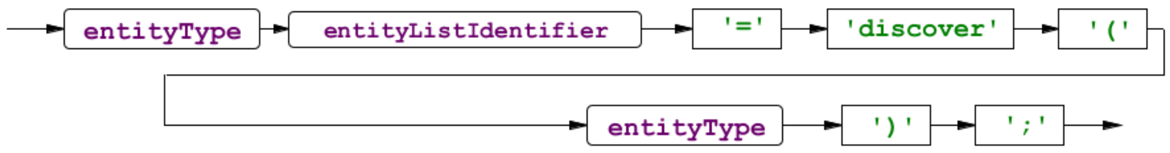




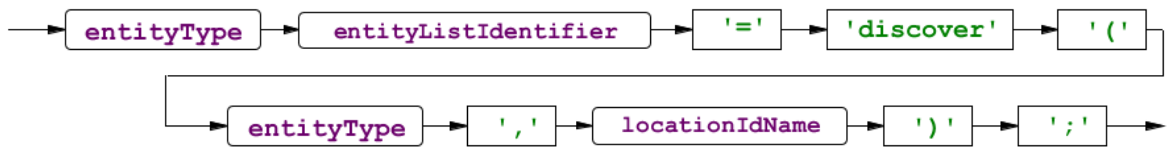
entityByIdDeclaration



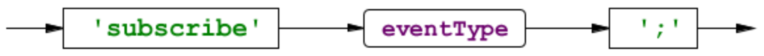
entityListDeclaration



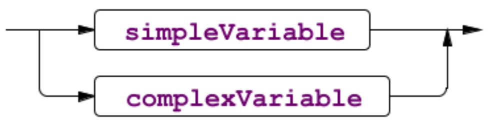
entityListByLocationIdDeclaration



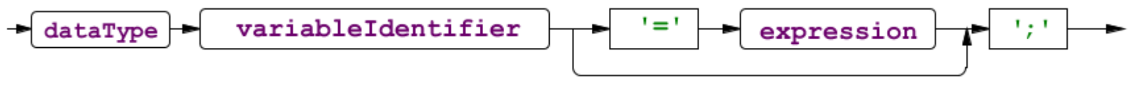
eventRegistration



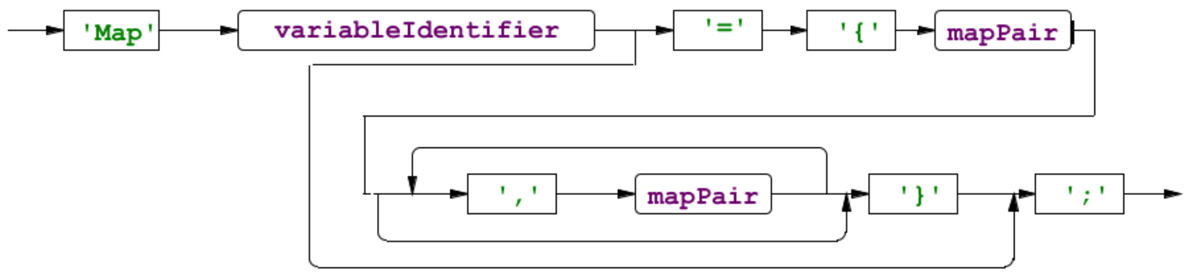
variableDeclaration



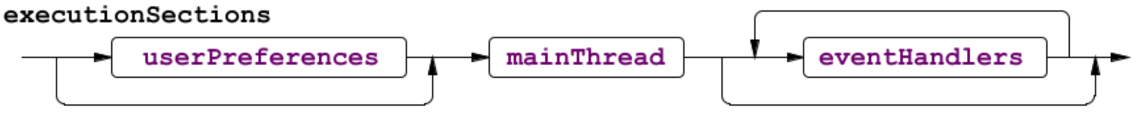
simpleVariable



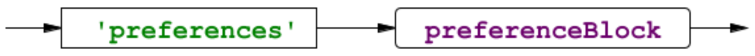
complexVariable



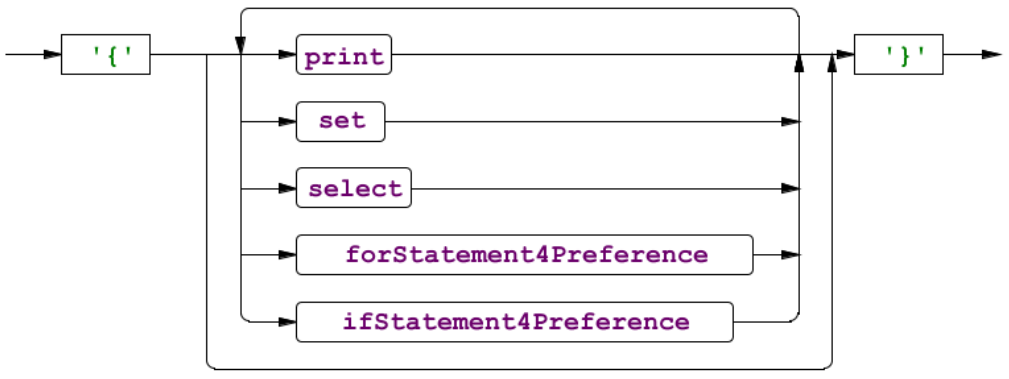
executionSections

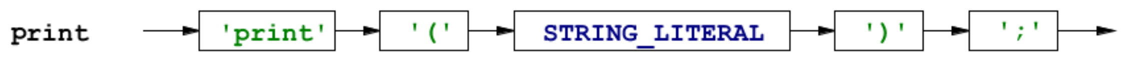


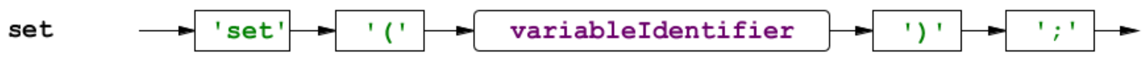
userPreferences

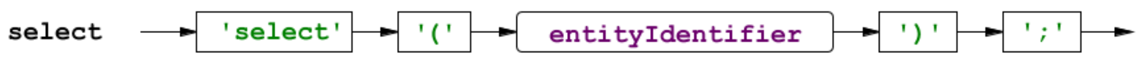


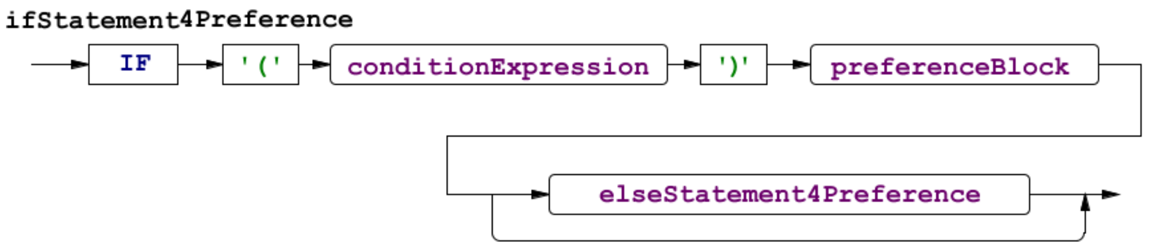
preferenceBlock

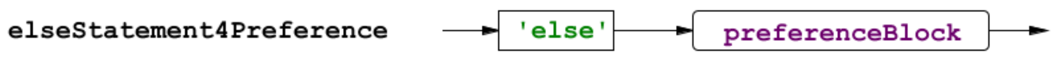


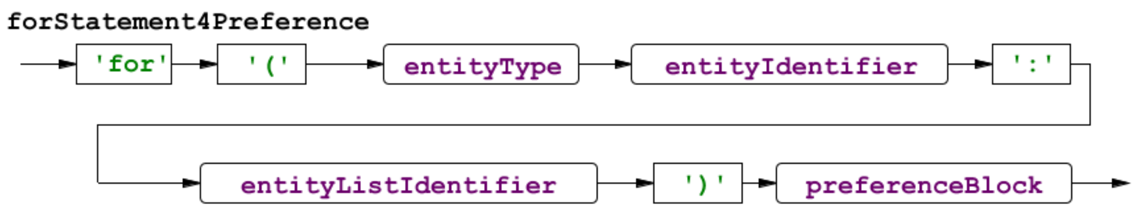

print
'print'
'('
STRING_LITERAL
')'
';'

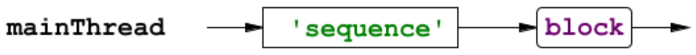

set
'set'
'('
variableIdentifier
')'
';'

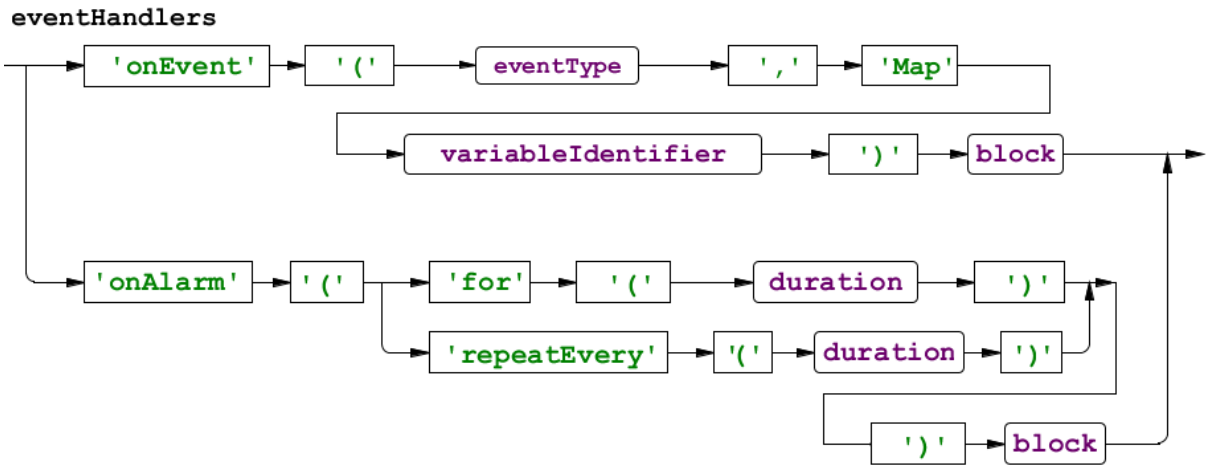

select
'select'
'('
entityIdentifier
')'
';'

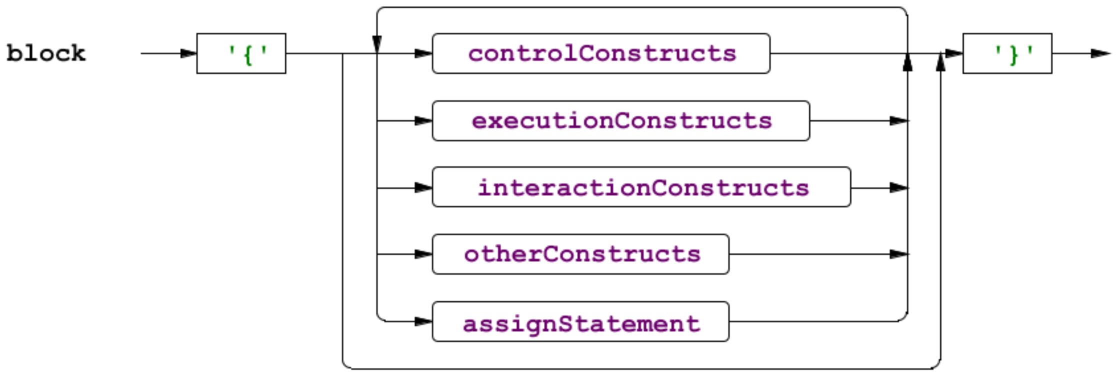

ifStatement4Preference
IF
'('
conditionExpression
')'
preferenceBlock
elseStatement4Preference

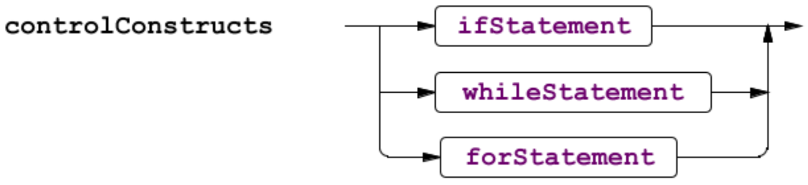

elseStatement4Preference
'else'
preferenceBlock


forStatement4Preference
'for'
'('
entityType
entityIdentifier
':'
entityListIdentifier
')'
preferenceBlock

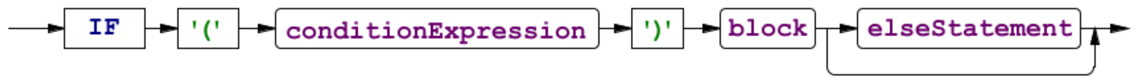

mainThread
'sequence'
block


eventHandlers
'onEvent'
'('
eventType
','
'Map'
variableIdentifier
')'
block
'onAlarm'
'('
'for'
'('
duration
')'
'repeatEvery'
'('
duration
')'
')'
block

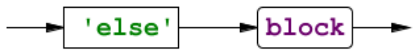

block
'{'
controlConstructs
executionConstructs
interactionConstructs
otherConstructs
assignStatement
'}'


controlConstructs
ifStatement
whileStatement
forStatement

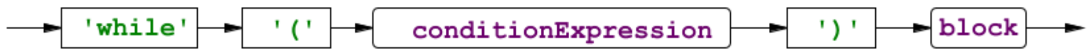
ifStatement



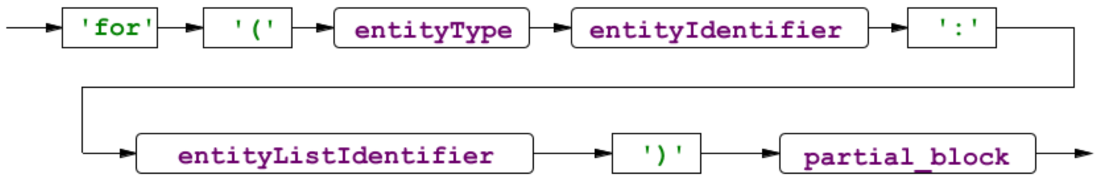
elseStatement



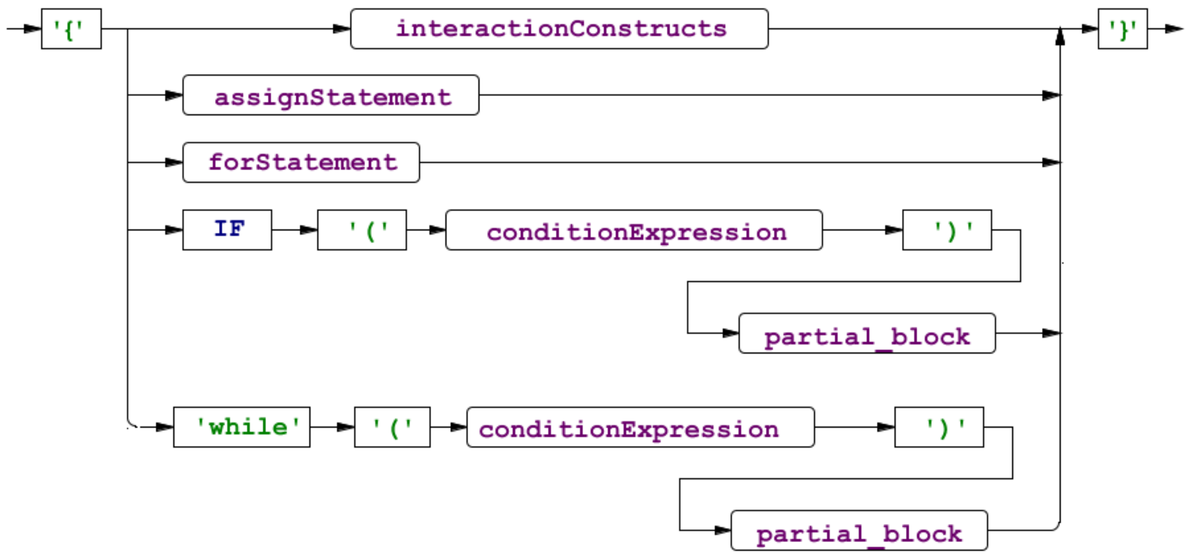
whileStatement



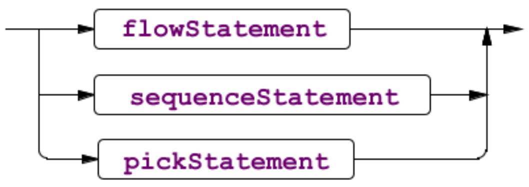
forStatement



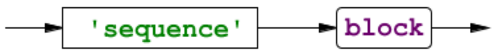
partial_block



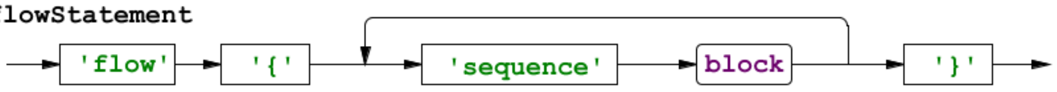
executionConstructs



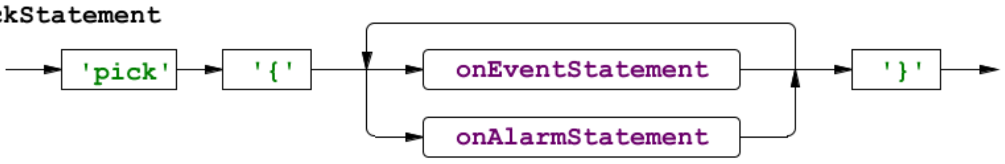
sequenceStatement



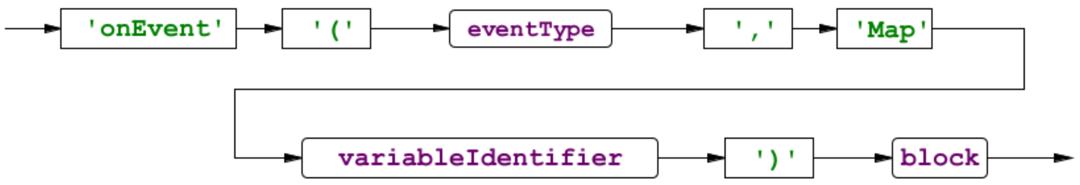
flowStatement



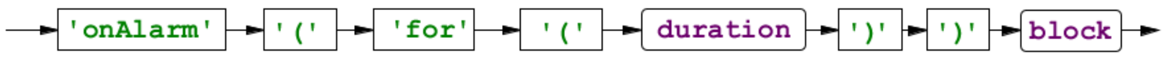
pickStatement



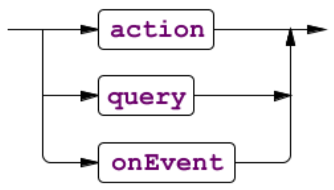
onEventStatement



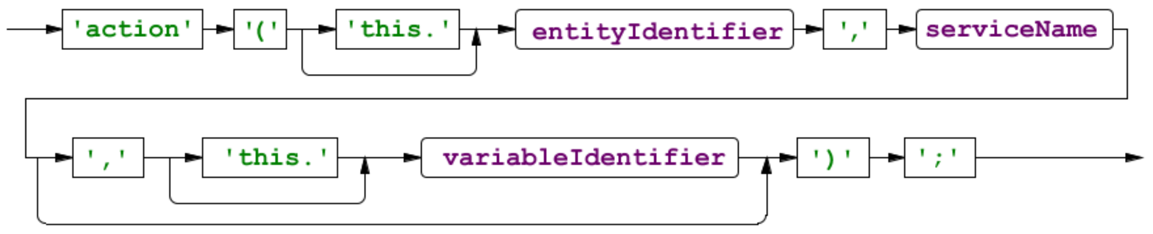
onAlarmStatement



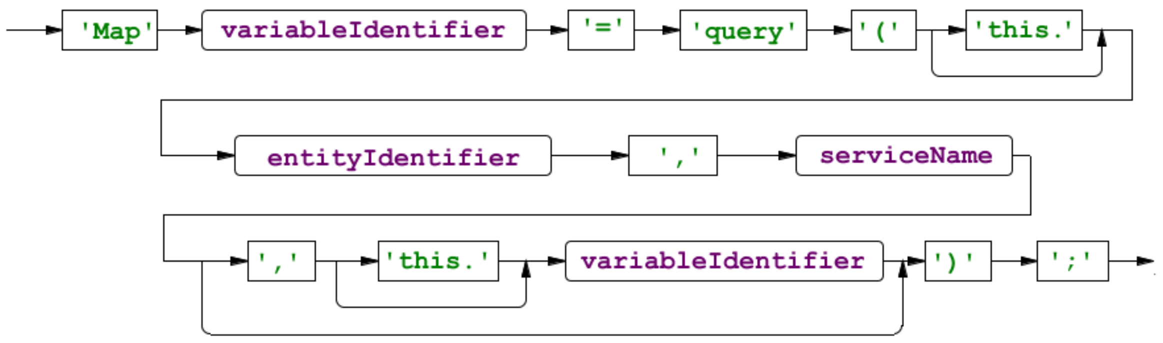
interactionConstructs



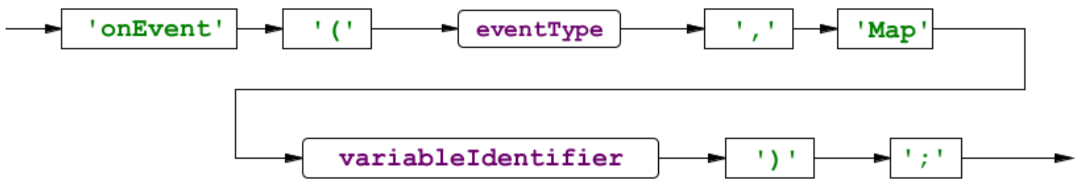
action



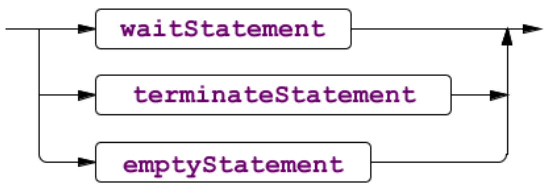
query



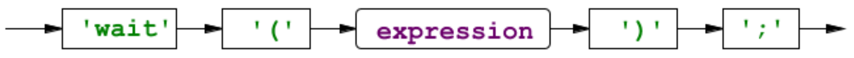
onEvent



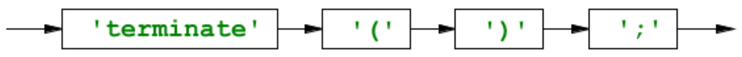
otherConstructs



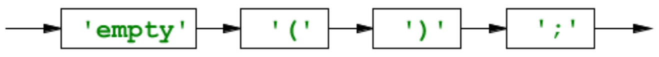
waitStatement



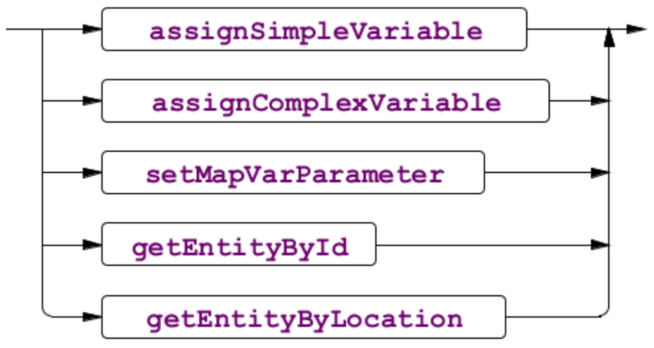
terminateStatement



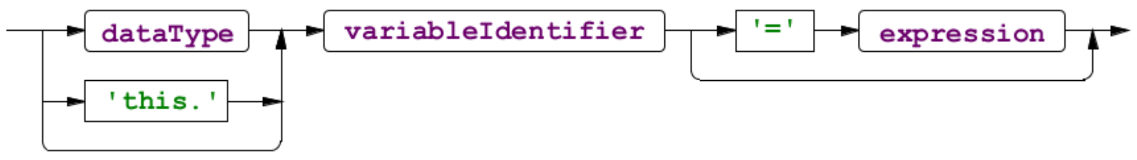
emptyStatement



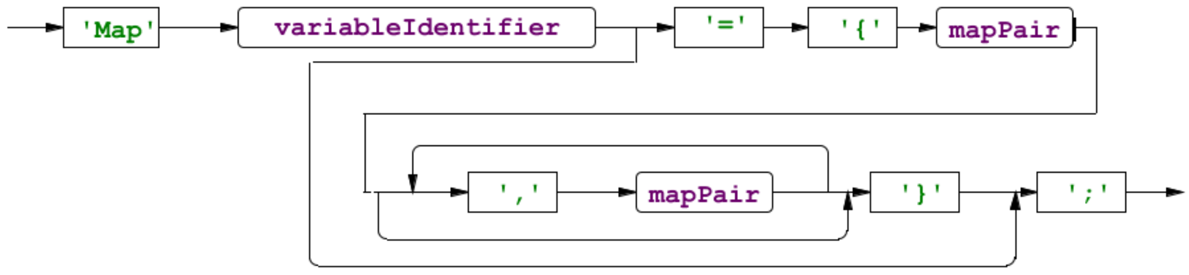
assignStatement



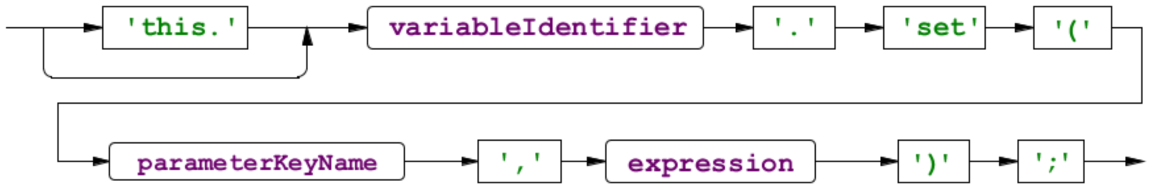
assignSimpleVariable



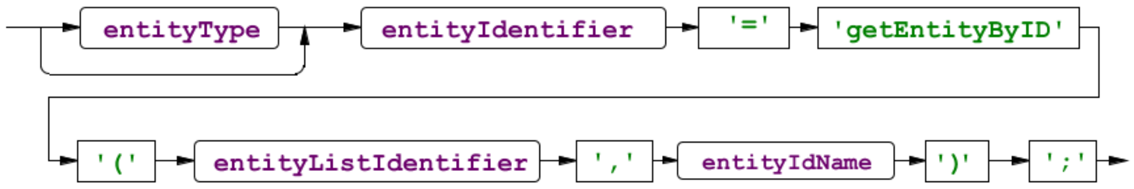
assignComplexVariable



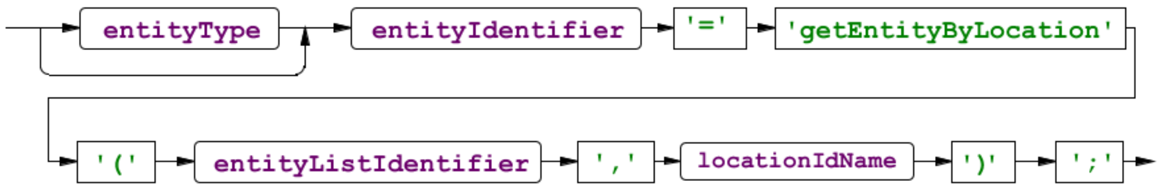
setMapVarParameter



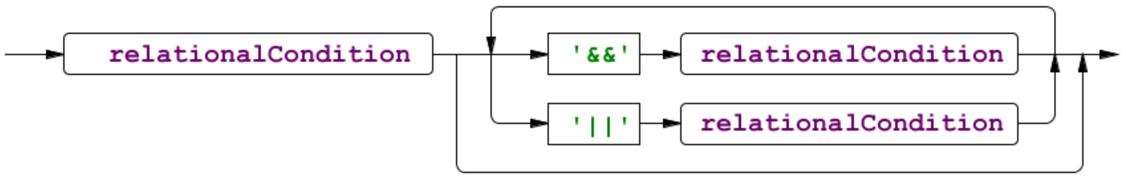
getEntityById



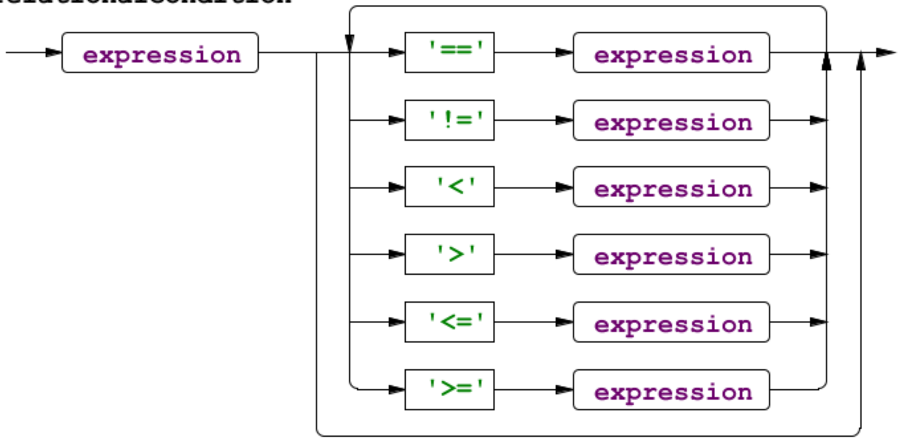
getEntityByLocation



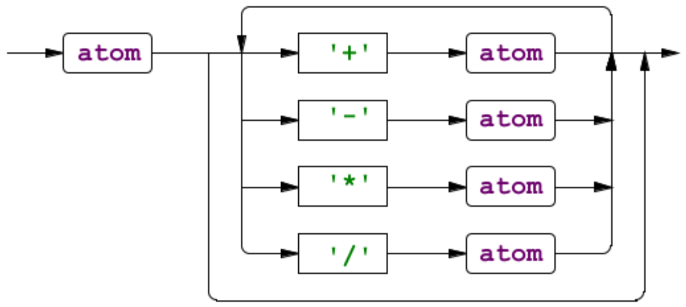
conditionExpression



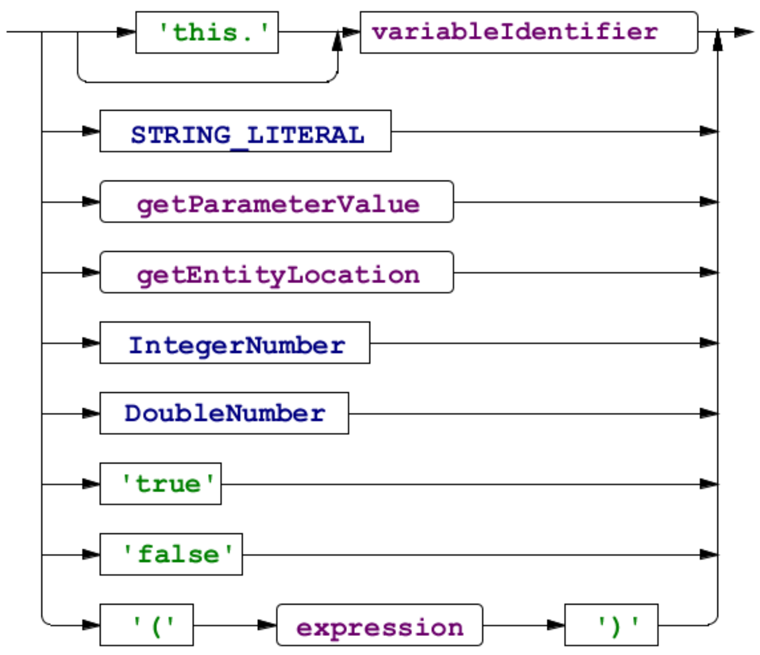
relationalCondition



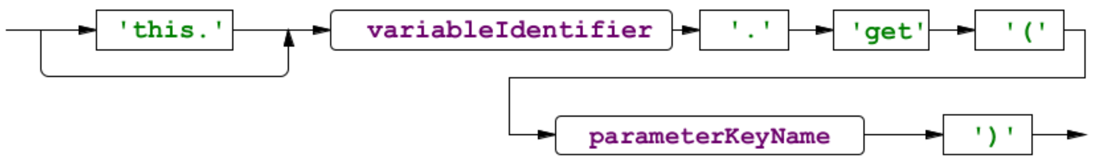
expression



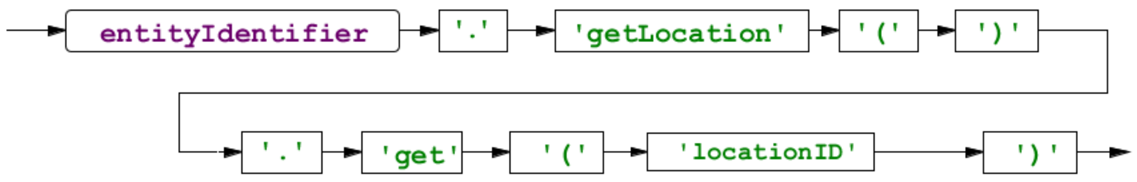
atom



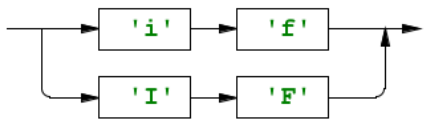
getParameterValue

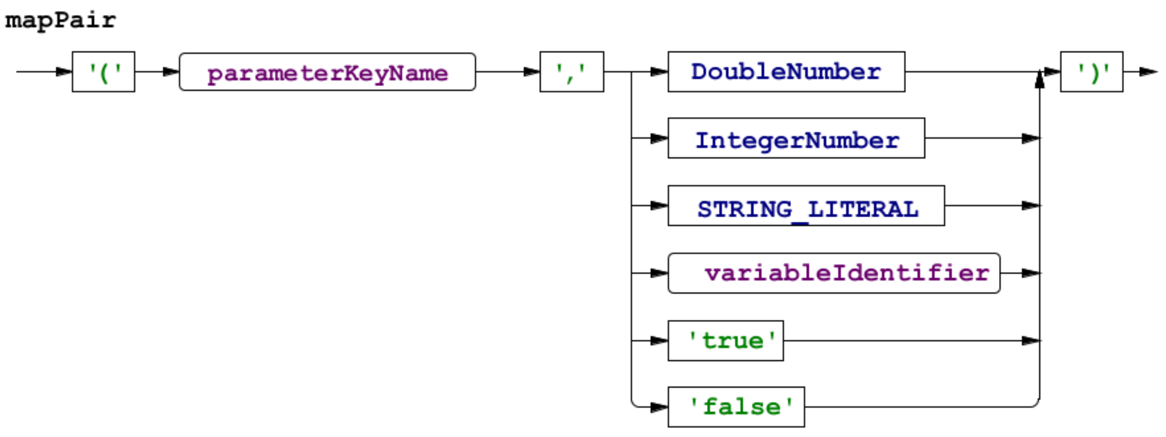


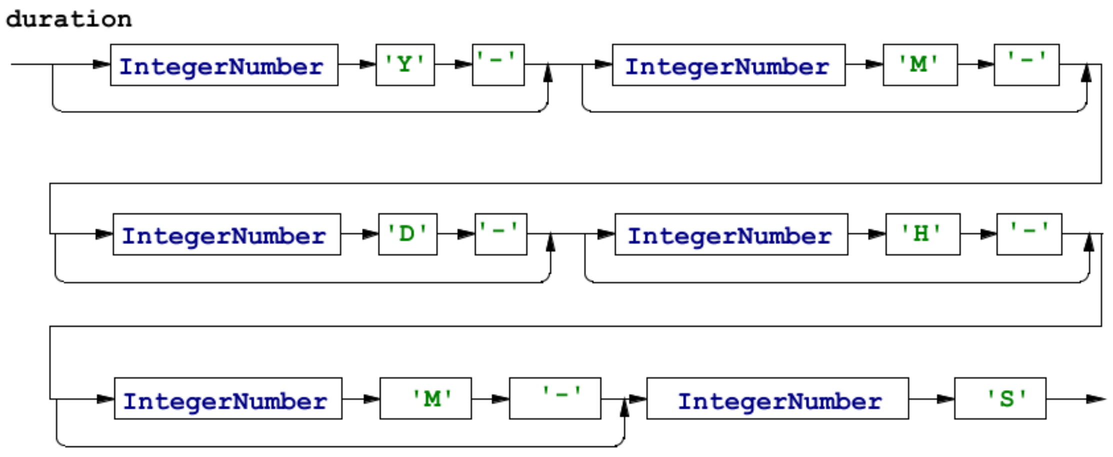
getEntityLocation

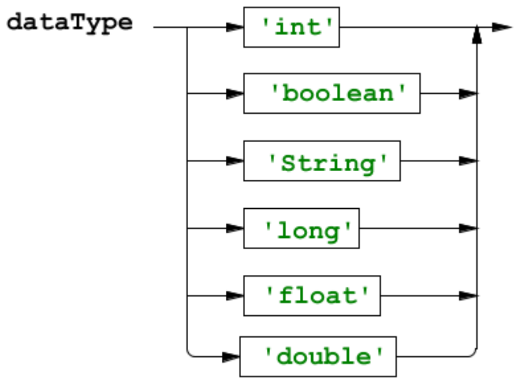


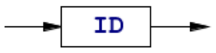
IF

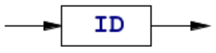


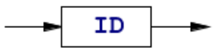

mapPair
'('
parameterKeyName
','
DoubleNumber
IntegerNumber
STRING_LITERAL
variableIdentifier
'true'
'false'
')'

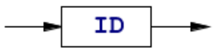

duration
IntegerNumber
'Y'
'-'
IntegerNumber
'M'
'-'
IntegerNumber
'D'
'-'
IntegerNumber
'H'
'-'
IntegerNumber
'M'
'-'
IntegerNumber
'S'

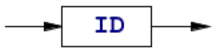

dataType
'int'
'boolean'
'String'
'long'
'float'
'double'

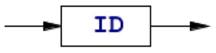
eventType


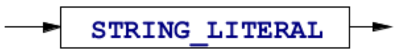
entityType


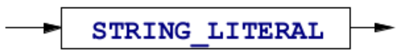
entityIdentifier


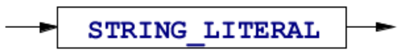
entityListIdentifier


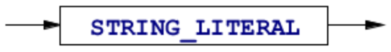
processIdentifier


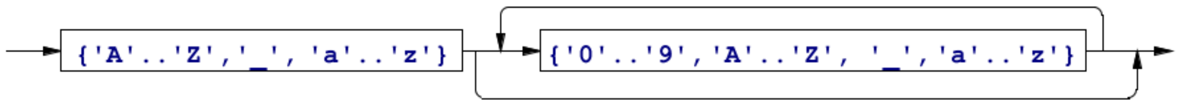
variableIdentifier


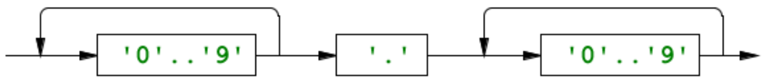
locationIdName


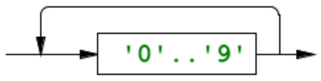
serviceName


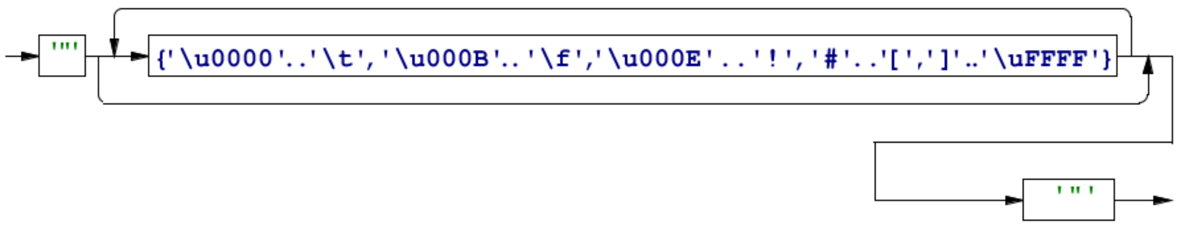
entityIdName


parameterKeyName


ID


DoubleNumber


IntegerNumber


STRING_LITERAL

'"'
{'\u0000'..'\t','\u000B'..'\f','\u000E'..'!','#'..'[',']'..'\uFFFF'}
'"'

References

[1] Weiser, M. *The Computer for the 21st Century*. Scientific American. 1991.

[2] Diane Cook, Sajal Das. *SMART ENVIRONMENTS Technology, Protocols, and Applications*. WILEY INTERSCIENCE. 2005.

[3] Alliance. ZigBee from, http://www.zigbee.org (accessed 22.05.2015).

[4] Jonathan Hui, David Culler. IP is dead, long live IP for wireless sensor networks. *In Proceedings of 6th ACM conference on Embedded network sensor systems*, Raleigh, NC, USA, ACM, p. 15-28, 2008.

[5] Thanos Stavropoulos, Konstantinos Gottis, Dimitris Vrakas and Ioannis Vlahavas. aWESoME: A web service middleware for ambient intelligence. *Expert Systems with Applications*, Tarrytown, NY, USA, Pergamon Press, p. 4380-4392 2013.

[6] Elmar Zeeb, Guido Moritz, Dirk Timmermann, Frank Golatowski. WS4D: Toolkits for Networked Embedded Systems Based on the Devices Profile for Web Services. *In Proceedings of 39th International Conference on Parallel Processing Workshops (ICPPW)*, San Diego, CA IEEE, p. 1-8, 2010.

[7] DPWS. *Devices Profile for Web Services Version 1.1*. 2009.

[8] Christin Groba, Siobhan Clarke. Web services on embedded systems-a performance study. *In Proceedings of 8th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, Mannheim, IEEE, p. 726-731, 2010.

[9] Petros Belimpasakis, Seamus Moloney A Platform for Proving Family Oriented RESTful Services Hosted at Home *IEEE Transactions on Consumer Electronics*, 55(2):690-698, 2009.

[10] Mauro Caporuscio, Marco Funaro, Carlo Ghezzi, Valerie Issarny. ubiREST: A RESTful Service-oriented Middleware for Ubiquitous Networking. *Advanced Web Services* Springer, p. 475-500, 2014.

[11] Daniel Romero, Gabriel Hermosillo, Amirhosein Taherkordi, Russel Nzekwa, Romain Rouvoy, Frank Eliassen RESTful Integration of Heterogeneous Devices in Pervasive Environments. *Distributed Applications and Interoperable Systems*, Springer Berlin Heidelberg, 6115:1-14, 2010.

[12] Alliance, OSGi. Open Services Gateway initiative (OSGi). from, http://www.osgi.org (accessed 22.05.2015).

[13] Sheng-Tzong Cheng, Chi-Hsuan Wang, Gwo-Jiun Horng. OSGi-based smart home architecture for heterogeneous network. *Expert Systems with Applications*, 39(16):12418–12429, 2012.

[14] Son N. Han, Gyu Myoung Lee, Noel Crespi. Semantic Context-Aware Service Composition for Building Automation System. *IEEE Transactions on Industrial Informatics*, 10(1):752-761, 2014.

[15] Tiziana Catarci, Claudio Di Ciccio, Vincenzo Forte, Ettore Iacomussi, Massimo Mecella, Giuseppe Santucci, Giuseppe Tino Service Composition and Advanced User Interfaces in the Home of Tomorrow: The SM4All Approach. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Ambient Media and Systems*, Springer Link, 70:12-19, 2011.

[16] Maria J. Santofimiaa, Scott E. Fahlmanb, Xavier del Toroc, Francisco Moyaa, Juan C. Lopeza. A semantic model for actions and events in ambient intelligence. *Engineering Applications of Artificial Intelligence*, 24(8):1432–1445, 2011.

[17] Qingquan Sun, Weihong Yu, Nikolai Kochurov, Qi Hao, Fei Hu. A Multi-Agent-Based Intelligent Sensor and Actuator Network Design for Smart House and Home Automation. *Journal of Sensor and Actuator Networks*, 2:557-588, 2013.

[18] OASIS. Web Services Business Process Execution Language. from, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (accessed 22.05.2015).

[19] Poslad, Stefan. *Ubiquitous Computing: Smart Devices, Environments and Interactions*. John Wiley & Sons. 2009.

[20] Marcelo Kallmann, Daniel Thalmann. Modeling Objects for Interaction Tasks. *In Proceedings of 9th Eurographics Workshop on Animation and Simulation (EGCAS)*, Lisbon, p. 73-86, 1998.

[21] R. Giffinger, C. Fertner, H. Kramar, R. Kalasek, N. Pichler.Milanovic, E. Meijers. *Smart cities-Ranking of European medium-sized cities* Centre of Regional Science (SRF), Vienna University of Technology. 2007. from, http://www.smart-cities.eu/download/smart_cities_final_report.pdf.

[22] Luis M., Klaus Wünstel. *Smart Cities Applications and Requirements*. Networks Eurpean Technology Platform. 2011. from, http://www.networks-etp.eu/fileadmin/user_upload/Publications/Position_White_Papers/White_Paper_Smart_Cities_Applications.pdf.

[23] Enocean-Alliance. Building Automation. from, www.enocean-alliance.org (accessed 22.05.2015).

[24] alliance, IPSO. Enabling the internet of things. from, http://www.ipso-alliance.org/ (accessed 22.05.2015).

[25] Group, Bluetooth. Bluetooth Core Specification. from, https://www.bluetooth.org/en-us/specification/adopted-specifications (accessed 22.05.2015).

[26] HomePlug. *HomePlug 1.0.1 Specification*. 2001.

[27] X10. X10 Basics. from, http://www.x10.com/x10-basics.html (accessed 20.05.2015).

[28] Samaneh Movassaghi, Mehran Abolhasan, Justin Lipman, David Smith, Abbas Jamalipour. Wireless Body Area Networks: A Survey. *Communications Surveys & Tutorials*, 6(3):1658-1686 2014.

[29] Association, IEEE Standards. IEEE Standard for Local and metropolitan area networks—Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). *Jounral Name*, New York, IEEE Publisher, p. 1-314, 2011.

[30] Sang Hyun Park , So Hee Won , Jong Bong Lee , Sung Woo Kim. Smart home–digitally engineered domestic life. *Personal and Ubiquitous Computing*, 7(3):189-196, 2003.

[31] Albreshne Abdaladhem, Ayoub Ait Lahcen, Jacques Pasquier. A Framework and its Associated Process-Oriented Domain Specific Language for Managing Smart Residential Environments. *International Journal of Smart Home*, 7(377-392, 2013.

[32] chander, Vinob. Novel Ubiquitous Interoperable Context-aware Smart Environments through Web Services. *In Proceedings of the International MutliConference of Engineers and Computer Scientists*, Hong Kong, 1:646-650, 2012.

[33] Nenad Stojanovic, Dejan Milenovic, Yongchun Xu. An Intelligent Event-driven Approach for Efficient Energy Consumption in Commercial Buildings. *In Proceedings of 5th ACM international conference on Distributed event-based system*, New York, NY, USA, ACM, p. 303-312, 2011.

[34] Viktoriya Degeler, Alexander Lazovik. Architecture pattern for context-aware smart environments *Creating Personal, Social and Urban Awareness through Pervasive Computing*, IGI Global, p. 108-130, 2013.

[35] Diane Cook, Sajal Das. Designing and Modeling Smart Environments. *In Proceedings of International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06)*, IEEE, p. 490-494, 2006.

[36] Association, KNX. *KNX System Specications*. 2011.

[37] BACnet. ASHARE BACnet. from, http://www.bacnet.org/ (accessed 20.05.2015).

[38] Viktoriya Degeler, Luis Gonzalez, Mariano Leva, Paul Shrubsole, Silvia Bonomi, Oliver Amft, Alexander Lazovik. Service-Oriented Architecture for Smart Environments. *In Proceedings of IEEE International Conference on Service Oriented Computing and Applications (SOCA)*, p. 99-104, 2013.

[39] Rodolfo Santos, Paulo Carreira. Service Oriented Development of Building Energy Management Systems. *In Proceedings of 3rd International Conference on Smart Grids and Green IT Systems (SMARTGREENS)*, Barcelona, Spain, p. 315-323, 2014.

[40] Erl, Thomas. *Service-Oriented Architecture, Concepts, Technology, and Design*. Prentice Hall Indiana. 2006.

[41] Antonio Pintus, Davide Carboni, Andrea Piras, Alessandro Giordano. Connecting smart things through web services orchestrations. *In Proceedings of 10th international conference on Current trends in web engineering (ICWE'10)*, Springer-Verlag Berlin, p. 431-441, 2010.

[42] Miguel S. Familiar, José F. Martínez, Lourdes López. Pervasive Smart Spaces and Environments: A Service-Oriented Middleware Architecture for Wireless Ad Hoc and Sensor Networks. *International Journal of Distributed Sensor Networks*, 2012:1-11, 2012.

[43] Michael P., Papazoglou. *Web Services: Principles and Technology*. PEARSON Prentice Hall. 2008.

[44] Thinagaran Perumal, Abd Rahman Ramli, Chui Yew Leong. SOA-Based Framework for Home and Building Automation Systems (HBAS). *International Journal of Smart Home*, 8(5):197-206, 2014.

[45] Cerami, Ethan. *Web Services Essentials: Distributed Applications with XML-RPC, SOAP, UDDI & WSDL.* O'Reilly Media. 2002.

[46] Leonard Richardson, Sam Ruby. *RESTful Web Services* 1ed. O'Reilly Media. 2007.

[47] W3C. Web Services Architecture. from, http://www.w3.org/TR/ws-arch/ (accessed 20.06.2015).

[48] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*, PhD Thesis, Department of Informatics, University of California, USA, 2000.

[49] W3C. HTTP Specifications and Drafts. from, http://www.w3.org/Protocols/Specs.html (accessed 20.05.2015).

[50] OMG. Common Object Request Broker Architecture (CORBA) Specification, Version 3.3. from, http://www.corba.org/ (accessed 22.05.2015).

[51] Group, Network Working. RPC: Remote Procedure Call Protocol Specification Version 2. from, (accessed 22.05.2015).

[52] Guinard, Dominique. *A Web of Things Application Architecture -Integrating the Real-World into the Web*. PhD thesis, Department of Informatics, ETH Zurich, Switzerland, 2011. Thesis No. 19891.

[53] Cesare Pautasso, Olaf Zimmermann, Frank Leymann Restful web services vs. "big"' web services: making the right architectural decision. *In Proceedings of 17th international conference on World Wide Web* ACM, p. 805-814, 2008.

[54] Ovidiu Vermesan, Peter Friess. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. River Publisher. 2013.

[55] Luigi Atzori, Antonio Iera, Giacomo Morabito. The Internet of Things: A survey. *Computer Networks Journal*, 54:2787-2805, 2010.

[56] Deze Zeng, Song Guo, Zixue Cheng. The Web of Things: A Survey. *Journal of Communications*, 6(6):424-438, 2011.

[57] Alliance, OSGi. *About the OSGi Service Platform*. 2007.

[58] Chao-Lin Wu, Chun-Feng Liao , Li-Chen Fu. Service-Oriented Smart-Home Architecture Based on OSGi and Mobile-Agent Technology. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(2):193-205, 2007.

[59] Davide Carneiro, Paulo Novais, Ricardo Costa, José Neves. Developing Intelligent Environments with OSGi and JADE *In Proceedings of 3rd International Conference on Artificial Intelligence (IFIPAI2010)*, Brisbane, Australia, Springer Berlin Heidelberg, p. 174-183, 2010.

[60] Ji Eun Kim, George Boulos, John Yackovich, Tassilo Barth, Christian Beckel, Daniel Mosse. Seamless Integration of Heterogeneous Devices and Access Control in Smart Homes *In Proceedings of 8th International Conference on Intelligent Environments*, Guanajuato IEEE, p. 206-213, 2012.

[61] W3C. Web Services Description Language. from, http://www.w3.org/TR/wsdl (accessed 20.05.2015).

[62] W3C. SOAP Simple Object Access Protocol. from, http://www.w3.org/ TR/2007/REC-soap12-part2-20070427 (accessed 20.05.2015).

[63] Daniel Schall, Marco Aiello, Schahram Dustdar. Web Services on Embedded Devices. *International Journal of Web Information Systems*, 2(1):45-50, 2006.

[64] Machado, G.B., Siqueira, F., Mittmann R., Augusto C., e Vieira V. Integration of Embedded Devices Through Web Services: Requirements, Challenges and Early Results *In Proceedings of 11th IEEE Symposium on Computers and Communications (ISCC '06)* IEEE, p. 1530-1346, 2006.

[65] Yeon-Seok Kim, Kyong-Ho Lee. A Light-weight Framework for Hosting Web Services on Mobile Devices *In Proceedings of 5th European Conference on Web Services (ECOWS '07)* Halle, Germany p. 255-263, 2007.

[66] Engelen, Robert van. Code generation techniques for developing light-weight XML Web services for embedded devices. *In Proceedings of ACM symposium on Applied computing (SAC'04)*, p. 854-861, 2004.

[67] Nissanka B. Priyantha, Aman Kansal, Michel Goraczko, Feng Zhao. Tiny web services: design and implementation of interoperable and evolvable sensor networks. *In Proceedings of the 6th ACM conference on Embedded network sensor systems* ACM p. 253-266 2008.

[68] Marvell. Marvell Smart Energy Platform Solutions. from, http://www.marvell.com/smart-energy/assets/Marvell-Smart-Energy-Platform-Brief.pdf (accessed 22.05.2015).

[69] University of Rostock, University of Dortmund and MATERNA. Web Services for Devices (WS4D). from, http://ws4d.e-technik.uni-rostock.de/ (accessed 20.05.2015).

[70] W3C. OWL-S: Semantic Markup for Web Services from, http://www.w3.org/Submission/OWL-S/ (accessed 22.05.2015).

[71] W3C. Semantic Annotations for WSDL and XML Schema. from, http://www.w3.org/TR/sawsdl/ (accessed 20.05.2015).

[72] W3C. Web Service Modeling Ontology (WSMO). from, http://www.w3.org/Submission/WSMO/ (accessed 20.05.2015).

[73] Kopecky, J., Innsbruck , Vitvar, T., Bournez, C., Farrell, J. . SAWSDL: Semantic Annotations for WSDL and XML Schema *Internet Computing, IEEE*, 11(6):60-67, 2007.

[74] Agency, Defense Advanced Research Projects. The DARPA Agent Markup Language. from, http://www.daml.org/ (accessed 22.05.2015).

[75] Apache. DJ-JiniTM Discovery & Join Specification v2.1.2. from, https://river.apache.org/doc/specs/html/discovery-spec.html (accessed 22.05.2015).

[76] OASIS. UDDI Specification Version 3.0.2. from, http://uddi.org/pubs/uddi-v3.0.2-20041019.htm (accessed 20.05.2015).

[77] Feng Wang, Kenneth J. Turner. An Ontology-based Actuator Discovery and Invocation Framework in Home Care Systems. *In Proceedings of 7th International Conference on Smart Homes and Health Telematics*, Berlin, Springer, p. 66-73, 2009.

[78] Hen-I Yang, Ryan Babbitt, Johnny Wong, Carl K. Chang *A Framework for Service Morphing and Heterogeneous Service Discovery in Smart Environments.* Springer Berlin Heidelberg. p. 9-17, 2012.

[79] microsystems, Sun. Service Location Protocol Administration Guide. from, http://docs.oracle.com/cd/E19455-01/806-1412/806-1412.pdf (accessed 20.05.2015).

[80] ISO. *UPnP Device Architecture*. Standard:SO/IEC 29341-1:2011(E) 2008.

[81] Zhi Wang , Jizhong Zhao. Improved algorithm for UPnP discovery in smart space. *In Proceedings of IEEE 3rd International Conference on Software Engineering and Service Science (ICSESS)*, IEEE, p. 519-522, 2012.

[82] Elira Hoxha, Dhimitri Tole, Kreshnik Vukatana. Semantics and OWL in UDDI Registry: Improving the Discovery Process of Web Services. *Academic Journal of Interdisciplinary Studies*, 3(1):341-346, 2014.

[83] Falak Nawaz, Kamran Qadir, H. Farooq Ahmad. SEMREG-Pro: A Semantic based Registry for Proactive Web Service Discovery using Publish-Subscribe Model. *In Proceedings of 4th International Conference on Semantics, Knowledge and Grid* Beijing, IEEE, p. 301-308, 2008.

[84] He Yu'an, Wu Dongqi, Yu Tao. Research on Service Discovery and Matching based on Ontology and Service Capabilities in Manufacturing Grid. *World Congress on Computer Science and Information Engineering*, OEEE Computer society, p., 2009.

[85] Yue-an Zhu, Xiao-hua Meng A Framework for Service Discovery in Pervasive Computing *In Proceedings of 2nd International Conference on Information Engineering and Computer Science (ICIECS 2010)*, p. 1-4, 2010.

[86] Manshan Lin, Heqing Guo, Jianfei Yin. Goal Description Language for Semantic Web Service Automatic Composition. *Proceedings of the The 2005 IEEE, Symposium on Applications and the Internet*, Washington, USA p. 190-196, 2005.

[87] Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins *PDDL-The Planning Domain Denition Language Version 1.2*. Yale Center for Computational Vision and Control. 1998. from,

[88] Tsung-Hsien Yang, Wei-Po Lee. A Service-Oriented Framework for the Development of Home Robots *International Journal of Advanced Robotic Systems*, 10:1-11, 2013.

[89] OMG. Business Process Model and Notation (BPMN) 2.0. from, http://www.omg.org/spec/BPMN/2.0/ (accessed 22.05.2015).

[90] Michael Rietzler, Julia Greim, Marcel Walch, Florian Schaub, Björn Wiedersheim, Michael Weber. homeBLOX: introducing process-driven home automation. *In Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication: Workshop on Design, Technology, Systems and Applications for the Home*, ACM, p. 801-808, 2013.

[91] ORACLE. Oracle SOA Suite. from, http://www.oracle.com/us/products/middleware/soa/suite/overview/index.html (accessed 20.05.2015).

[92] Apache, ODE. WS-BPEL 2.0. from, http://ode.apache.org/ws-bpel-20.html (accessed 20.05.2015).

[93] JBOSS. Business Process Management (BPM) Suite. from, http://www.jbpm.org/ (accessed 22.05.2015).

[94] W3C. *Web Services Choreography Description Language Version 1.0*. 2005.

[95] Diane J. Cook, Michael Youngblood, Sajal K. Das. A Multi-agent Approach to Controlling a Smart Environment. *Journal of Ambient Intelligence and Smart Environments*, 1(1):51-55, 2009.

[96] Giuseppe Loseto, Floriano Scioscia, Michele Ruta, Eugenio Di Sciascio. Semantic-based smart homes: a multi-agent approach. *In Proceedings of 13th Workshop on Objects and Agents*, 892:49-55, 2012.

[97] W3C. OWL 2 Web Ontology Language. from, http://www.w3.org/TR/2012/REC-owl2-syntax-20121211 (accessed 20.05.2015).

[98] W3C. Resource Description Framework (RDF). from, http://www.w3.org/RDF/ (accessed 22.05.2015).

[99] W3C. SPARQL Query Language for RDF. from, http://www.w3.org/TR/rdf-sparql-query/ (accessed 20.05.2015).

[100] Research, Stanford Center for Biomedical Informatics. Protégé Tool. from, http://protege.stanford.edu/ (accessed 20.05.2015).

[101] Abdaladhem Albreshne, Ayoub Ait Lahcen, Jacques Pasquier. Using a Residential Environment Domain Ontology for Discovering and Integrating Smart Objects in Complex Scenarios. *In Proceedings of the International Workshop on Enabling ICT for Smart Buildings (ICT-SB 2014)*, Hasselt, Belgium, Elsevier, 32:997-1002, 2014.

[102] Abdaladhem Albreshne , Jacques Pasquier. A Template-Based Semi-Automatic Web Services Composition Framework: Case Study of Smart Home Management. *In Proceedings of the PhD Symposium at the 9th IEEE European Conference on Web Services ECOWS11*, Lugano, Switzerland, p. 11-17, 2011.

[103] Abdaladhem Albreshne , Jacques Pasquier. Semantic-Based Semi-Automatic Web Service Composition. *In Proceedings of the 5th Libyan Arab International Conference On Electrical and Electronic Engineering LAICEEE*, Tripoli, Libya, p. 603-615, 2010.

[104] Abdaladhem Albreshne, Jacques Pasquier. A Domain Specific Language for High-Level Process Control Programming in Smart Buildings. *In Proceedings of the 6th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2015)*, Berlin, Germany, Elsevier, 63:65-73, 2015.

[105] Zohar Etzioni, John Keeney, Rob Brennan, David Lewis Supporting Composite Smart Home Services with Semantic Fault Management *In Proceedings of 5th International Conference on Future Information Technology (FutureTech)*, Busan, p. 1-8, 2010.

[106] Adnan Afsar Khan, Hussein T. Mouftah. Web services for indoor energy management in a smart grid environment. *In Proceedings of 22nd International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC)*, IEEE, p. 1036-1040, 2011.

[107] Nils Glombitza, Dennis Pfisterer, Stefan Fischer. Integrating Wireless Sensor Networks into Web Service-Based Business Processes. *In Proceedings of 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*, ACM, p. 25-30, 2009.

[108] Feng Gao, Maciej Zaremba, Sami Bhiri, Wassim Derguerch. Extending BPMN 2.0 with Sensor and Smart Device Business Functions. *In Proceedings of 20th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Paris, France, IEEE, p. 297-302, 2011.

[109] Stefano Tranquillini, Patrik Spieß, Florian Daniel, Stamatis Karnouskos, Fabio Casati, Nina Oertel, Luca Mottola, Felix Jonathan Oppermann, Gian Pietro Picco, Kay Römer, Thiemo Voigt Process-Based Design and Integration of Wireless Sensor Network Applications. *Lecture Notes in Computer Science*, 7481:134-149, 2012.

[110] Jopera. JOpera for Eclipse. from, http://www.jopera.org/ (accessed 20.05.2015).

[111] Thanos G, Stavropoulos . Dimitris Vrakas, Ioannis Vlahavas. A survey of service composition in ambient intelligence environments. *Artificial Intelligence Review, Springer*, 40(3):247-270, 2011.

[112] Fulvio Corno, Muhammad Sanaullah. Design-Time Formal Verication for Smart Environments: An Exploratory Perspective. *Journal of Ambient Intelligence and Humanized Computing , Sprinker*, 5(4):581-599, 2014.

[113] Tommaso Magherini, Alessandro Fantechi, Chris D. Nugent, Enrico Vicario. Using Temporal Logic and Model Checking in Automated Recognition of Human Activities for Ambient-Assisted Living. *IEEE Transactions on Human-Machine Systems,* 43(6):509-521, 2013.

[114] Riccardo De Masellis, Claudio Di Ciccio, Massimo Mecella, Fabio Patrizi. Smart Home Planning Programs. *7th International Conference on Service Systems and Service Management*, Tokyo, p. 1-6, 2010.

[115] Eirini K., Ehsan W., Jaap B.,Alexander L., Marco A. Interoperation, Composition and Simulation of Services at Home. *In Proceedings of 8th International Conference, ICSOC*, San Francisco, CA, USA, p. 167-181, 2010.

[116] Christian Reinisch, Mario J.Kofler, Félix Iglesias, Wolfgang Kastner. ThinkHome Energy Efficiency in Future Smart Homes. *EURASIP Journal on Embedded Systems*, p. 1-18, 2011.

[117] Florian Marquardt, Adelinde Uhrmacher. Evaluating AI planning for service composition in smart environments. *In Proceedings of 7th International Conference on Mobile and Ubiquitous Multimedia*, ACM, p. 44-55, 2008.

[118] Faris Nizamic, Tuan Anh Nguyen, Alexander Lazovik ,Marco Aiello. GreenMind - An Architecture and Realization for Energy Smart Buildings. *In Proceedings of 2nd International Conference on ICT for Sustainability (ICT4S 2014),* Atlantis Press, p. 20-29, 2014.

[119] Davide Cavone, Berardina De Carolis, Stefano Ferilli, Nicole Novielli. Smart Composition of Services in Situation-Aware Home Environments. *In Proceedings of 2nd International Workshop on User Modeling and Adaptation for Daily Routines (UMADR)*, Spain, p. 5-12, 2011.

[120] Manuel García-Herranz, Xavier Alamán, Pablo A. Haya. Easing the Smart Home: A rule-based language and multi-agent structure for end user development in Intelligent Environments. *Journal of Ambient Intelligence and Smart Environments*, 2(4):437-438 2010.

[121] Chui Yew Leong, A.R.Ramli, Thinagaran Perumal. A Rule-Based Framework for Heterogeneous Subsystems Management in Smart Home Environment *IEEE Transactions on Consumer Electronics*, 55(3):1208-1213, 2009.

[122] Jin Xiao, Raouf Boutaba. The Design and Implementation of an Energy-Smart Home in Korea. *Journal of Computing Science and Engineering (JCSE)*, 7(3):204-210, 2014.

[123] Turner, Kenneth. Flexible Management of Smart Homes. *Journal of Ambient Intelligence and Smart Environments, IOS Press,* 3:83-110, 2011.

[124] Alexander Smirnov, Alexey Kashevnik, Nikolay Shilov, Nikolay Teslya. Context-based access control model for smart space. *In Proceedings of 5th International Conference on Cyber Conflict*, IEEE, p. 1-15, 2013.

[125] Pierpaolo Baglietto, Massimo Maresca, Michele Stecca. Smart Object Cooperation through Service Composition. *15th International conference on Intelligence in Next Generation Networks*, Berlin, p. 133-138, 2011.

[126] Claudio Di Ciccio, Massimo Mecella. The homes of tomorrow: service composition and advanced user interfaces. *ICST Transactions on Ambient Systems*, 11(10-12):1-13, 2011.

[127] Christos Goumopoulos, Ioannis Calemis, Achilles Kameas Deployment of adaptive workflows in intelligent environments *In Proceedings of 6th International Conference on Intelligent Environments*, IEEE Computer Society, p. 197-202, 2010.

[128] A. Katasonov , M. Palviainen Towards ontology-driven development of applications for smart environments. *In Proceedings of Pervasive Computing and Communications Workshops (PERCOM Workshops), 8th IEEE International Conference on* Mannheim, p. 696-701, 2010.

[129] Michael Compton, Payam Barnaghi, Luis Bermudez et al. The SSN Ontology of the W3C Semantic Sensor Network Incubator Group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25-32, 2012.

[130] Henrik Dibowski, Klaus Kabitzsch. Ontology-Based Device Descriptions and Device Repository for Building Automation Devices. *EURASIP Journal on Embedded Systems*, 3:1-17, 2011.

[131] Zakwan Jaroucheh, Xiaodong Liu, Sally Smith. An approach to domain-based scalable context management architecture in pervasive environments. *Personal and Ubiquitous Computing*, 16(6):741-755, 2012.

[132] Tao Xu, Bertrand David, René Chalon, Yun Zhou. A context-aware middleware for ambient intelligence. *In Proceedings of the Workshop on Posters and Demos Track*, Lisbon, Portugal, ACM, p. 10:1-10:2, 2011.

[133] Kyungeun Park, Yanggon Kim, Juno Chang. Semantic Reasoning with Contextual Ontologies on Sensor Cloud Environment. *International Journal of Distributed Sensor Networks*, 1-13,

[134] Natalia Díaz Rodríguez, M. P. Cuéllar, Johan Lilius, Miguel Delgado Calvo-Flores. A Survey on Ontologies for Human Behavior Recognition. *ACM Computing Surveys (CSUR)* 46(4):1-33, 2014.

[135] Lorenzo, Sommaruga, Tiziana, Formilli and Nicola, Rizzo. DomoML-an Integrating Devices Framework for Ambient Intelligence Solutions. *In Proceedings of 6th International Workshop on Enhanced Web Service Technologies* Lugano, Switzerland, ACM, p. 9-15, 2011.

[136] Dario Bonino, Fulvio Corno DogOnt - Ontology Modeling for Intelligent Domotic Environments. *In Proceedings of 7th International Semantic Web Conference (ISWC)*, Springer Berlin Heidelberg, p. 790-803, 2008.

[137] Thanos G., Dimitris Vrakas, Danai Vlachava, Nick Bassiliades. BOnSAI: a Smart Building Ontology for Ambient Intelligence. *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics (WIMS '12)*, Craiova, Romania, ACM, p. 1-12, 2012.

[138] YERLY, SAMUEL. *Instanciation d'une "Smart Home" en OWL et recherche des ses propriétés en SPARQL*. Bachelor Work, Software Engineering Group, Department of Informatics, University of Fribourg. 2014.

[139] Yew Kwang Hooi, M. Fadzil Hassan, Azmi M. Shariff A Survey on Ontology Mapping Techniques. *Advanced in Computer Science and its Applications, Springer Berlin Heidelberg*, 279:829-836 2014.

[140] Shen, Jin-Shyan Lee ; Yu-Wei Su ; Chung-Chou. A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. *In Processing of the 33rd Annual Conference of the IEEEIndustrial Electronics Society (IECON 2007)*, Taipei, Taiwan, IEEE, p. 46-51, 2007.

[141] Sharmilaa S., Jagadeesan A., Sathesh Kumar T. Shortcut Tree Routing Algorithm for Efficient Data Delivery in ZigBee Wireless Networks. *International Journal of Innovative Science and Modern Engineering (IJISME)*, 2(4):11-15, 2014.

[142] Qingchao Gong, Guangming Li, Yong Pang. Design and Implementation of Smart Home System Based on ZigBee Technology. *International Journal of Smart Home*, 8(6):143-156, 2014.

[143] Claro Noda, Shashi Prabh, Mario Alves, Thiemo Voigt. On packet size and error correction optimisations in low-power wireless networks. *In Proceedings of IEEE International Conference on Sensing, Communications and Networking (SECON)*, IEEE, p. 212-220, 2013.

[144] Ehsan Ullah Warriach, Eirini Kaldeli, Alexander Lazovik, Marco Aiello. An Interplatform Service-Oriented Middleware for the Smart Home. *International Journal of Smart Home* 7(1):115-142, 2013.

[145] Thinagaran Perumal, Abdul Rahman Ramli, Chui Yew Leong, Shattri Mansor, Khairulmizam Samsudin. Interoperability for Smart Home Environment Using Web Services. *International Journal of Smart Home*, 2(4):1-16, 2008.

[146] Jonas Gustafsson, Rumen Kyusakov, Henrik Mäkitaavola, Jerker Delsing. Application of Service Oriented Architecture for Sensors and Actuators in District Heating Substations. *Sensors*, 14(8):15553-15572, 2014.

[147] KamalEldin Mohamed, Duminda Wijesekera. A lightweight Framework for Web Services Implementations on Mobile Devices *In Proceedings of 1st International Conference on Mobile Services*, Honolulu, HI IEEE, p. 64-71, 2012.

[148] Engelen, Robert van. Code Generation Techniques for Developing Light-Weight XML Web Services for Embedded Devices. *In Procceedings of ACM SIGAPP SAC Conference (Embedded Systems Track)*, p., 2004.

[149] Wei Wang, Payam Barnaghi, Gilbert Cassar, Frieder Ganz, Pirabakaran Navaratnam. Semantic sensor service networks. *IEEE*, p. 1-4, 2012.

[150] Ehsan Ullah Warriach, Eirini Kaldeli, Jaap Bresser, Alexander Lazovik, Marco Aiello. Heterogeneous Device Discovery Framework for the Smart Homes. *In Proceedings of Conference of Exhibition (GCC)*, Dubai, IEEE, p. 637-640, 2011.

[151] Zhi Wang, Jizhong Zhao. Improved Algorithm for UPnP Discovery in Smart Space. *3rd International Conference on Software Engineering and Service Science (ICSESS)*, Beijing IEEE, p. 519-522, 2012.

[152] Nuno Costa, António Pereira, Carlos Serôdio. Integration of Resource Poor Wireless Sensor Networks into Smart Spaces. *In Proceedings of 8th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC '09),* Chengdu IEEE, p. 496-501, 2009.

[153] Noha Ibrahim, Frédéric Le Mouël. A Survey on Service Composition Middleware in Pervasive Environments. *IJCSI International Journal of Computer Science*, p. 1-12, 2009.

[154] Sanjin Sechic, Fie Li, Schahram Dustdar. COPAL: A Macro Language for Rapid Development of Context-aware Applications in Wireless Sensor Networks. *In Proceedings of the 2nd Workshop on Software Engineering for Sensor Network Applications*, ACM, p. 1-6, 2011.

[155] Diego Adolf, Ettore Ferranti, Stephan Koch. SmartScript- A Domain-Specific Language for Appliance Control in Smart Grids. *In Proceedings of 3rd International Conference on Smart Grid Communications (SmartGridComm)*, p. 465-470, 2012.

[156] Parr, Terence. *The Definitive ANTLR Reference:Building Domain Sepcific Languages*. Pragmatic Bookshelf. 2011.

[157] Rosslin John Robles, Tai-hoon Kim. A Review on Security in Smart Home Development. *International Journal of Advanced Science and Technology*, 15:13-22, 2010.

[158] Jon Robinson, Ian Wakeman, Dan Chalmers Composing software services in the pervasive computing environment: Languages or APIs? *ELSEVIER: Pervasive and Mobile Computing*, 4(4):481-505, 2008.

[159] Daniel Retkowitz, Sven Kulle Dependency Management in Smart Homes *Distributed Applications and Interoperable Systems, Lecture Notes in Computer Science , Springer*, 5523:143-156, 2009.

[160] Hamdan Sayuti, Rozeha A.Rashid, Mu'azzah Latiff, A Hamid, N. Fisal, M. Sarijari, Alias Mohd, Kamaludin Yusof, Rozaini Abd Rahim. Lightweight Priority Scheduling Scheme for Smart Home and Ambient Assisted Living System. *International Journal of Digital Information and Wireless Communications (IJDIWC)*, 4(1):114-123, 2014.

[161] Tamim Sookoor, Kamin Whitehouse. RoomZoner: Occupancy-based Room-Level Zoning of a Centralized HVAC System. *In Proceedings of the International Conference of Cyber-Physical Systems (ICCPS)*, Philadelphia, PA, IEEE, p., 2013.

[162] Sílvia Resendes, Paulo Carreira, André C. Santos Towards automatic conflict detection in home and building automation systems. *Pervasive and Mobile Computing*, 12:37-57, 2014.

[163] Rui Camachoa, Paulo Carreiraa, Inês Lyncea, Sílvia Resendesa An ontology-based approach to conflict resolution in Home and Building Automation Systems. *Expert Systems with Applications, Elsevier*, 41(14):6161–6173, 2014.

[164] Sílvia Resendes, Paulo Carreira, André C. Santos. Conflict detection and resolution in home and building automation systems: a literature review. *Journal of Ambient Intelligence and Humanized Computing, Sprinker*, 5(5):699-715, 2014.

[165] ORACLE. *Architecting BPEL Systems for High Availability*. ORACLE. 2007. from,

[166] Kennedy, Deanna Bradshaw and Mark. Oracle BPEL Process Manager Developer's Guide, 10g (10.1.3.1.0) from, http://docs.oracle.com/cd/E11036_01/integrate.1013/b28981.pdf (accessed 20.05.2015).

[167] Sirajum Munir, John Stankovic. FailureSense: Detecting Sensor Failure using Electrical Appliances in the Home. *In Proceedings of 11th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, IEEE, p. 73-81, 2014.

[168] Angel Jesus Varela-Vaca, Rafael M. Gasca, Diana Borrego, Sergio Pozo. Towards Dependable Business Processes with Fault-Tolerance Approach. *In Proceedings of 3th International Conference on Dependability*, IEEE Computer Society, p. 104-111, 2010.

[169] Um, Dugan. Massive Sensor Array Fault Tolerance: Tolerance Mechanism and Fault Injection for Validation. *Journal of Robotics*, p. 1-8, 2010.

Curriculum Vitae

Personal Data

Name:	Abdaladhem ALBRESHNE
Date of Birth:	August 19th, 1973 in Agelat (Libya)
Nationality:	Libyan
Marital Status:	Married

Education

2011-2015:	PhD Assistant, Department of Computer Science, University of Fribourg, Switzerland.
2002-2004:	MSc (Master of Science) in Computer and Communication Networks, mention A, Institut National des Télécommunications (Télécom SudParis), Paris, France.
1992-1997:	BSc (Bachelor of Engineering Science), field of Telecommunications Engineering, level: very good, University of Tripoli, Tripoli, Libya.
1990-1991:	Higher Secondary Diploma, level: excellent, High Secondary School of Agelat, Agelat, Libya.

Languages

- Arabic
- English
- French
- German (spoken)

Publications

1. Abdaladhem Albreshne, Jacques Pasquier. A Domain Specific Language for High-Level Process Control Programming in Smart Buildings. *In Proceedings of the 6th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2015)*. Berlin, Germany: Elsevier. 63:65-73, 2015.

2. Abdaladhem Albreshne, Ayoub Ait Lahcen, Jacques Pasquier. Using a Residential Environment Domain Ontology for Discovering and Integrating Smart Objects in Complex Scenarios. *In Proceedings of the International Workshop on Enabling ICT for Smart Buildings (ICT-SB 2014)*. Hasselt, Belgium: Elsevier. 32:997-1002, 2014.

3. Albreshne Abdaladhem, Ayoub Ait Lahcen, Jacques Pasquier. A Framework and its Associated Process-Oriented Domain Specific Language for Managing Smart Residential Environments. *International Journal of Smart Home*, 7:377-392, 2013.
4. Albreshne Abdaladhem, Jacques Pasquier. A Template-Based Semi-Automatic Web Services Composition Framework: Case Study of Smart Home Management. *In Proceedings of the PhD Symposium at the 9th IEEE European Conference on Web Services ECOWS11*. Lugano, Switzerland, p. 11-17, September, 2011.
5. Albreshne Abdaladhem, Jacques Pasquier. Semantic-Based Semi-Automatic Web Service Composition. *In Proceedings of the 5th Libyan Arab International Conference on Electrical and Electronic Engineering LAICEEE*. Tripoli, Libya, 1: 603-615, Octobre, 2010.