Context-Aware Management of Multi-Device Services in the Home

MPhil Thesis

MPhil Student : Mr. Ioannis Barakos

Supervisor : Dr. Stefan Poslad

School of Electronic Engineering & Computer Science

Queen Mary, University of London

Declaration

The work presented in the thesis is the author's own.

DATE:

SIGNATURE:

To Kostas and Maria

Abstract

More and more functionally complex digital consumer devices are becoming embedded or scattered throughout the home, networked in a piecemeal fashion and supporting more ubiquitous device services. For example, activities such as watching a home video may require video to be streamed throughout the home and for multiple devices to be orchestrated and coordinated, involving multiple user interactions via multiple remote controls.

The main aim of this project is to research and develop a service-oriented multi-device framework to support user activities in the home, easing the operation and management of multi-device services though reducing explicit user interaction. To do this, user contexts i.e., when and where a user activity takes place, and device orchestration using pre-defined rules, are being utilised.

A service-oriented device framework has been designed in four phases. First, a simple framework is designed to utilise OSGi and UPnP functionality in order to orchestrate simple device operation involving device discovery and device interoperability. Second, the framework is enhanced by adding a dynamic user interface portal to access virtual orchestrated services generated through combining multiple devices. Third the framework supports context-based device interaction and context-based task initiation. Context-aware functionality combines information received from several sources such as from sensors that can sense the physical and user environment, from user-device interaction and from user contexts derived from calendars. Finally, the framework supports a smart home SOA lifecycle using pre-defined rules, a rule engine and workflows.

Acknowledgements

Firstly, I would like to take this opportunity to thank my supervisor, Dr. Stefan Poslad, for his guidance, experience, impressive visions that guided my whole research all the way through and his incredible patience. I would like to give my big thank to the School of Electronic Engineering and Computer Science, Queen Mary University of London for their support during my PhD research. For part of my project, I knowledge that I was also funded by an EPSRC Industrial Case award (EP/C537831/1) and by British Telecommunications plc ("BT") as the industrial sponsor. As a result, I thank also two key personnel from BT that gave much guidance on the research and development, John Sherpherdson (who has since left BT) and (Dr.) Dr Botond Virginas, Principal Research Professional at Martlesham Heath, BT.

It is also a pleasure to thank my research colleagues and friends who helped me and also made this journey much more enjoyable, in particular, Kraisak, Dejian, Zhenchen and Janny.

Finally, my love and gratitude go to my family, for their great love and support, and their encouragement have made the journey easier, without them, this thesis would not have happened.

Table of Contents

Table of Figures

Index of Tables

1 Introduction

1.1 Motivation

The vision of ubiquitous computing, UbiCom [1], is that we will become immersed in a digital world that will enable us to transparently access services to support user-centred activities based upon our current context and needs. For example, when we are travelling, we will have access to relevant maps and route information centred about our current location context. When we are in a voice call and need to arrange an appointment, a shared calendar will pop up, determined through detection of our time context and intended task context. The term, and use of the term, context is multi-dimensional [2]. There are multiple types of context which affect user activities such as physical world context (e.g., location, time, weather etc), human context (individual versus social identity, competency and goals) and distributed computing (ICT) context. A context can be sensed, presented to users, automatically adapted and can be used to control environments. A context can affect both pre-planned executing tasks or act as spontaneous conditions to trigger unplanned tasks (situated actions and tasks).

Much progress has been made in making device interaction more adaptable and more natural in order to support Weiser's vision of hidden computing [1]. For example, the use of touch screens enables fingers to be used as input devices rather than using more obtrusive input devices such as computer pointing devices and keyboards. Touch screens support two dimensional gestures as a means to change selections. Devices such as mobile devices are also increasingly incorporating accelerometer, gyroscope, etc., sensors to define the orientation of the device and to support 3D gestures of hands. For example, changing the orientation of the device can be used to change the orientation of the content being displayed. The incorporation of accelerometers which can sense gestures in 3D gives additional means to support 3D haptic input into systems.

The range of activities in an ICT home includes device and service access throughout the home, dynamic installation and termination of devices, sensors or home appliances, maintenance and upgrading of interoperable appliances and support for

various users and types of activities such as entertainment, leisure, work and social interaction. The number of computing devices that people use to support their personal and work activities in their homes is growing. The increasing number of possible combinations of connected ICT home appliances and devices, the fault identification, the diversity of house environments and the partial understanding of events and activities by different users implies a major increase in complexity when operating and managing ICT based home services.

This project will explore how to support and manage a wider variety of (sensor based and controller) devices situated more in the physical environment, devices that are physical context aware and that can promote the vision of UbiCom. Two main types of systems will be researched and developed. First, a managed remotely accessible system portal or hub enables multiple physical devices that are situated in specific physical, human and ICT contexts, to interoperate and to be orchestrated, and choreographed[1] at a peer-to-peer level, in an open service environment (Sections: 4.4, 4.5 and 4.7). Second, more flexible, programmable, local sensor and devices such as the Sun SPOT (Sun Small Programmable Object Technology) [3] sensor platform (Section 4.6) is used to sense the physical world, to determine human environment (user's) contexts, to assess how these affect planned tasks and how they trigger spontaneous tasks.

1.2 Research Objectives

The research objectives are:

1. Investigate more flexible operation and management of ubiquitous computing applications which span multiple devices. Of particular interest are techniques for orchestrating and choreographing the interaction of multiple distributed devices using SOA (Service Orientated Architectures), based on the OSGi standard specifications.
2. Examine techniques for applying interaction models that will enable more user-centred and intrinsic service interaction. Of particular interest here is to

[1] Orchestration refers to use of centralised control of distributed devices whereas choreography refers to more distributed control of devices.

study the use of both planned user activity models and context-driven situated action models.

3. Examine, propose and implement techniques for sensing, processing and storing user contextual data. Of particular interest are techniques to identify and locate the user inside the home premises and to enable scalable gesture-based device control.

The combined goal is then to enable planned user tasks to be executed across multiple devices situated in the physical world and human-centred environments and to consider how these planned tasks are affected by specific environment contexts and how favourable contexts could spontaneously trigger and schedule new tasks.

1.3 Report Outline

The remainder of this report is structured as follows. Chapter 2 characterises the main properties of advanced device interaction, outlines some applications and then discusses the design issues in a more general, less application specific way. Chapter 3 surveys the work of other researchers along two main themes: how planned users' activities which span multiple devices can be supported and how unplanned, situated action driven activities which can disrupt active planned activities can be supported. Chapter 4 describes some preliminary prototypes undertaken to support planned activities (unplanned activities are discussed in further work). Chapter 5 discusses the use of the framework, achievements during the research project period and specifies a detailed plan for further work.

2 Problem Analysis: Advanced Device Interaction for Future Home Environments

The problem domain focus is on home environments in the future, in which a profusion of different digital information and task devices are both embedded into the physical environment (smart environments) in order to enhance and automate everyday activities, e.g., automatic doors and lighting. Devices can act as tags, sensors, and controllers and can support application specific tasks using an Application Specific Operating System (ASOS) and support multiple tasks using a MultiTask Operating System (MTOS). Smart mobile devices can accompany the users such as phones, remote controls, cameras and can increasingly sense and control digital devices and physical objects in this environment. This environment is referred to synonymously as a smart environment, smart home or home environments which provides the living space for daily activities.

The interaction between humans, devices and the physical world is the result of a number of various activities and tasks. In this research project **an activity** is defined as an action that is performed by a human and may or may not have a purpose. **A task** may be the result of a number of activities and is an action with a purpose. For example when a person has the task to have dinner, he/she performs a number of activities such as cook food, make table and turn the lights on.

The **context** for interaction and can be divided into three domains:

- **The human context.** It is the context domain that holds information about the user such as his location in the house premises, identification and personal preferences.
- **The physical world context.** This context domain holds the information of the physical context: outside temperature, inside temperature, local time and local weather are some examples of the physical context domain.
- **The system context.** System context is the information extracted from the home devices, this information includes: device uptime, device operations, capabilities, configuration, etc.

Context-driven applications exist today but they tend to use a simple inbuilt predefined rule model to perform a task based on the current context. A lighting

system with light sensors may be used to turn on the lights of a room at night and turn them off during the day. Another example of a simple context-driven application is the thermostat-based heating system that regulates the central heating of a house.

Complex context-driven situated action models are influenced by people's actions (human context), the physical environment (physical world context) and device operations (system context).

Advanced device interaction within this research project refers to device interaction which has these properties:

- Planned user activities that may span multiple devices.
- Devices are distributed and interact at a P2P level rather than being operated via a centralised controller.
- Use of devices by human users can often be affected by unplanned spontaneous situated actions by human users which cause planned activities to be altered in some way, e.g., by being suspended and resumed.
- Users' mental models of understanding the complexity of multiple interacting devices may differ from the actual operating system model of these devices: they may need to be relate multiple levels of the system at different levels of abstraction.

2.1 Use Scenarios

The following table (Table 2-1) summarises some example scenarios of next generation device interaction.

Future Home Interaction Scenarios	**Example**
Orchestrate service interaction which spans multiple devices (scenario 1)	Orchestrate AV content delivery, sound and lighting, possibly from one console or service portal
User Device Portal interaction (scenario 2)	Orchestrate AV content delivery and combining the various device interfaces into one, creating a virtual service that a user can interact with.
Context-aware device control (scenario 3)	Situated, task specific, control devices can be dynamically configured and regulated with respect to user context. Detect user actions and gestures and control devices based on where the user is located and where he gestures to.
Policy-based device Management (scenario 4)	Device interaction and interoperability based on pre-defined policies and events generated from devices and sensors, e.g., someone enters the living room, a motion sensor is triggered by this event and a policy instructs the light service to turn on the light of the living room.
User goal-based planner interaction using rules and workflows (scenario 5)	Workflows and rules drive the lifecycle of a service: installation, operation and maintenance, uninstallation.

Table 2-1: Future Home Scenarios

Although these scenarios seem deceptively simple, the challenge for future home environments is that they will require substantial advances in existing ICT system research and design in order to support them. This challenge and the advances needed are described below.

2.1.1 Basic Device Infrastructure

In a typical home entertainment system a variety of devices exist such as a radio, music player with radio, video player (e.g., DVD) or recorder, one or two audio-video amplifiers, a television, sound speakers and maybe a lighting diming system. Each of these devices may have its own control device and the user needs to control each of these devices separately even if he has only one goal (e.g. to watch a movie). Some of his actions may include: turn the radio off, turn the radio amplifier off, turn the DVD player on, turn the DVD 5.1 audio amplifier on, turn the television on, adjust volume and decrease the light intensity. Furthermore, the goal “change the volume” requires a

user to interact with at least three devices: DVD player, television and a surround sound system. As each of them has its own volume control.

In a more user-centred scenario, the devices interact with each other allowing the user to give a single instruction about his goal (e.g. turn the volume down) using a single action even if this goal needs more than one device to operate to reach this goal.

2.1.2 User Device Portal Interaction

A home may include over a hundred electronic devices and use a basic or more sophisticated operating system and a screen. Multi device interaction raises the problem of having multiple user interfaces for users interact with.

A device portal in this scenario allows the devices to interact and allows the user to combine multiple device interfaces into one main interface that controls the interoperable devices. The picture and volume settings from the television, the DVD player and the surround system are combined from three interfaces into one and the user controls it using a single controller (e.g. DVD's remote control). The user in this scenario interacts with a virtual service that is the result of various services offered by different devices.

2.1.3 Context-aware Device Interaction

The next scenario expands the previous scenario by enabling a control device (e.g. a sensor-based, e.g., Sun SPOT or Nindendo Wii, Controller) to point at spaces in the physical world and to get information and to interact with home services by pointing at physical objects.

Home users may interact with a mobile device that acts as a universal controller. This device has various sensors and run modules that allows services to query the user's location, position and hand movement (simple gestures) in the home. In this scenario a user points the universal controller towards the window (physical object) and then points it towards the television screen (device). The two objects are identified and a planner works to reach the user goal "show me the weather forecast on the screen".

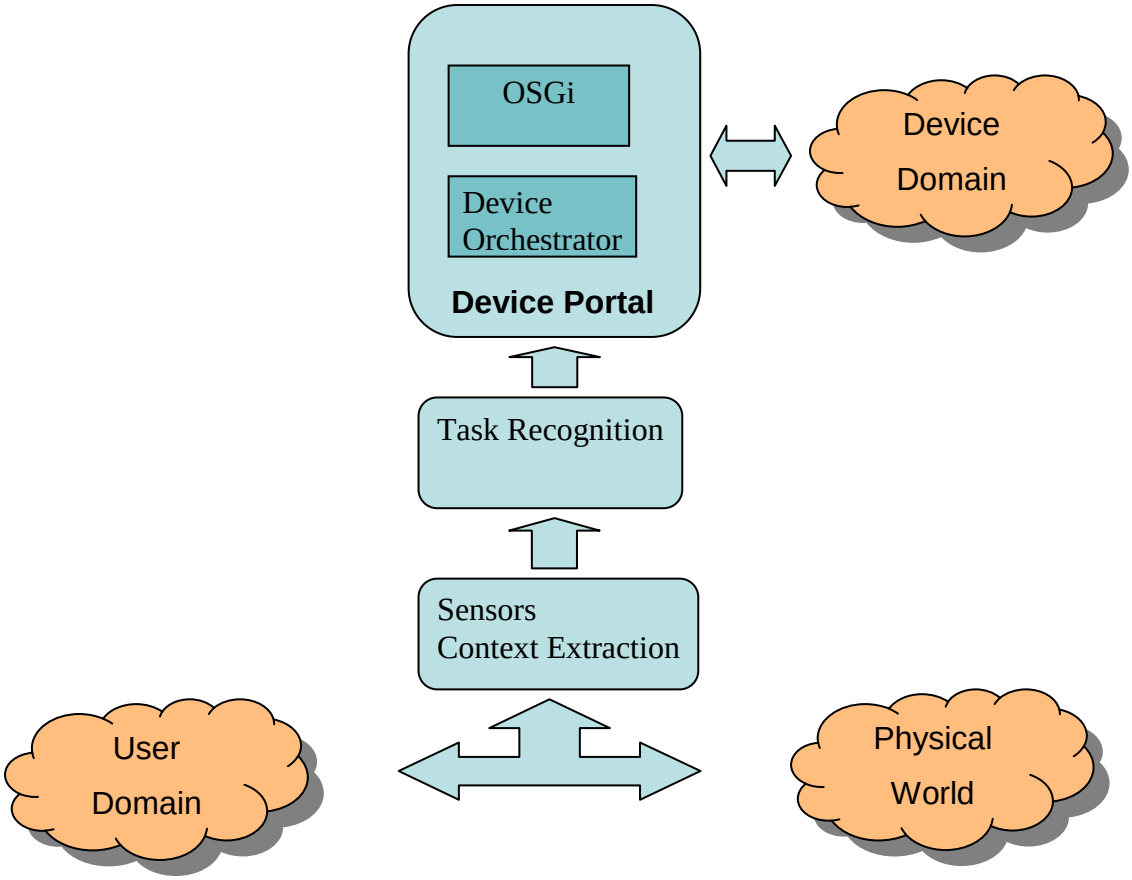


Figure 2-1: Interaction between the three context domains (Device, Physical world and User) – Context Aware device Interaction

In Figure 2-1 the interaction in three context domains is presented in a more generic view. Activities between a User Domain and a Physical World domain are captured and extracted by sensors (scenario example: user points with the controller towards the window). A Task Recognition process identifies the user goal (scenario example: watch the weather forecast) and sends this to the device orchestrator located in the device portal. The device orchestrator then creates a method to achieve the user goal and feeds this plan to the OSGi framework. Finally, the OSGi framework sends control messages to devices following the plan received by the orchestrator.

2.1.4 Policy-based Device Management

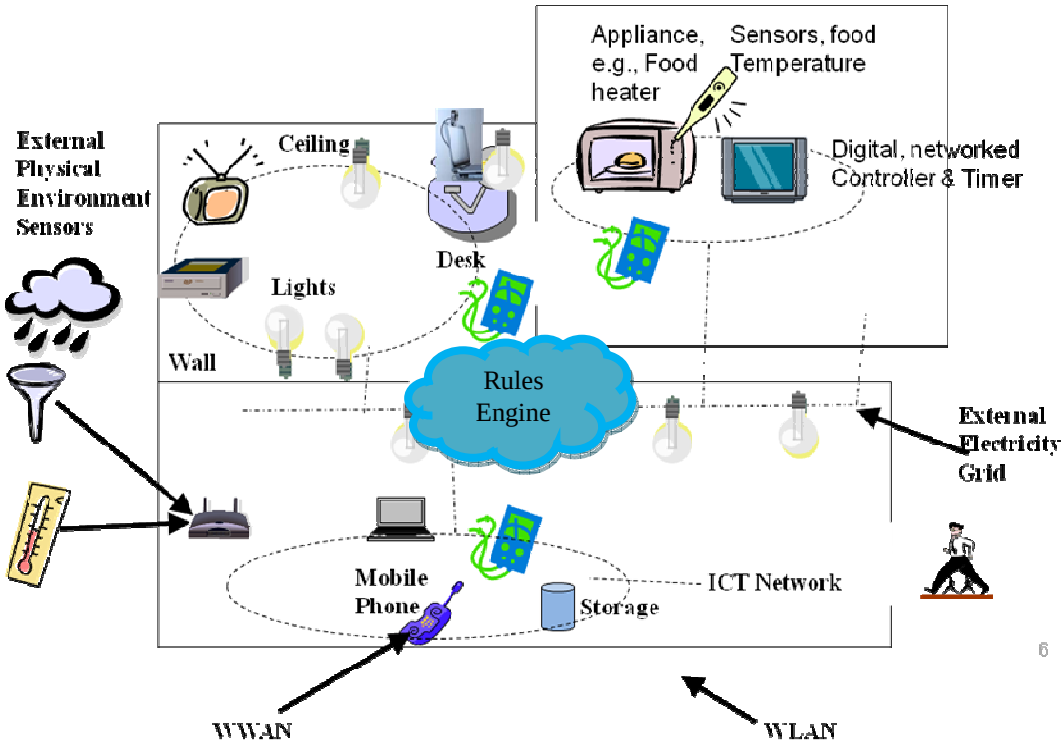


Figure 2-2: Various home services and networks controlled by a rule engine

A set of pre-defined policies may be used to configure and control one or more services in the home. Policies are rules that specify what action can be performed when one or more conditions are satisfied. These types of systems use an Event Condition Action (ECA) or event driven architecture and consist of three parts: the event, the condition and the action part.

The *event* part specifies the signal that triggers the invocation of a specific rule. The *condition* part is a logical test that if satisfied or evaluates to be true, it causes the action to be carried out. Finally, the *action* part consists of invocations on the local service or data. The next scenario demonstrates an ECA architecture running in a home domain.

A house is equipped with motion sensors in each room and door sensors. A "low-energy consumption" policy would make each of the home's lights to be turned on only when a person is situated inside the room, only at specific times and when there is no sunlight and the room is dark. In that case an event driven policy manager takes context data (e.g. sensors' data, time) (events), applies its rules (conditions), and controls the services through the home's gateway (action). An example of a rule that

instructs the lights of a room to turn on each time someone enters the room can be considered as:

```
IF   SENSOR_DOOR_1.isTriggered()==true   and   TIME.now()>17:00   and
TIME.now()<07:00
      THEN LIGHTS_ROOM_1.setOn()
```

Figure 2-3 shows the ECA architecture for the above scenario and its main components.

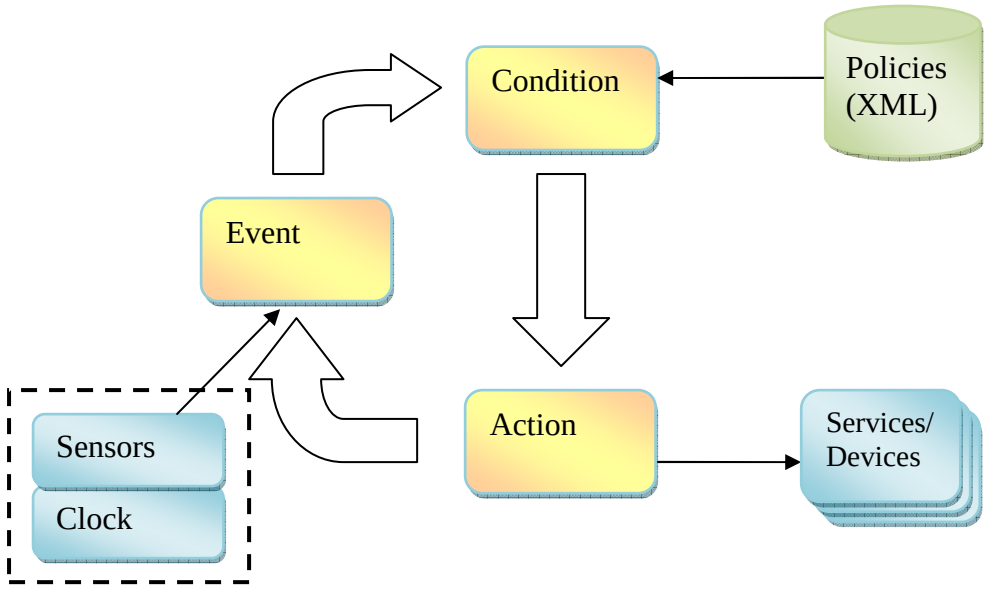


Figure 2-3: User to service interaction using event driven actions (policy manager)

Context information (sensors, time, and location) triggers the event process. An event process feeds the contextual data into the condition process that searches for conditions (policies, rules) in a rules database. When a suitable policy that meets the specified condition is found, the action part is triggered. The action process sends control messages to devices and services based on the policy that triggered it and also may trigger a new event and so on.

2.1.5 User Goal-based Planner Interaction Using Rules and Workflows

The device and service life-cycle in a home environment includes four phases: installation: where someone configures the device for operation in the home network; operation: the operation of the devices from its users; maintenance: the re-

configuration and updating of the device and finally, uninstallation: the removal of the device from the home network.

The above four phases may seem complicated to people without technical and electronics skills and experience. These can become much more complicated when more networked and interoperable devices are installed in a house.

To solve this problem this scenario uses a planner that plans the above four phases (installation, operation, maintenance and uninstallation) of each device based on what the user wants to do (the user goal). A service workflow begins the process from phase one when a new device is introduced into the home environment and is driven by pre-defined rules and user events to configure the new device and any related home devices (on demand).

For example, consider the multimedia installation described in the previous scenarios. When a new video player (e.g. an internet enabled Blu-ray player) is turned on for the first time in the home, the planner triggers a first phase's workflow where the configuration of the player takes place. Pre-defined rules and UPnP messages configure the player. When extra configuration details are needed they are requested from the user in easy to understand queries (e.g. shown on the television screen). After the configuration phase is complete the planner triggers the next two workflows: operation and maintenance workflows to work in parallel. Operation workflow puts a device in stand-by mode until an event is received from a user or from another device. Maintenance workflow is on stand-by as well until a miss-configuration or a device malfunction event is received. Pre-defined rules in these workflows decide the actions that the device should take.

2.2 Device and Service Use

Services in a home may be divided and distributed across high-resources devices and low-resources devices. A clarification of the various features of the home devices that are considered in this project is listed on the next table (Table 2-2).

Device Feature	Characteristics
Simple/ Complex service access	Device provides services that are simple or complex to initiate and operate.
Mobility	The device is situated inside the premises and is static (e.g. desktop computer, TV set) or it is a mobile device (e.g. PDA, laptop, wireless camera)
Open Access	The device can execute both local processes and remote processes
Shared resources and services	The device has the ability to share a service with multiple devices.
Networked	The device has a network connection with the ICT home.
Local resource capabilities	How powerful the device is in terms of CPU speed, memory, network connection and storage.
Context-aware	The device can be aware of the current context. Context-aware devices fall in three sub-categories (section 2.4): ICT context-aware devices Physical context-aware devices User context-aware devices

Table 2-2: Features of ICT home devices and their characteristics

Home devices typically embody services in fixed systems to support task-specific functions and they have limited ICT resources. However, even these fixed system devices can be part of a distributed home network if they at least support networking and sharing their resources.

In a Service-Oriented Architecture (SOA), a service is organised and utilised by a number of distributed components providing required capabilities. Thus, even the lower-resources devices can be used in an SOA to support a service as long as there is a way to interoperate and share their capabilities.

In a SOA, distributed applications are built by orchestrating or choreographing reusable services using high-level workflows or business processes. The complexity

of developing and maintaining these processes is addressed by SOA development cycles that identify the roles of participants at each stage.

2.2.1 Device Service Lifecycle

SOAs can facilitate the practical development of large-scale systems. Some of the s limitations of SOAs are based on the use of a centralised design of their core services that is more sensitive to errors and provides a single point of failure. For example, if the component that is responsible for service discovery (service discovery component) goes offline in the SOA network, then the discovery service will remain unavailable until that component goes online again.

The life-cycle includes four phases: the *Creation*, *Execution*, *Maintenance* and *Dissolution* of the service. Each of these phases contains a number of actions that need to be taken in order for the phase to be completed. For example to successfully complete the Creation phase a service must have mechanisms to propose, discover and select other services and to successfully create an initial plan or workflow that will enable the proposed and selected services to interact.

A typical service interaction life-cycle is summarised in Figure 2-4. Each life-cycles phase includes a number of actions (e.g. 1.1 Service Proposal, 1.2 Service Discovery, etc.) that is either part of a CCI or CHI interaction (CCI actions are highlighted).

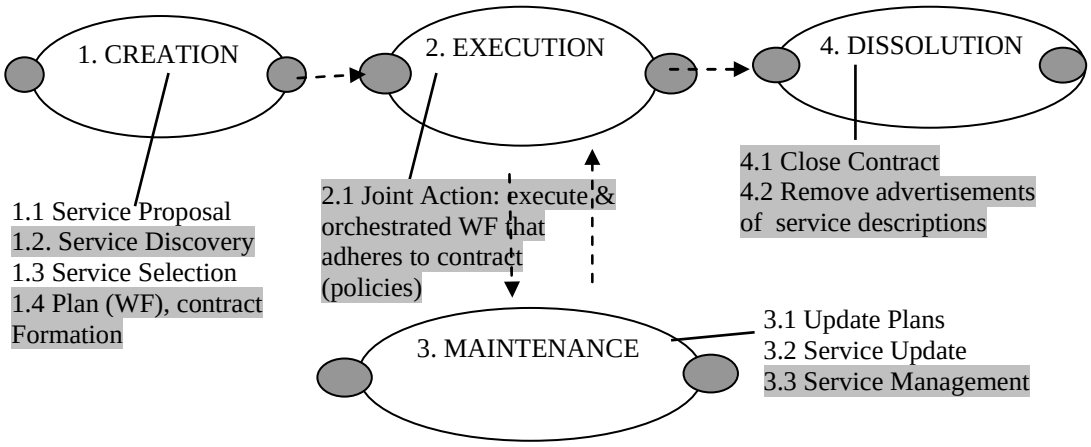


Figure 2-4: Service Oriented Architecture model (CCI actions are highlighted)

2.3 Types of Interaction between Users, Devices and the Physical World

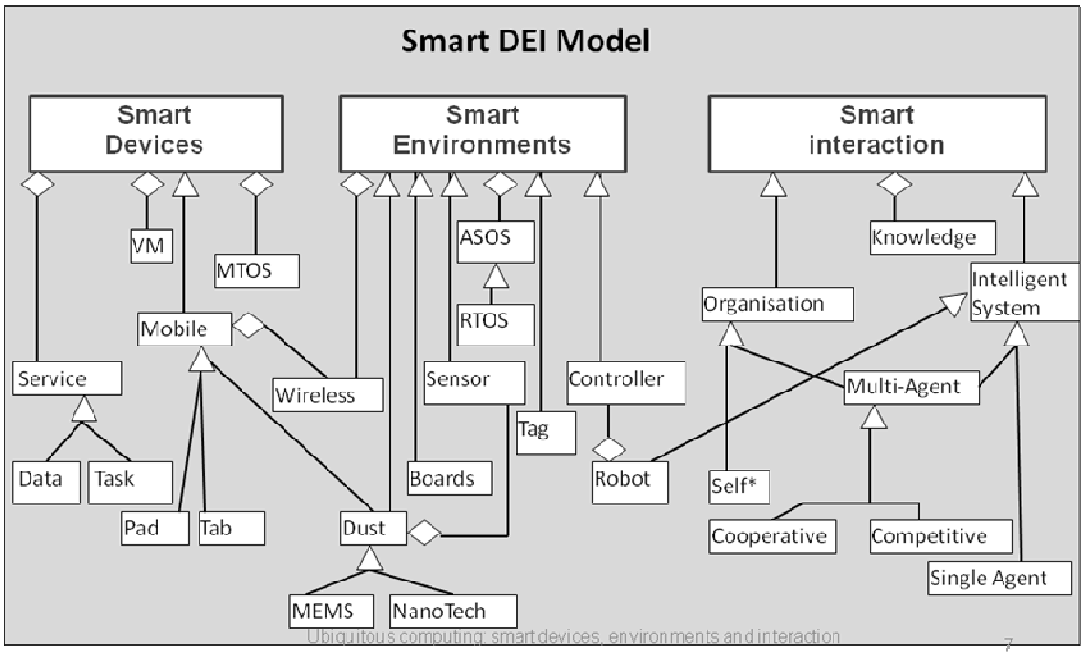


Figure 2-5: Smart subsystems and components [2]

Home services act in open system environments. Home environments are considered as smart environments as they are dynamic and complex: they are uncertain and non-deterministic, are partially viewed or sensed by the services, are sequential and consist of other intelligent components and services. A system's smart and context-aware environment can be categorised in three domains as described in the next section (section 2.4): the physical environment (physical world environment), the device operations (system environment) and the people's actions (human environment).

Based on the above three environment domains there are five types of interaction in a home environment: interaction between humans, interaction between a human and the physical world, interaction between a human and home devices and services, interaction between home devices and finally interaction between the physical world and home devices. Table 2-3 lists these interaction models. Human to human interaction (HHI) models are not the focus of this research project and will not be described any more in this document.

Interaction Model	Example
Human to Human (HHI)	People socialise in the house, share activities, communicate, etc.
Human to World (HWI)	People interact with physical objects in the house: move chair, open door. People move, change their location.
Human to Device (HCI)	People interact with devices: operate devices, configure devices, install new devices, etc.
Device to Device (CCI)	Devices interact with each other, e.g., a TV switches to a SCART input when a DVD player turns on.
World to Device (WCI)	Physical world triggers devices (sensors), e.g., daylight triggers a light sensor to turn the lights off, wind speed changes triggers a weather sensor to measure wind speed, etc.

Table 2-3: Different interaction models in the home environment

2.4 Device environments

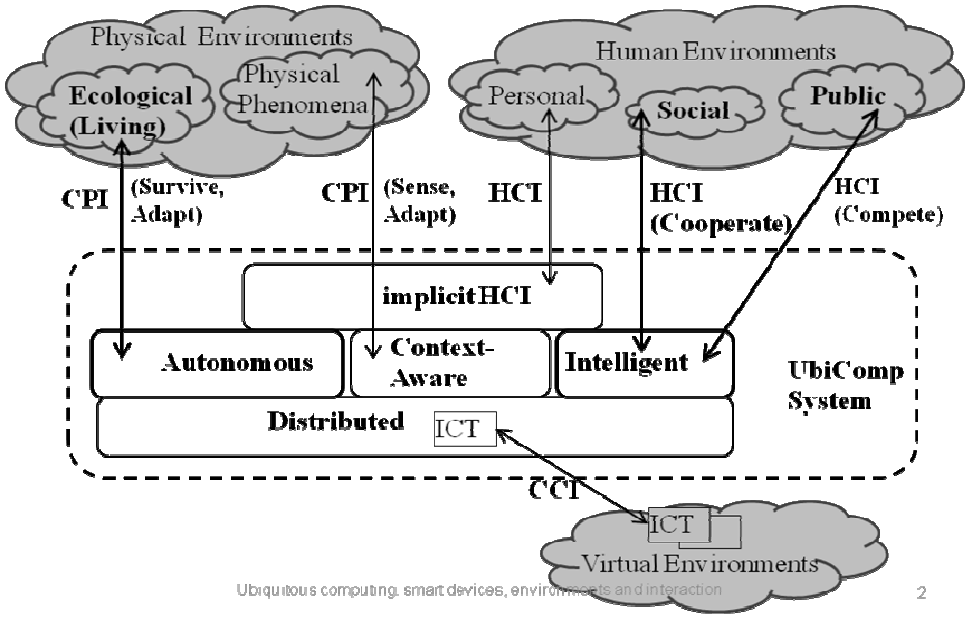


Figure 2-6: Extended set of internal system properties and device environments [2]

Device interaction within each of the environment domains varies between devices (Figure 2-6). There are devices designed to interact only with the human environment and react only to simple control commands that a user applies while other smarter devices execute services in each of the environment domains (e.g. support user

interaction, sense the physical world and exchange information with other services simultaneously).

2.4.1 Physical Environment

The physical environment domain contains the context information of the physical world. This information includes for example, the outside temperature, local time and daylight intensity. Physical environments are not fully observed by devices and the amount and type of the physical environment context constraint experienced depends on the type and amount of sensors a device includes.

Devices that operate in this environment domain are aware of the physical environmental changes on behalf of the user and may automatically adjust the system to the context without the user being aware of it. Design issues in these devices include people's privacy (e.g. if the device may acquire the identification and position of a user), how and what amount of physical environment information is acquired, where the acquired context is being stored and what context information and to whom it is distributed to.

2.4.2 ICT Environment

The system (ICT) environment domain contains the information that is extracted from the devices and services registered in the home network (also called distributed computer network domain). This information may include: device configuration data, device capabilities, service outputs (e.g. video, sound) and sensor data exchange.

The variety of home devices makes the ICT environment more complex: it can be non-deterministic as devices can enter and leave the environment and it is partly observable. Design issues in devices interacting with the ICT environment include: how the messages are exchanged between devices, how to adapt to system changes and re-configurations, and how to identify and adapt to system failures.

This environment domain also depends on the physical environment domain, as some factors of the physical environment can affect the ICT environment. For example,

really bad weather may affect the satellite video broadcasting received by a set-top box.

2.4.3 Human Environment

The human environment domain holds the user context information. Examples include: user identification, user activities, user interaction with a service and user personal preferences (user profiles). Humans interact with both the physical environment (they open the windows in a sunny day) and the system (ICT) environment (they turn on the lights when it's dark). They can provide personalised information to the system environment to make the system adapt to their needs. They may use the home services while they are away from home. Some design issues in this environment domain include: Where and how the human preferences are modelled and stored, which devices and what amount of human information a device can extract and what methods and sensors can be used to extract human information such as location and identification.

3 Literature Survey

Present day home environments include a range of digital and analogue devices ranging from light switches to desktop computers, digital satellite set-top boxes and various types of handheld mobile devices. Moreover, various input/output wireless devices exist such as remote controls, wireless mice/keyboards, displays, printers, projectors and accessories such as storage media and various types of cables. This increased number of analogue and digital devices in the home and the variety of their accessories generate a number of open questions listed in Table 3-1.

Open question	**Example**
How devices can be utilized in different combinations and to what extent?	A device offering service A to be combined with a device offering service B to produce a new service C.
How to enable users to operate home services based on their habits, behaviours and (past) experiences	A user does not want to receive telephone calls from work while at home between 6:00pm to 9:00pm
How to acquire context information from a home environment or user	Types of sensors to installed in the house. Location of where these sensors should be installed. Filtering of data acquired from sensors.
How to generate user goals by observing their daily in-home activities, events and experiences within a home environment (and devices)	User enters home User sits down User relaxes User usually takes a shower after work (past experience) Goals: Warm up the house Make bathroom ready for a shower

Table 3-1: Open questions generated by the variety of devices and accessories in home environments.

This survey focuses on and is organised into three main topic areas. First, general challenges of using increasing numbers of heterogeneous digital devices with a greater ad hoc device interaction capability to support smart environment activities, are discussed. Second, issues to do with how planned activities that can span multiple autonomous and heterogeneous devices can be supported, are considered. Third, how unplanned situated activities, related to context (change) determination can be supported, is analysed. Finally, an analysis of this related work is given.

3.1 Visions for Pervasive Device Interaction

According to Weiser [1], work on ubiquitous computing began at Xerox Palo Alto Research Center or PARC in 1987 when Bob Spraque, Richard Bruce and others proposed developing wall-sized displays. Later in 1989 Olivetti Research labs and the University of Cambridge developed the first context-aware computing application: the Active Badge system [4].

In the late 1990s, Philips proposed the Ambient Intelligence project as a novel paradigm for consumer electronics that are sensitive to, and responsive to, the presence of people. Prototypes of Ambient Intelligence ranged from electronics that could recognise voice and movement to digital displays within a bathroom mirror to new “toys” that were designed to help children expand their creativity [2]. The Aura project at Carnegie Mellon University [5] introduces the concept of a personal information aura: an invisible part of computing and information services that persists regardless of location and that spans wearable, hand-held, desktop and infrastructure computers.

Giving everyday objects the ability to connect to a data network would have a range of benefits: making it easier for homeowners to configure their lights and switches, reducing the cost and complexity of building construction, assisting with home health care. Many alternative standards currently compete to do just that - a situation reminiscent of the early days prior to the Internet, when computers and networks came in multiple incompatible types.

The Internet-0 (I0) project defines seven principles that extend the original notion of internetworking to inter-device internetworking [6].These principles define specifications and properties for designing and implementing internetworking devices. Each connected device must include a number of tasks and processes to produce one or multiple services in order to be useful for the smart space – device services.

Several researchers have analysed the general research challenges in developing next generation device services. Whereas, Harper et al. [7], and Abowd and Mynatt [8] focus on the human environment characteristics, Edwards and Grinter focus on the system characteristics [9].

Harper et al. [7] in a report called “Being Human: Human-Computer Interaction in 2020” reflects on how changes in smart devices will lead to major transformations in the nature of human computer interaction. They have illustrated these transformations using three cases studies: trafficking content across mobile devices; tracking versus surveillance in families and augmented personal human memories.

The way that people use multiple services in the home implies four problems [10]. First, associating a user’s activities with a particular device is problematic for multiple device users because *many activities span multiple devices*. Second, *device use varies with respect to users and their circumstances*. Users assign different roles to devices both by choice and by constraint. For example one person may use a PDA only to receive and make phone calls while another person may take advantage of other PDA’s capabilities such as web browsing, e-mail viewing, audio and video playback, etc. Third, users want to *separate activities across* work, leisure and personal devices, but this is not easy to do in practice because it is difficult to categorise activities into work and leisure types. A user for example may wish to see a combined work and private calendar at work in order to better integrate work and leisure activities and may wish to share their work but not their leisure activities. Finally, users employ a variety of techniques for *accessing information across multiple devices*. This may also be interpreted as users requiring access to information anywhere, anytime. Activities in home use resources that span multiple devices, rather than just using one device for a particular task and they may even request information from resources located outside of the home environment. For example a weather forecasting action may require information from a web server on the Internet. These properties and the associated design principles are summarised in Table 3-2.

Activity Property	**Design Implications**
Activities span multiple devices	Device interoperability
Device use varies by user and circumstance	Configurable, context-aware system behaviours, personalised system configuration
Separation of activities across devices	Context-bounded system behaviours
Access to activities via multiple devices	Device interoperability

Table 3-2: Summary of the activity properties from [10] and the design implications

In the Classroom 2000 project [8], Abowd and Mynatt expounded several design challenges and principles for everyday computing to support more informal, daily, activities. These are summarised in Table 3-3.

Activity Property	**Design Implications**
Some user activities rarely have a clear beginning or end	Provide persistence, visibility of the current state, support multiple levels of activeness; support context-driven or situated actions.
Interruptions are to be expected	Interaction can be modelled as a plan that may be suspended at any time and be resumed at a later time Model unplanned or context- or situated action- driven actions which interrupt and resume plans. Make users aware of uncompleted processes; inconsistencies may arise as device states may change.
Multiple activities operate concurrently	Support for context-shifting amongst multiple activities is needed; support background awareness; interfaces should support multiple levels of "intrusiveness" in conveying monitoring information that matches the relative urgency and importance of events
Activities can be filtered and adapted to the contexts.	Contexts such as time are rarely represented in computer interfaces and available to interlink to activities. Generic, e.g., time, location, and specific contexts, e.g., outcome or rating of a past similar activity, need to be supported,
Activities may interlink with other activities and devices	Associative knowledge-based models of information are needed that support information reuse on multiple occasions, from multiple perspectives

Table 3-3: Summarises the activity properties from Abowd and Mynatt [8] and their design implications

Device interoperability challenges are summarised by Edwards and Grinter in [9] as

Unintelligible interaction, No top to bottom holistic design, piecemeal adoption of devices	Use mediating devices as universal portals, brokers, or local controllers to simplify access or to orchestrate the use of multiple services
Impromptu Interoperability, multi-vendor devices added piecemeal but fluid operation still expected	Detect and Manage situated or context-driven actions Use policies to constrain the interoperability
Inference in the presence of ambiguity of context	Use of multiple sensor sources to reduce ambiguity; Balance full system adaptation versus some user control of
No System Administrator: lack of ability or interest by users in home ICT	Six different design models for user centred management have been proposed [2].
Nature of user activities given in [7] and [8].	Store state of instances of processes of operation of multiple devices;
New social norms created, more open and interlinked access, privacy control of more vulnerable groups	Support both individual private operation and closed public operation
Reliability: current appliances & embedded ASOS systems are more reliable & available than MTOS systems	Designs based upon planning can replan to handle failures. Policies can be used to constrain interactions to prevent problems arising.

Table 3-4: Edwards and Grinter [9] device interoperability challenges

In addition to device interoperability and service control through a home gateway (OSGi), the ability to detect changes in the physical status of things is also essential for recording changes in the environment. In this regard, sensors play a pivotal role in bridging the gap between the physical and virtual worlds, and enable things to respond to changes in their physical environment. Sensors collect data from their environment, generating information and raising awareness about context. For example, sensors in an electronic jacket can collect information about changes in external temperature and the parameters of the jacket can be adjusted accordingly to insulate more or ventilate more.

Embedded intelligence in things themselves will distribute processing power to the edges of the network, offering greater possibilities for data processing and increasing the resilience of the network. This will also empower things and devices at the edges of the network to take independent decisions.

The Internet of Things [6] will draw on the functionality offered by all of these technologies to realize the vision of a fully interactive, responsive and ubiquitous network environment.

3.2 Device Interoperability and Device Gateways

Supporting activities that span multiple devices introduce two problems: how the devices are connected and how the devices transfer information between them. Schilit [22][23] surveys the technologies, standards and research into device ensemble. Device interoperability applies to many system levels that can be divided into four layers:

- Link layer: to enable for example low-power, short range communication
- Network layer; to find the best route of information between connected devices
- Data layer: to support sharing and synchronisation of information (photos, emails, music)
- Application layer: to support applications that span multiple devices where one device may act as an input of information where another one may act as output.

3.2.1 OSGi

Several standards and protocols have been introduced to manage multiple home devices and appliances. Traditionally, a Residential Gateway in a home network ensures that all devices and appliances can be connected to the Internet, while possibly sharing a single address in the Internet Protocol address space. But a Residential Gateway can be used for more than just routing and address translation: it can be used as a service execution platform, enabling service providers to manage their services dynamically within the home network. It can also be used to provide IP based services to non-IP devices such as IEEE 1394-enabled consumer electronics.

An open standards organisation, the Open Services Gateway initiative or the OSGi Alliance is an innovative system that has been proposed to enable device inter-

connection and control in ICT home networks. OSGi specifies a Java service platform framework that defines an application life cycle model and a service registry [24]. This framework defines a number of OSGi services including: configuration and preferences management, logging, http services, UPnP explorer, application tracking, power and device management, ubiquitous security, IO Connector Service and diagnostic services of the connected devices. Components and applications in the OSGi framework can be installed, uninstalled, started, stopped and updated remotely without requiring reboot. New services and device components are detected automatically by the OSGi service platform.

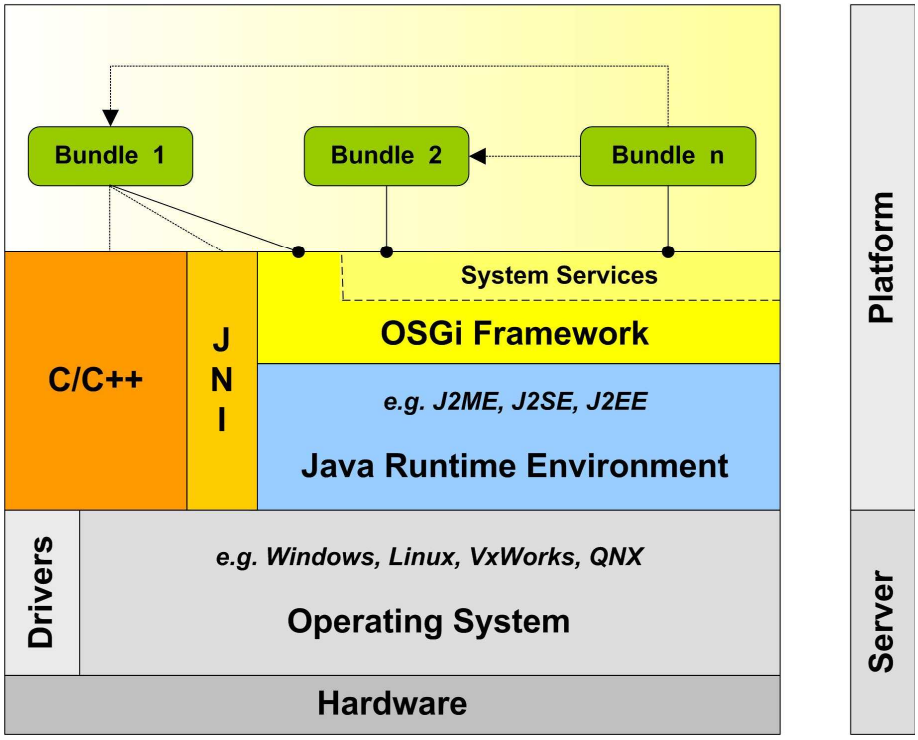


Figure 3-1: The OSGi layer specification

The following table lists a set of principles to guide the development of the OSGi specifications and frameworks [16].

OSGi Principle	Description
Platform Independence	The OSGi software environment can be implemented on many platforms, with widely varying capabilities
Application Independence	OSGi provides a horizontal platform that is applicable in any computing environment where the capabilities of the software environment are useful
Multiple Service Support	OSGi environments are capable of hosting multiple applications from different service providers on a single service platform
Service Collaboration Support	The OSGi environment allows services to be deployed that provide functionality to other services. Applications can dynamically discover these services and adapt their behaviour to the configuration of the environment and other services that are present
Security	An OSGi environment can concurrently support many services from different service providers. Security between these services is of paramount importance
Multiple Network Technology Support	OSGi cannot mandate particular choices of network and it is network agnostic, as far as is reasonably practical
Simplicity	The OSGi environment offers a service environment where the complexity of managing the service environment can be placed into the hands of professionals in the form of the gateway operator. This does not, however, preclude individuals from configuring their own gateway as appropriate.

Table 3-5 OSGi specification design principles

3.2.2 Web Services (WS)

Web Services are Application Programming Interfaces (APIs) that are accessed via the Hypertext Transfer Protocol (HTTP). W3C [34] defines web services as software systems designed to support interoperable machine-to-machine interaction over a network. A web service has an interface described in a machine-processable format called Web Services Description Language (WSDL) [35]. WSDL is used to describe services in terms of actions, input data, output data and service processes. Other systems interact with the web service using the Simple Object Access Protocol (SOAP) [36] which is a lightweight XML-based transport independent protocol for exchanging structured information between peers in a distributed environment.

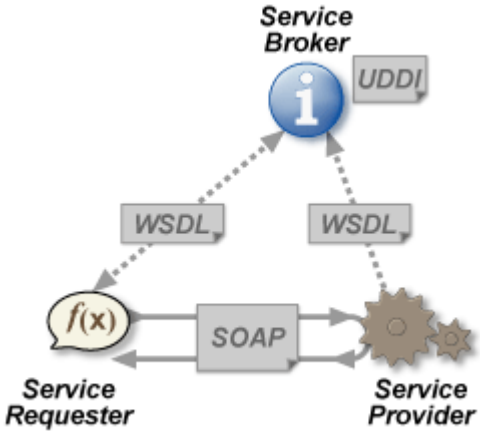


Figure 3-2: Message exchange in Web Service architecture

3.2.3 UPnP

The Universal Plug and Play protocol (UPnP) is another promising technology to allow device control in home networks. It is based on the SOAP protocol and offers pervasive peer-to-peer network connectivity of computers, home appliances and wireless devices. UPnP can be supported on essentially any operating system and in any type of network technology wired or wireless.

3.3 Universal Device Portal

The increasing number of home appliances, including televisions, video players (DVD, Blu-ray players, etc.), light switches and sensors are examples of more computerised devices that offer more services. This makes these devices more complex with complex remote controls that make their interfaces harder to use [38].

Universal portal systems have been introduced in the market during the last few years and their purpose is to make the management of a device or the combination of two or more devices simpler. In such a system, devices use a portal to connect to and to be managed by the portal's interface that is simpler. The aim of these portals is to enable heterogeneous applications on heterogeneous hardware devices to communicate with each other within a common defined hardware and software environment.

The Multimedia Home Platform (MHP) [39] is an example of a portal application that receives digital broadcast video, audio streams and the Internet, offering a common

API that is accessible for other applications, television, set-top boxes and any combination of these.

3.3.1.1 Local Sensing and Control

The complexity of device functions and interfaces make researchers focus not only on building universal portal systems but also on building universal remote controls that offer virtual portal capabilities and LCD screens displaying dynamic interfaces based on user profiles to improve existing device interfaces.

The Personal Universal Controller (PUC) [41] is a remote control device for improving the interfaces to complex appliances. A PUC remotely connects to everyday appliances using a two-way communication and an interface generator that generates graphical and speech interfaces. Hodes, *et. al.* have designed a similar project to PUC called Universal Interactor [42], while the XWeb project creates dynamic interfaces to devices by defining an XML language[43]. Other later approaches include the iCrafter [44] and Roadie [45] with the latter offering goal-oriented interfaces using planning and common-sense reasoning.

3.4 Supporting Planned Activities Spanning Multiple Devices

Observing human activities, determining what data to use and where to acquire this data to support a context-aware application is challenging. This difficulty increases when these activities are not centred on a single device but they span multiple independent devices [11].

The ACHE system[2] designed by Mozer [12] describes a house that learns patterns in its occupants' behaviours and automatically controls some basic house appliances and devices. ACHE monitors the environment, observes the occupants' actions (e.g. adjusting thermostats, turning on a particular configuration of lights) and attempts to infer patterns in the environment that predict these actions.

[2] ACHE stands for Adaptive Control of Home Environments

Cohen et al. in [13] describes a multi agent system that uses multimodal collaborative interface technology to facilitate human interaction with a pre-existing distributed simulator. Cohen's system uses a novel interface that allows users to employ speech and gestures commands simultaneously while a multimodal interpreter is used to engage a distributed agent framework based on the Open Agent Architecture [14] that tries to execute the recognised commands by accessing a number of devices.

A system that supports collaboration of activities is described in [15]. Their system uses a dialogue model that is able to manage conversations about multiple tasks and collaborative activities and builds a multi-modal conversational interface to the devices.

3.5 Context-aware Devices

In a context aware design, applications must be aware of the existence and the characteristics of user's activities and the rest of the networked system operation (system's activities). For context aware applications to function according to its users' expectations, they must consider all the relevant entities that enable them to adapt their behaviour based on the current conditions and situation. These relevant entities are commonly referred to as context and contain the user situation and activities, service situation and state, physical environment information and other relevant information such as available equipment, system configuration and service capabilities.

The first context aware systems were developed in the beginning of the 90s and were led by the desire to use computer systems ubiquitously in a variety of physical environments. The Active Badge project [4] developed by the Olivetti Research Lab was one of the first attempts to adapt applications to people regarding their locations in a building. In the Active Badge location system people carry a small device with them known as Active Badges that emit a unique code for approximately a tenth of a second every 15 seconds. These signals are picked up by a network of sensors that are placed in the host building's rooms and corridors. The sensors' data is processed by a master station also connected to the network that polls the data and makes it available applications and clients.

Other context aware systems were developed later that tried to determine the location of an individual using the context related user location information. These systems

include global positioning systems (GPS) that use location information from satellites and distributed systems that use location information available from underlying communication infrastructures such as GSM and Wi-Fi. In ubiquitous computing, context aware systems must consider a context that is more than a location in order to provide information and services to the user relevant to the user activities, tasks and goals. A special case of a context aware application for example is the *Follow-me* application [46]. The user interfaces of these applications can follow the user as they move using the equipment and resources (e.g. network resources) available as the user moves out of his office. In the *Follow-me* application framework the context not only provides user location but also equipment location and capabilities.

Systems can discover and take advantage of a situation or context such as: location, time and user activity. Context holds a huge amount of information to be addressed as a whole thus we divide the context into 3 main categories or domains based on the information it contains (Table 3-6). These are: the user context domain, the system context domain and the physical environment context domain.

Context Domain	Context Information	Examples
User Context	User identification	User id is George
	User location	User is at point X,Y
	User action/situation	User is walking/sleeping
	User tasks	User has an appointment with the doctor at 10:00am
	Social activity	George is watching the television with Michael
System or ICT Context	Device location	DVD player is in living room MP3 player is on user George Laptop is in Wi-Fi range
	Device properties	PDA has a X by Y screen
	Device status	Set top box is in sleeping mode Awareness of QoS when transmitting messages
Physical Environment Context	Environment constraints	Inside temperature Outside temperature Air quality Light intensity
	Date/Time	18th of September, 2008

Table 3-6: Description of the three context domains

3.5.1 User Context-awareness

The user context domain contains context data such as the user location, user identification, user activities and tasks. With advances in sensors technology user location and identification can be obtained by sensors placed inside buildings (e.g. Active Badges [4] and RFID receivers [47]), while user activities can be obtained by sensors that are located on the user such as accelerometer sensors (wearable computing).

3.5.1.1 User Location and Identification awareness

Harter et al. [48] describe a context aware design for a sensor-driven platform that collects environmental data and exports the data in a form suitable for context-aware applications. Harter's biggest design challenge is to get the user location. Global Positioning Systems (GPS) is not suitable for use inside buildings while electromagnetic methods suffer interference from other devices. Optical systems such as face recognition systems require expensive image processors and detectors and may not work effectively in environments containing many objects and much furniture. To address this challenge they designed BAT, a sensor-driven system that uses ultrasonic techniques to detect the location of the user. In the BAT system, objects and people inside the system's environment are tagged by small wireless transmitters. Each of these transmitters known as *bats* has a unique ID associated with it and consists of a radio transceiver, controlling logic and an ultrasound transducer. Receivers consist of an ultrasound receiver and a serial port interface. They are placed at known places on the ceiling of the rooms and are connected together by a serial wired network to form a matrix. The BAT system consists of a number of mobile and fixed *bats* (wireless transmitters), a matrix of receiver elements, a central RF base station and a computer that does all the data analysis for tracking the transmitters. The RF base station initiates the process by resetting the ultrasound receivers via the wired network. Then it periodically broadcasts a radio message containing a single identifier addressed to each of the bats (transmitters) in turn. When the corresponding bat receives the broadcasted message, emits a short non-encoded pulse of ultrasound. The ultrasound receivers hear the incoming ultrasound and record the time of arrival of each signal from the bat. Finally, the bat-to-receiver distance can be calculated by using the speed of sound and the times-of-flight of each ultrasound pulse transmitted by the bats to three or more receivers (multilateration[3]).

An extension of the BAT system is described in [49] that uses context not only for location awareness but for orientation or state awareness as well. The ORL system requires three or more transmitters to be attached to an object or a person and uses the same ultrasound technique used in the BAT system. The orientation calculation is

[3] When the distance from a transmitter to three or more receivers is known, its position in 3D space can be found using the process of multilateration which is an extension of trilateration.

made by calculating the distance of each of the transmitters placed on the object's body. When the points of the transmitters are known, the orientation of a known object can be determined in 3D space.

The BAT and ORL systems use the ultrasound to measure distances and extract location information from the context. Other systems have been proposed to enable location awareness using Wi-Fi values to estimate location. These fall in two categories. In the first category, are systems that use a deterministic method and in the second category are systems that use a probabilistic approach.

The RADAR (Radio Detection And Ranging) system [50] follows a deterministic approach and implements a location service utilising the information collected from an already existing Wi-Fi infrastructure. It uses signal strength information that is collected from multiple receivers to calculate the distance between the transmitter and each receiver. Then these distances are used to triangulate the user's location. A probabilistic method applying the Bayesian sampling approach is described in [51]. It uses multiple probabilistic models and histograms to find the probabilities over locations calculated from RSSI values. RSSI (Received Signal Strength Indication) values are provided by most wireless Wi-Fi interface cards.

Other user location and identification awareness approaches include:

- Cricket [52], a low-cost decentralised location awareness system that uses RF signals to determine user location.
- Smart Floor [53] where users are identified based on their footstep force profiles. It uses a biometric user identification system collecting information from floor tiles that are fitted with force measuring sensors. The biometric identification system is based on the uniqueness of each person's footstep (e.g. every human walks in a different way).
- The GETA Sandals [54] project, like Smart Floor, identifies the user based on his footstep. RFID tags and accelerometer sensors are attached on sandals and transmit their data to a central computer that extract footstep biometric information based on the accelerometer values.

3.5.1.2 User Task Modelling

User task models specify the tasks that someone performs using an application and how they relate to each other. They capture a user's task and system's behaviour with respect to the task-set to identify what the user does or wants to do and why. User task modelling can be used to enable user interfaces to be adapted to users' own ways of working.

Gaffar, *et. al.* in [55] attempt to build, link and instantiate generic task models by using task patterns. They propose an XML Schema for the specification of task patterns and a mark-up language called Task Pattern Markup Language (TPML) in order to model task patterns into a re-usable manner for HCI frameworks.

Another attempt to generate concrete user interfaces is the KnowiXML [56] project. It is a knowledge based system that models user tasks by applying design rules to abstract models. KnowiXML uses UsiXML[4] [57] to store the user task models and allow them to be reused by user interfaces.

The creation of various task models and the process of linking them to each other is a tedious, time-consuming activity. Current model-based frameworks lack the flexibility of reusing already modelled solutions, while very few approaches offer a simple form of copy-paste reuse. Gaffar's TPML presents a more disciplined approach for modelling patterns to supplement current model-based approaches by facilitating the construction and transformations of models as well as formatting them to encourage reuse. On the other hand, KnowiXML approach defines an interactive model that is easy to learn and (re)use model tasks for helping users in performing their daily tasks. However, the KnowiXML's knowledge acquisition process is performed semi-automatically (information is gathered by HCI designers and experts) resulting in a difficult and time-consuming process.

[4] A user interface markup language (UIML)

3.5.1.3 User Activity Recognition

User activity is another property that can be extracted from the user domain of the context. The challenge in user activity recognition is to identify what activities are undertaken and who is involved.

The Wearable Sensor Badge & Sensor Jacket project [58] proposes an activity recognition system using two types of sensors: accelerometer and knitted stretched sensors. Two accelerometer sensors, one horizontal and one vertical are placed on a belt (Sensor Badge). These sensors can reflect the g-force and the direction of the user's current state when a user wears the belt. Knitted stretched sensors are fitted on a wearable jacket to detect the postures and movements of the user. Seon-Woo Lee et al. in [59] and [60] extended the above user activity recognition techniques by proposing the recognition of walking behaviour through counting steps.

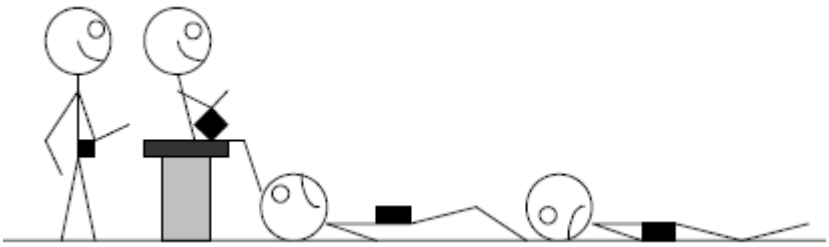


Figure 3-3: Sensor Badge orientation during different user positions (standing, sitting, lying)

Ling Bao et al. [61]argue that with only two sensors, a system is not able to capture user activities. They proposed a system consisting of five biaxial accelerometers worn simultaneously on different parts of a user's body. The results of their research suggest that their sensor model can achieve recognition rates for some activities of over 80% for 20 every day activities (such as walking, running, sitting, climbing stairs) and the user does not have to train the system. Nicky Kern et al. [62] and Kristof Van Laerhoven et al. [64] suggest that a user activity recognition system should include between 12 and 32 3D acceleration sensors attached on the user in order to model the user activity.

3.5.2 Physical Environment Context-awareness

Context Aware applications must extract, interpret and use context information and adapt its functionality to the current context. The challenge for such applications/systems lies in the complexity of *capturing*, *representing*, *managing (e.g., storing and retrieving huge volumes of context information efficiently)* and *processing* contextual data [65], [66].

3.6 Multi-Device Orchestration

Home devices are distributed in the home network as nodes that are able to join or leave at any time at any place in the network. Thus the home network is a Distributed Transient Network or DTN that has the following properties:

- Decentralised network architecture
- Heterogeneity of network node resources
- Self-managing/healing network
- Node discovery

Two well-known distributed transient networks are ad-hoc and the Peer-to-Peer (P2P) computer networks. A P2P network does not include clients or servers but every node in the network is an equal *peer node* that simultaneously acts as both client and server to the other *peer nodes* on the network. An example of a Peer-to-Peer network is any file sharing network such as BitTorrent, Gnutella and Kazza, while a non-P2P network is for example a FTP network between a FTP server and FTP clients.

An ad-hoc network usually is associated with mobile and wireless devices. The connections in that network are established only for the duration of one session and require no central server. Service discovery is implemented in devices so they can discover others within range to form a network with those devices. For example an ad-hoc network can consist of a network of different mobile phones and PDAs that use Bluetooth or Wi-Fi to discover each other and to communicate.

Service Oriented Architectures (SOAs) [17] and their characteristics provide social organisation models such as multi-agent systems (MAS) with the aim to better build

on conventional information technology and do so in a standardized manner so that tools can facilitate the practical development of large-scale systems. Some of the service oriented computing limitations are due to the centralised design of their core services. They provide a single point of failure. For example, if the component that is responsible to discover services (service discovery component) goes offline in the SOA network, then the discovery service will remain unavailable until that component goes online again. A typical life-cycle of a SOA is summarised in Figure 3-4.

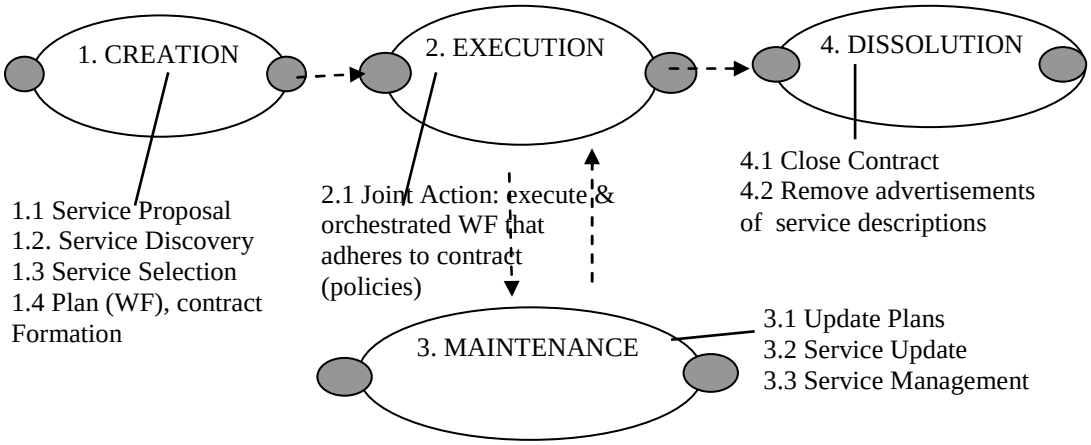


Figure 3-4: Service Oriented Architecture Life Cycle model

In a Service-Oriented Architecture, distributed applications are built by orchestrating or choreographing reusable services using high-level workflows or business processes. The complexity of developing and maintaining these processes is addressed by SOA development cycles that identify the roles of participants at each stage.

In ICT home networks, *orchestration* models include the types of workflow that contain a centralised control system, where a central station manages and defines or updates (in dynamic workflows) the processes of the surrounding devices. Elting et al. [18] and Kray et al. [19] propose an architecture for orchestrating multi-device networks which generates Synchronized Multimedia Integration Language (SMIL) [20] presentations for multi-display environments.

In *Choreography* workflow models, each device includes its own workflow management system. Languages such as the Web services Choreography Description Language (WS-CDL) defined by W3C [21] aims to strictly define observable interactions between services from a global point of view.

3.6.1 Planning

Planning is arguably one of the most important capabilities for an intelligent system to possess. In almost all cases, the tasks which these systems must carry out are expressed as goals to be achieved. Goal-based intelligent agents of these systems must then develop a series of actions designed to achieve this goal.

The ability to plan is closely linked to the agent's representation of the world. Effective planning requires that knowledge of the world is available to the planner agent. Typically, this knowledge contains information about possible actions in the world, which is then used by the planner in constructing a sequence of actions.

Gat's ATLANTIS planner [25] is used for task planning and navigation. The design of the ATLANTIS architecture was based on the observation that there are differing levels of activity in the environment. These levels require different mechanisms for dealing with them, depending on what is important at what level. At some levels, planning might be important, and at others, a quick reaction time. Its architecture is an instance of a layered architecture. There are three main layers: control, sequencing, and deliberation. The planning is done at the deliberation layer by using a symbolic world model; however the planning algorithm used is unclear.

Theo’s [26] architecture uses a global world representation and a search based planner. It stores all of its data into a frame-based system that gives the ability to store a large knowledge base and to access this knowledge efficiently. The frame-based nature of this planner allows it to be tested with different planning algorithms (e.g. STRIPS, linear planner).

In the SOAR [27] architecture, a problem solving process attempts to find a set of operators that lead from a given state to a goal state. This set of operators can be considered as a plan to move the intelligent agent from the initial state to the goal state. If SOAR reaches a conflict or a state where it does not know the next operator,

it forms a sub-goal to choose the next operator. Within this sub-goal it can perform a look-ahead search, simulating the effect of different operators on the current state, in order to determine which operator to apply next.

In goal-oriented management systems, policies are used to constrain how tasks can be partially ordered to lead to a goal-state. Lieberman et al. in [28] approach this challenge by introducing Roadie, a user interface agent that provides intelligent context-sensitive help and assistance for consumer devices. Roadie's main objectives are to provide its user with proactive advice, automate complex tasks and provide debugging help when something goes wrong. To address these objectives, Roadie combines two AI techniques: the Openmind Commonsense knowledge base [29] and a planner. Roadie uses the knowledge queried from the ConceptNet knowledge base to understand a user's actions and goals. Then these actions are mapped into Roadie's planner recogniser called EventNet and sent to planner. The role of planner is to create a set of actions needed to configure or manipulate a specific device satisfying the user's goals, while EventNet presents this set of actions to the user and maps device configuration actions into a format that the device can understand. Roadie's planner is based on the standard Graphplan planning structures [30].

The Graphplan planning approach applies to STRIPS-like domains including a planner that tries to create a shortest possible partial-order plan and if cannot manage it states that a valid plan does not exist. STRIPS [31], [32] is a planner but it also uses a formal language to describe the inputs to this planner and its states: "the initial state", "the goal state" and "operators".

Elting et al. in [33] and Andre et al. in [37] propose two different planners that produce interactive presentations and user interfaces on the basis of an abstract system goal. Using the Open Agent Architecture their systems start from this abstract goal and generate a user interface presentation in real-time using animated presentation agents and speech synthesis.

3.6.2 Rule-based and Policy-based Management of Devices

Policy-based management is needed to constrain the combinations and operation of multiple devices.

In a policy-based system, policies or rules are defined to govern or constrain the behaviour of a system. Policies are executed using a policy engine that analyses trigger events to examine if guard conditions cause further actions to be triggered and to deal with policy prioritisation and conflict when several policies apply. In a reactive policy-based system, policies are triggered solely by incoming trigger events.

3.7 Discussion

In this chapter, research projects have been surveyed that support pervasive device interaction (section 3.1), device interoperability and gateways (section 3.2), device portals (section 3.3), planned activities that span multiple devices (section 3.4), context driven activities (section 3.5) and finally multi-device orchestration (section 3.6).

The general issues have already been analysed in the previous sections. The focus of this section is to justify which solutions are the best to use in designing the proposed framework for this research project.

3.7.1 Message exchange between Services (OSGi/ Web Services)

When devices interoperate for executing a process in combination to provide a common goal, messages should be transferred between the interoperated devices. The two standards for message exchange between different hosts (devices) often used are OSGi RMI/RPC (Remote Method Invocation / Remote Procedure Calls) and Web-Service SOAP/WSDL (Simple Object Application Protocol / Web Service Specification Language).

On top of the OSGi framework, the OSGi Alliance has specified many services. They are specified by a Java interface. OSGi bundles implement this interface and register the service with the Service Registry. Clients of the service can find it in the registry, or react to it when it appears or disappears. OSGi uses the Java RMI API to implement RPCs. Java RMI allows an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine.

This process is similar to the service oriented architecture used by Web services. The main difference between Web services and OSGi framework is that Web services often use the SOAP messages in a WSDL format (including header elements for

transport information and body elements for the actual data to be transferred), which makes it thousands of times slower than OSGi-RMI services that use method invocation. Also, OSGi components can directly react on the appearance and disappearance of services.

Davis and Parashar in [39] evaluate the latency performance between SOAP over HTTP and compare these results with the performance of Java RMI. Their results show that Java RMI is much faster than SOAP protocols (Table 3-7).

System	int[200] Latency (ms)	int[400] Latency (ms)	int[800] Latency (ms)
JavaRMI	2.3	3.2	4.7
CORBA	2.7	4.0	5.5
MS SOAP Toolkit †	33.3	55.8	104.1
SoapRMI	75.0	73.0	108.6
SOAP::Lite	400.3	600.8	1120.1
Apache SOAP	310.3	380.5	561.7
Apache SOAP †	293.7	285.0	449.2
Apache Axis Alpha 3	76.9	138.6	243.7

† Marked results are for XML node lists.

Table 3-7 SOAP – RMI Evaluation - GetIntegers() use between a client and server

Considering the above advantages of OSGi over Web Services, OSGi is preferred as the gateway to access home services in conjunction with the UPnP standard that supports service discovery and selection.

3.7.2 Planned Activities that Span Multiple Devices

Planning activities that span multiple devices require fulfilling two objectives: the planning of user activities and multiple device interoperability.

The ACHE and Cohen's systems attempt to understand user activities regarding their interaction with various home appliances (e.g. adjusting thermostats, turn on devices) while monitoring the physical environment changes. These systems only sense the user activity using interfaces although Cohen's system allows users to interact with

speech and gesture commands. It is also unclear if these systems manage more than one device simultaneously in order to allow device interoperability.

In order to achieve multi device interoperability, a method should exist that will direct the operations of the devices.

Managing multiple devices in an ICT home includes two different management architecture types: centralised management architecture (orchestration) and de-centralised management architecture (choreography). Considering that in a home there are a number of mobile devices that enter or exit the home's network boundaries and also there are devices with limited capabilities, a de-centralised architecture for device management seems to have disadvantages over a centralised architecture.

OSGi and UPnP frameworks surveyed in this thesis, show that they are the state-of-the-art technologies for device interoperability and management of services in the home:

- OSGi allows multi device configuration, synchronising and control.
- OSGi uses Java RMI to invoke RPC processes which is faster than other RPC frameworks.
- UPnP enables devices that enter the home networks boundaries to be discovered and their settings and capabilities to be quoted.

However, OSGi and UPnP do not solve the problem of multi device interaction as there are devices that are not compatible with these two frameworks. UPnP may include device capabilities and settings but this is limited. It is not "open". It has been written by the manufacturer at design time and cannot be changed by the user. It does not contain user profiles (user personalised information to adapt that device.

ATLANTIS, Theo and SOAR surveyed in this chapter are planning architectures that use different planner to lead activities from a given state to a goal state. ATLANTIS uses a symbolic representation of it environment, while Theo attempts to plan its actions using a global world representation. SOAR requires a global knowledge of the world is available to the planner and it limits its use in the home since most ICT homes are reasonably complex environments.

Among the other planning models and planner projects surveyed, Roadie was designed to assist user interaction with home devices and appliances. Although

Roadie provides context-sensitive device management, it does not sense the home environment (context) and user activities. It requires its users to input any preferred action (goal) and it provides them with proactive advice.

3.7.3 Context-Driven Activities

There are various solutions available today for user or object tracking and location discovery. These include active or passive electromagnetic trackers, optical trackers, ultrasound trackers and RF trilateration algorithms. In the systems surveyed, the BAT and Active Badge systems are centralised architectures while Cricket is decentralised. These systems use RF and ultrasound techniques to find user location but may introduce privacy concerns.

Active or passive electromagnetic and optical trackers are expensive and their performance is affected by the presence of metallic or magnetic obstacles in the environment. Wi-Fi signal strength techniques may not be very accurate because of obstacles and interference from other networks and devices. Ultrasound tracking techniques proposed in the BAT project are immune to obstacles and network interference but require the use of special equipment such as ultrasound transmitters and receivers. Furthermore, ultrasound tracking techniques require the synchronisation of the transmitters and receivers and it makes this technique more complex. The Smart Floor project uses an expensive infrastructure because it requires custom made floors and floor tiles. User identification requires users to wear an ID track sensors such as RFID. Location tracking can be performed by comparing Wi-Fi signal strengths.

The main problem with wireless (Wi-Fi) location detection is mainly attenuation, change of environments and different radio strengths used by manufacturers.

There are ways to find people's location, state (actions) and identification (e.g. who is the person sitting on the sofa) but how can this information be used? How can the goals of the user that performs the actions be predicted?

Key challenges in modelling the user context are:

- User context may be incorrectly or incompletely determined. For example, a user location system may provide inaccurate data to exactly locate the user.

- Difficulty to determine the user location indoors. Traditional GPS techniques do not work inside buildings. Other techniques such as location detection based on imaging (face recognition) are expensive and need high computer resources.

4 Framework

This section specifies a framework to support the execution of planned user tasks and unplanned user tasks situated in a smart home environment. Two main models of systems are designed: a general life-cycle operational model for creating and maintaining services (Section 4.3, 4.4) and service oriented architecture whose service components will support planned tasks and unplanned tasks (Section 4.5, 4.6, 4.7).

4.1 Iterative System Development

The service oriented architecture model has been constructed in phases. The four main phases of system development to-date are as follows.

System 1: user device portal that supports dynamic device discovery and orchestration. The system communicates with devices using the OSGI model (Section 4.4.1)

System 2: adds support for an OSGi gateway, to interface to devices, and supports a planner system, to execute tasks that span multiple devices (Section 4.4.2).

System 3: adds support for context-based awareness and simple context-based task initiation (Section 4.6).

System 4: a framework to support a complete SOA lifecycle model. It orchestrates the lifecycle of services using policies and workflows (Section 4.7).

4.2 Service Architecture Overview

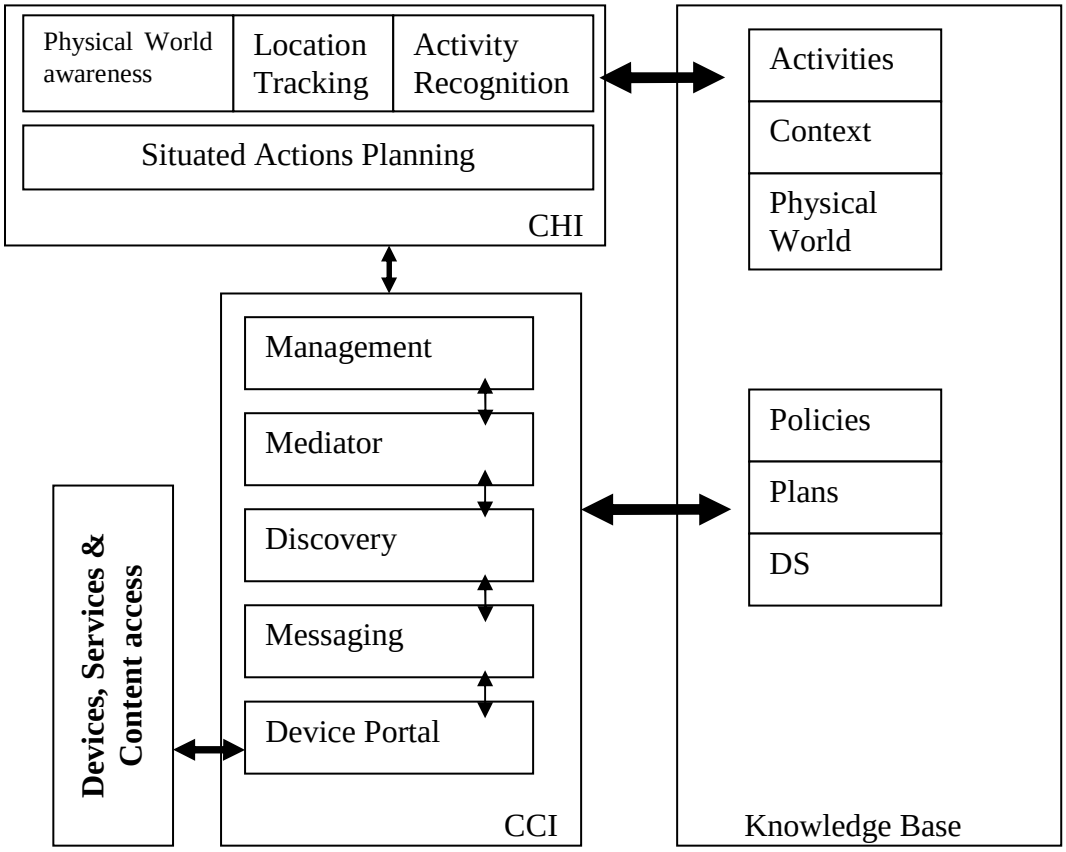


Figure 4-1: Generic System Architecture

The system architecture is shown in Figure 4-1. The main parts of this model are the five layers used between the input content and the Knowledge Base that act as the middleware. The four bottom layers are part of the CCI. At the bottom the Messaging layer allows information sharing and exchange between services. Above it, the Discovery layer searches, finds and registers any new services found in the environment. The Mediator layer enables service and content interaction. The Management layer at the top manages the other layers using predefined rules and policies. At the top of the CCI layers there is the CHI layer which acquires knowledge from user activities and context. Policies, Plans and Directory Services interact only with the CCI layers. A vertical layer called the device, services and content component represents the application specific versions of these that are interfaced to the system using device gateways such as an OGSi device gateway.

4.3 Core system

4.3.1 OSGi Gateway

The home environment to OSGi gateway relation is shown in Figure 4-2. It is clearly shown that OSGi can *read* from every domain in the home environment (Users, Devices, and Physical world) but it can only *write* (control) the device domain. In some cases, changing the status of a device in the service domain may affect the properties of another domain. For example, a change of the heating service thermostat status will probably affect the internal temperature.

The home environment domain includes a variety of sensors that can take physical environment readings such as temperature and humidity.

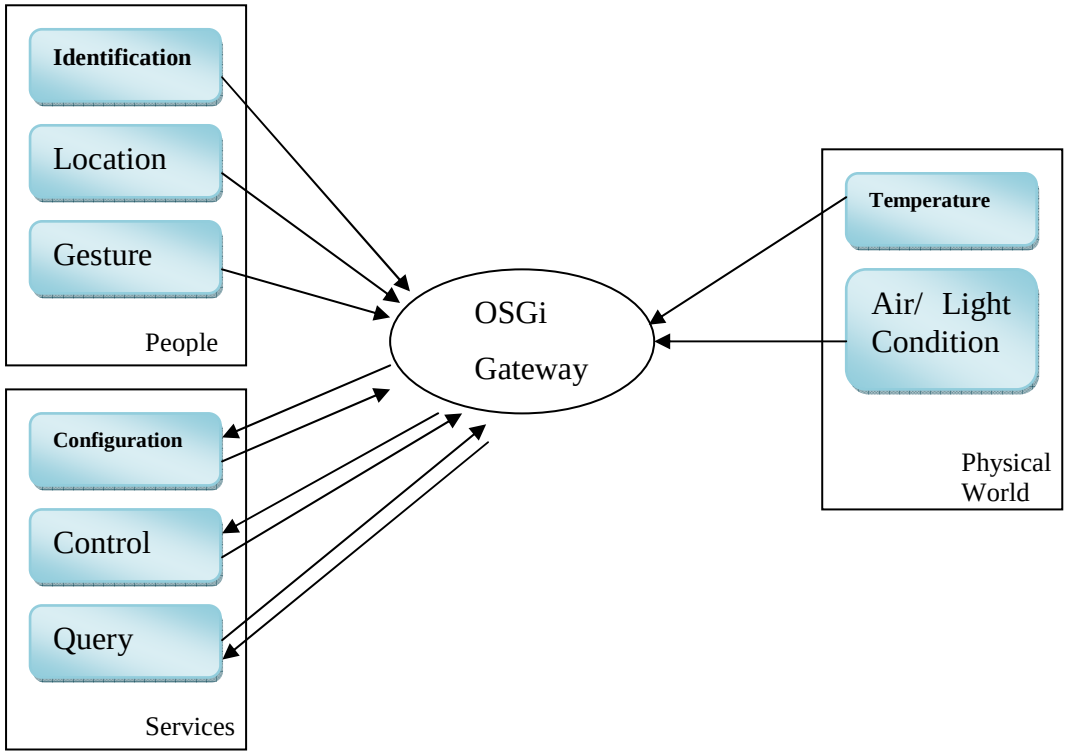


Figure 4-2: Home Environment domain to OSGi Gateway domain Interaction

4.3.2 Service Discovery and Addition

The life of a service in the home environment begins from the moment that a device or devices that produce a service are configured and start to operate in an environment. In order for the service to exist and be registered to the service registry, each of the required devices should be discovered and configured. The device

discovery and configuration is driven by the UPnP framework mechanisms. When a device enters the home environment, UPnP queries its capabilities and when the discovered capabilities meet a service's required capabilities then UPnP informs the service registrant that the service is in a ready state.

4.4 User Device Portal

Two applications have been built to demonstrate the two different CCI framework designs. The first framework supports service combination and the second framework supports service discovery, combination and dynamic interfaces. These are based upon the OSGI framework.

4.4.1 Device Orchestration (Prototype 1)

In this prototype we have built an application using web services and the OSGi framework. The home services are presented in an OSGi gateway as bundles while the CCI framework runs as a J2EE application on our web server (Figure 4-3).

New services and their capabilities are discovered by the service discovery OSGi bundle and each new service is then installed on the OSGi gateway. The web application queries the installed services and presents the available services to the user who defines service combinations. A demonstrator implementation can combine two services. The combination process to do this is as follows:

- There are two empty slots (slot 1 and slot 2) for the user to select the two different services to combine.
- The user may select a service to add to slot 1 from a list of discovered services. The list only contains services that their capabilities and configuration allows them to be combined with other services.
- When the slot 1 service is selected the second slot (slot 2) is enabled to allow the user to select the service to be combined. Slot 2 only lists the services whose capabilities and configuration allow them to connect to slot 1's service.
- When the slot 2 is filled up with the desired service a simple interface is presented to the user to let him configure the new combined service.

- The user now has two options. One is to install and start the new (combined) service in the OSGi gateway and the second option is to save the configuration for the two services.
- Finally, when the user triggers the installation and execution of services, the web application sends the configuration information to the OSGi discovery bundle and a new bundle is created, installed and starts running.

OSGi service discovery bundle holds a list of existing services with their capabilities in an XML-based (W3C XML schema) data source. This data source (e.g. services.xsd) has been created manually and contains the services as elements with their settings and capabilities as shown on the next XML file listing:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:serviceVO="http://www.bt.com/osgi/xmlvos/base"
targetNamespace="http://www.bt.com/osgi/xmlvos/base" elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:complexType name="service">
    <xs:sequence>
      <xs:element name="serviceName" type="xs:string"/>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="serialNo" type="xs:string"/>
      <xs:element name="compatibleServices" type="serviceVO:KNOWN_SERVICES"
minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="interfaceClass" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="KNOWN_SERVICES">
     <xs:restriction base="xs:string">
       <xs:enumeration value="timer"/>
       <xs:enumeration value="light"/>
       <xs:enumeration value="camera"/>
     </xs:restriction>
  </xs:simpleType>
  <xs:element name="serviceDiscovered">
    <xs:complexType>
      <xs:all>
        <xs:element name="SERVICE_DISCOVERY_INFO" type="serviceVO:service"
minOccurs="0" maxOccurs="1"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

From the above service discovery XSD file, the discovery mechanism accepts configuration data from a user to generate the following XML data that is used by the system to understand the existing services, functions and interoperability between them.

```
<?xml version="1.0" encoding="UTF-8"?>
<serviceVO:serviceDiscovered xsi:schemaLocation="http://www.bt.com/osgi/xmlvos/base
Untitled2.xml" xmlns:serviceVO="http://www.bt.com/osgi/xmlvos/base"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <serviceVO:SERVICE_DISCOVERY_INFO>
    <serviceVO:serviceName>timer</serviceVO:serviceName>
    <serviceVO:description>This is a simple timer</serviceVO:description>
    <serviceVO:serialNo>1234567890</serviceVO:serialNo>
    <serviceVO:compatibleServices>light</serviceVO:compatibleServices>
    <serviceVO:interfaceClass>TimerInterface.class</serviceVO:interfaceClass>
```

```
  </serviceVO:SERVICE_DISCOVERY_INFO>
  <serviceVO:SERVICE_DISCOVERY_INFO>
    <serviceVO:serviceName>light</serviceVO:serviceName>
    <serviceVO:description>This is a simple light control</serviceVO:description>
    <serviceVO:serialNo>1111</serviceVO:serialNo>
    <serviceVO:compatibleServices>timer</serviceVO:compatibleServices>
    <serviceVO:interfaceClass>LightInterface.class</serviceVO:interfaceClass>
  </serviceVO:SERVICE_DISCOVERY_INFO>
</serviceVO:serviceDiscovered>
```

After loading this XML service configuration, the system knows what service combinations it can accept. In this scenario, the timer is compatible with the light service. Also this XML provides the Java Interface class that the system should use in order to load specific user interface and control operations for each service (e.g. LightInterface.class).

The next task of the system is to load each of the services Java Interface class and also to load and initialise each OSGi service bundle. When the OSGi bundles have been initialised can exchange control messages (as they are compatible).

In a later version of this prototype, an attempt is made to make the interface simpler to use by non-expert user. To enable, users can specify how to combine services using plain English text. The application only understands a number of words that the user inputs as text as the vocabulary we use is a simple one. A system that understands user input text and translates it into service configuration data may be possible using common sense logic techniques but this is out of the scope of this project.

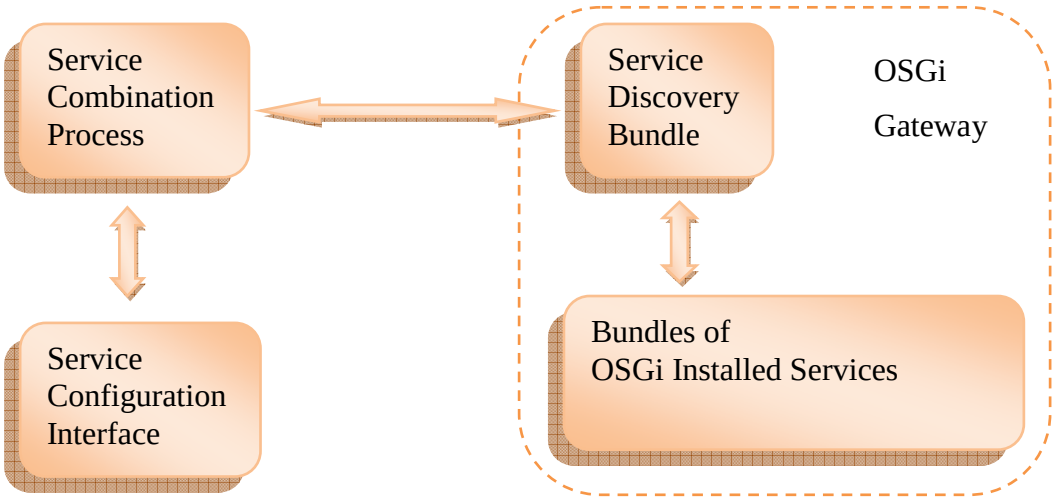


Figure 4-3: Process diagram of Demonstrator 1

4.4.1.1 Evaluation of Demonstrator 1

Screenshots of the first demonstrator can be found in Appendix. In the following three examples a typical run of the demonstrator is evaluated.

Example 1: A user wants to combine two services. In that case service 1 is the home lights service and service 2 is a timer. The simple lights service demonstrated here has two states: set the lights on or off. When the two services are selected the service combination process queries the capabilities of each service and the configuration interface adapts to these capabilities by displaying a configuration interface. The user selects the time when the lights should be turned on and the time for the lights to turn off.

Example 2: In this example a service is selected where its capabilities does not allow it to be linked with another service.

Example 3: In this last example the user inputs service configuration information as plain text. A configuration a user may input is: “turn on the lights at 8:25 and turn off the lights at 12:32”. When the string is set the configuration interface process extracts the actions (e.g. turn on the lights at 8:25) and adapts a device accordingly.

This demonstrator has a number of limitations that are listed below:

- Allows only two services to be linked.
- Configuration interfaces for each service are not dynamically generated but are hard coded into applications.
- The user must input configuration strings in a standard format for the web application to understand it. A number of errors occurred when configuration text is input using a slightly different syntax. In a real world situation each person has a different way to input configuration information using plain English.

Some of the above limitations have been addressed and solved in the next demonstrator.

4.4.2 Device Discovery (Prototype 2)

The second demonstrator focuses on service discovery and HCI. It is a Java application that lets the user link two or more services into one service and is configured using a dynamic interface.

As in the first demonstrator the application queries the OSGi gateway for installed services and their capabilities. The new services are listed and the user may select which of these is installed. A list of the installed services shows the service name and lets the user pick services to be combined. In this second demonstrator the limitation of the first demonstrator where only two services could be combined, have been solved. Now the user can combine two or more services together. The main components of the Demonstrator 2 framework are shown in Figure 4-4.

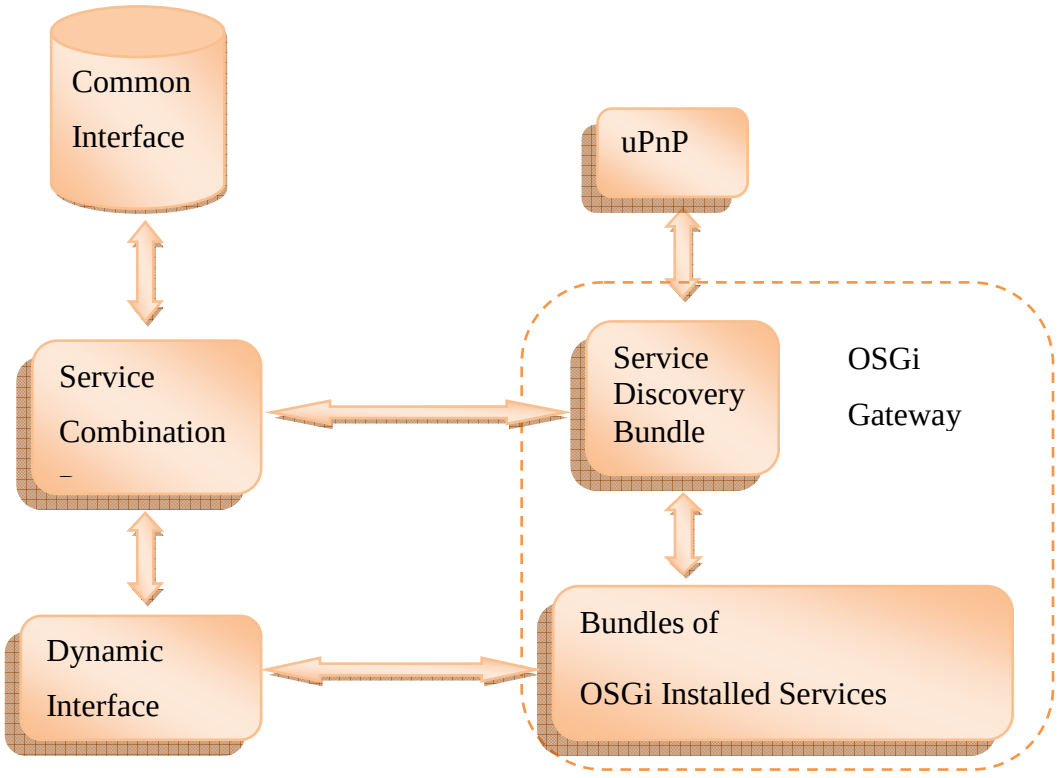


Figure 4-4: Main components of the Demonstrator 2

When the user picks services for combination a dynamic user interface presents configuration controls for the combined service. There are two places where these controls can be found:

- In devices. Some devices have interface information in their UPnP (universal Plug and Play) configuration data
- In a database: controls for the operation of common and typical services are stored in XML format.

Finally, when the interface of the combined service has been set up, the user can use it to control the two or more services as one.

The evaluation and examples for Demonstrator 2 are placed in the Appendix.

Home devices, mobile devices and sensors connect and interoperate in the home's environment based on the OSGi specification. Devices and sensors produce one or more services (bundles in OSGi terminology) that are discovered by the UPnP framework. They are registered in the OSGi service registry. The OSGi gateway manages each service's life-cycle as instructed by the planner (Figure 4-5).

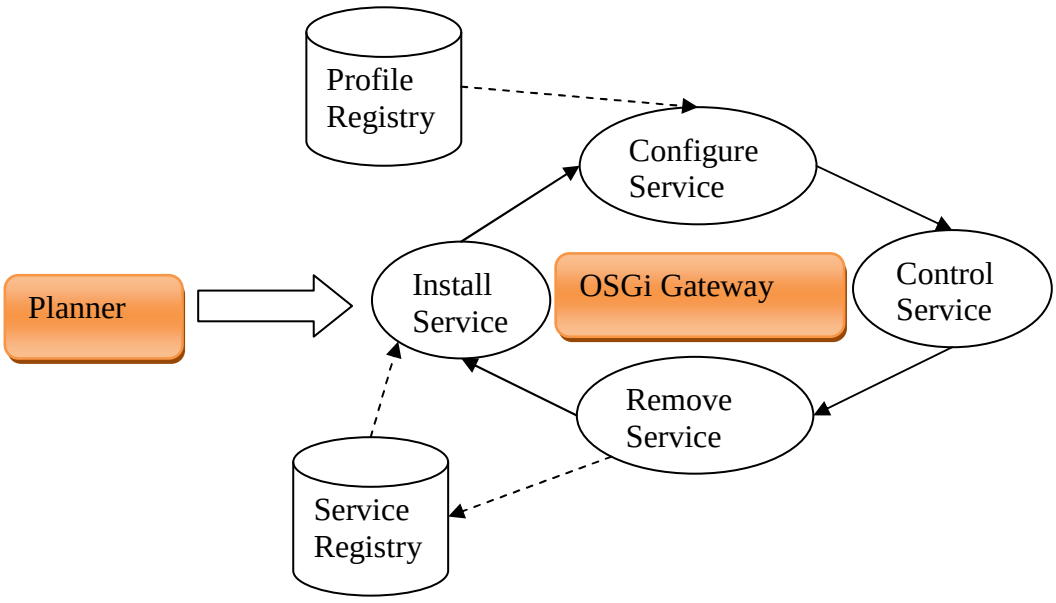


Figure 4-5: The OSGi gateway system architecture

Each service's life-cycle includes four main processes:

- **Install service process.** This process searches for any already installed and active services that match service descriptions sent by the planner to the service registry. If a suitable service is already active it gets its reference. If no suitable services are found, the process triggers the UPnP framework to query for new services. If a service is found then, it is installed as a bundle in the OSGi framework.
- **Configure service process.** The next process of the OSGi framework is the configuration of the service. This process configures the selected service based on users' preferences (user profiles) and resolves any conflicts that may occur by its installation.
- **Control service process.** This is the process that manages a service's operations. Service operations take place in one or more device hardware and include commands such as turn on, set and get methods, etc.
- **Remove service process.** The last process in the OSGi framework lifecycle is the disposal of an inactive service when it is not needed anymore or when the service has become unavailable (e.g. when a PDA providing the service has been moved out of home network coverage). This process removes the reference of the service from the service registry and resolves any conflicts that may occur.

4.5 User Tasks or Activity Planner

The proposed system architecture is presented in this section. An abstract model is presented in Figure 4-6 to introduce the main components of the

framework.

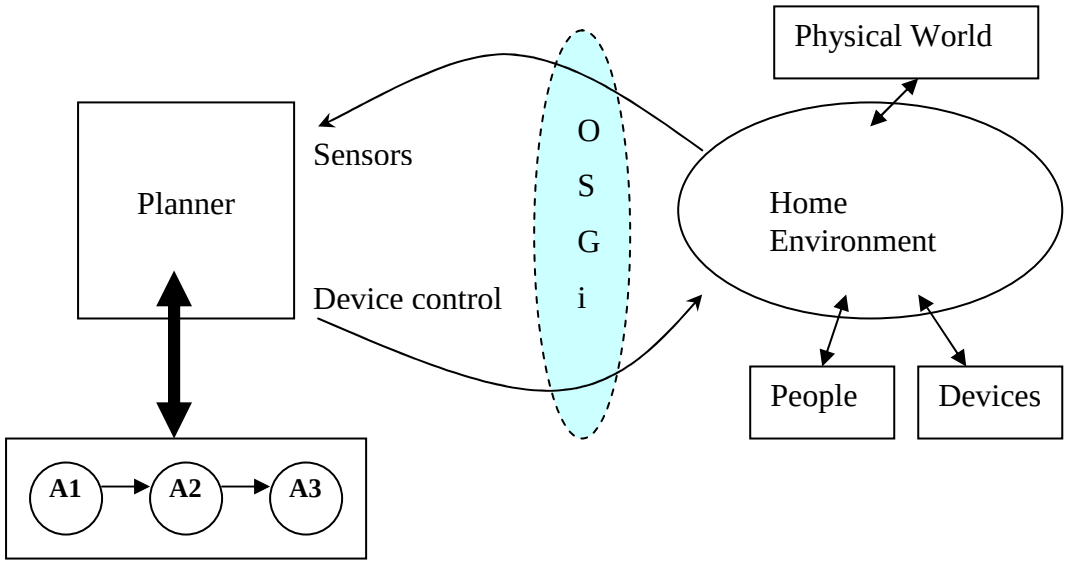


Figure 4-6: General view of the proposed framework

In this abstract view of the system architecture the three main domains are: the Home Environment, the OSGi gateway and the Planner. The *Home Environment* includes the context that is all the information that is available and can be used to feed the *planner*. Home environment includes humans who are placed inside the physical home environment, devices either situated in certain places in the house or mobile devices such as PDAs, mobile phones and laptops. Also, the home environment includes physical world variables and constraints: house and room boundaries, current minimum and maximum room temperature, quality of air, daylight and outside temperature is an example of these.

The *OSGi gateway* acts as a device access control point and as middleware between the devices and the planner. As a device access control point, the OSGi gateway allows device and service interoperability. Devices built based on OSGi standards can be connected, registered, configured, interact with other devices or services and uninstalled if necessary. As a home environment to planner middleware, the OSGi gateway's role is to enable the planner to have access to context from sensors readings, user ids, user location information and device context (e.g. the light in room 2 is on, the TV is on channel 4, etc.). Also, the OSGi gateway accepts device control messages from the planner and changes the device configuration or controls the device accordingly as instructed by the planner in order to reach a specific goal.

The planner continually senses the home environment. When a user goal is identified, the planner creates a plan, which is a set of actions that must be performed either by the devices or the home users, in order to reach this goal. Each plan action is a different device control message that is sent to the corresponding device through the OSGi gateway.

4.5.1 Planner System Design

There are two main challenges for the proposed planner framework. The first challenge is to identify the user goal following the user activity and the second challenge is to create a plan that manages a number of services in order to reach the user goal.

Figure 4-7 presents a diagram showing the flow of information between the main proposed planner components. The user produces some actions (user activity) that are captured by the home sensors. The user actions are modelled and are compared with predefined action models until the goal is identified (see next section). When the goal has been identified, a forward chain planner creates a plan to reach that goal considering the state of the home environment (context). Finally, the plan actions are sent to the OSGi gateway that takes care of the service management.

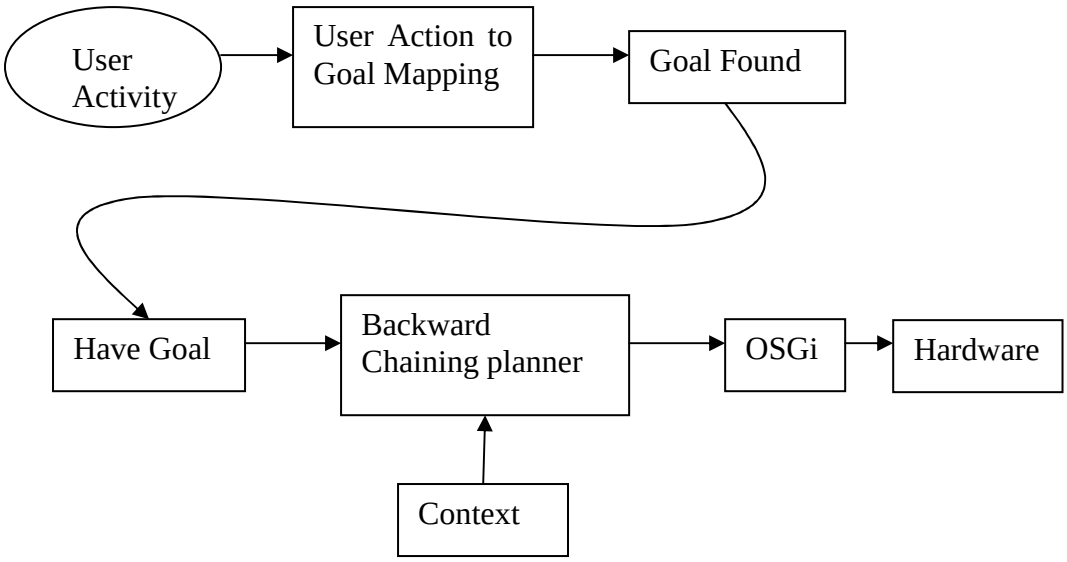
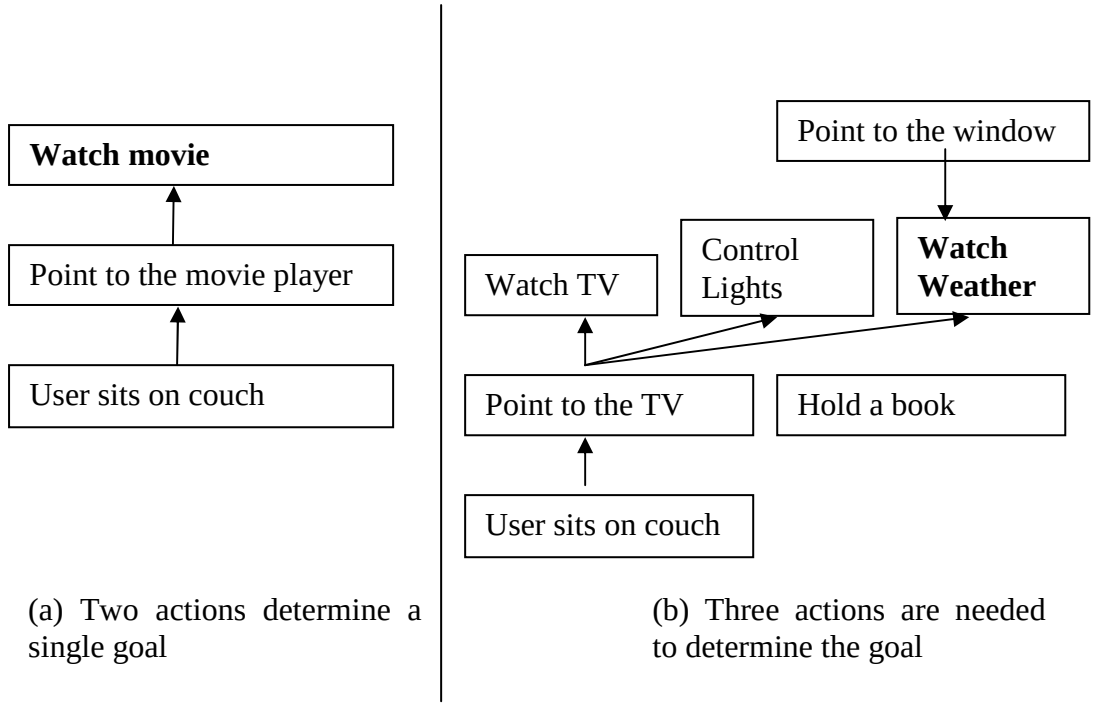


Figure 4-7: The two phases in proposed planner framework: Goal Identification and Plan Creation.

4.5.1.1 Phase 1- Goal Identification

User activities are sensed by a number of sensors located in the house environment and also carried by the user (e.g. RFID tags, Sun SPOTs, etc.). When the user performs an action, this action is captured and compared with the set of predefined actions from an action database. If this action is mapped to only one goal, then this goal description is sent to the Phase 2 of the planner framework. If an action cannot determine a goal, or the action determines more than one goal, a user is asked to select a goal out of a number of determined goals.



(a) Two actions determine a single goal

(b) Three actions are needed to determine the goal

Figure 4-8: The structure of a Hierarchical Goal Determination Process based on action mapping. (a) Action maps to a single goal, (b) action maps to multiple goals until a third action is performed.

In Figure 4-8, the goal "Watch movie" in (a) consists of the simple sequence of two actions: "sit on couch" and "point to the movie player". The action "sit on couch" may lead to a number of possible actions such as watch movie, listen to the radio, read a book, just relax, make a phone call, etc. These actions are not listed on the diagram of Figure 4-8 in order to simplify the diagram.

In (b) the goal determination process is more complicated because on the second user action: "point to the TV" the action-to-goal structure cannot map to a single goal and needs another user action to identify the goal. The third action: "Points to window" triggers the goal determination process to identify the goal as "watch the weather forecast".

4.5.1.2 Phase 2- Plan Creation

The plan creation and execution takes place inside a rule engine mechanism. Rule engines implement a forward, backward or bi-directional chaining mechanism.

Forward chaining: When one or more conditions are shared between rules, they are considered "chained." Chaining refers to sharing conditions between rules, so that the same condition is evaluated once for all rules.

Backward chaining: Backward chaining is very similar to forward chaining with one difference. Backward chaining engines query for new facts, whereas forward chaining relies on the application asserting facts to the rule engine. Backward chaining rule engines implicitly create sub-goals and use those sub-goals to execute queries.

Bi-directional chaining: When rule processing operates in both modes, it is considered to be bi-directional.

In this proposed planner framework version, a number of predefined plans have been modelled and stored. Each of these plan models contains a number of paths and nodes with service control information (e.g. service LIGHTS_ROOM1:TURN_ON).

Each node can have two states: satisfied or unsatisfied based on the outcome of its control information and service availability. When a node of a predefined plan can be satisfied then the route to this node and the node is copied to the current (new) plan. If a node of the predefined plan cannot be satisfied then the planner mechanism moves to the next node. The following example describes this planning process.

A goal is identified from the previous process and a predefined plan (see Figure 4-9) is queried from the predefined plan database. The planner mechanism moves to the first node of the predefined plan (Service 1). This service is available thus this node can be set to a satisfied state. It is copied to the current plan and the planner mechanism moves to the next set of nodes (Service 3, Service 4). Service 3 is not available and returns an unsatisfied state result. The planner moves to the next node (Service 4) which is available and is copied to the current plan. This process continues until all the nodes of the predefined plan have been checked and a new plan, the current plan, has been created. The created plan then is sent to the OSGi gateway that will process every plan’s node in order to manage the plan’s services based on the plan.

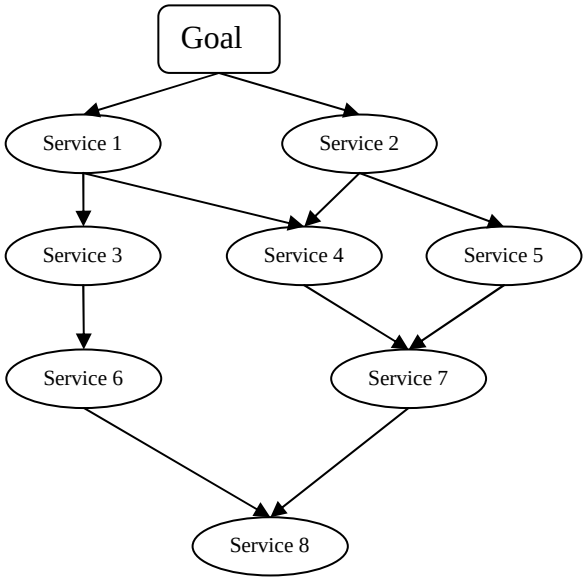


Figure 4-9: Forward chain plan

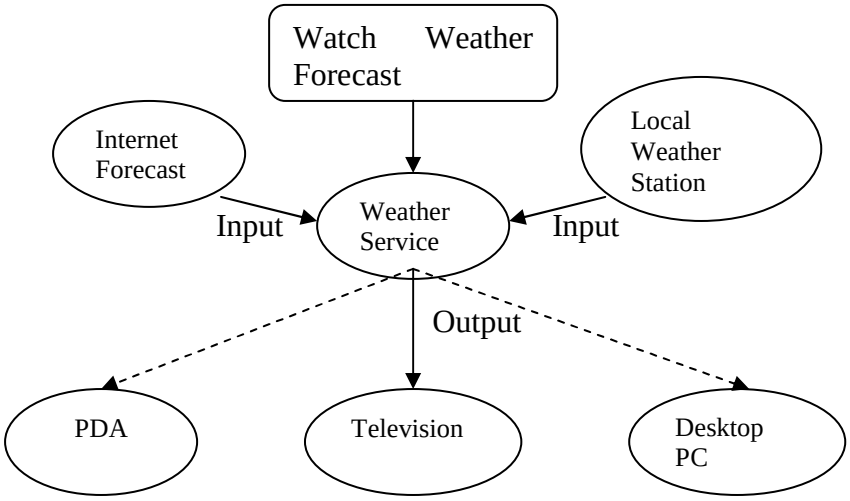


Figure 4-10: Forward chain plan for the Watch Weather Forecast goal

Figure 4-10 shows the plan created based on the activity-to-goal mapping of Figure 4-8 (b). The goal "Watch Weather Forecast" triggers the weather service to take weather information from the Internet and from the local weather sensors. The weather service formats this information and selects Television as the output method for the weather data.

4.6 Context-Awareness

In this research project the user context domain extraction focuses on the following user information:

- **User identification.** Who is the user that performs the activity?
- **User location.** Where is the user?
- **User state.** What is the user activity status? E.g. sleeping, walking, running, sitting.

Different types of sensors are needed for context extraction. In this proposed user context extraction methodology the Sun SPOT sensor platform [3] is selected.

Sun SPOT is a small wireless-enabled devices based on a 32 bit ARM-7 CPU and an 11 channel 2.4 GHz radio. They consist of a board that includes a number of sensors such as 3-axis accelerometer, temperature and light sensors. The device is programmable in Java. Developers can write Sun SPOT applications that get sensor information, perform some pre or post processing in the Sun SPOT platform and then send this information to a base station for further processing. Sun SPOTs also consist of 9 I/O pin port that enables peripheral sensors to be connected to the Sun SPOT. These characteristics make the Sun SPOTs a suitable choice for a sensor network for context extraction.

4.6.1 Indoor User Location Determination using RSSI

Location determination is very important for an infrastructure of a ubiquitous computing environment such as a smart home. Outdoor location determination technology such as GPS is already developed and is widely used. For indoor location determination, there are several technologies (Section 3.5.1.1) but they are not widely used.

Assuming that home users wear or carry Sun SPOTs, their location can be determined by triangulating the signal strength of their Sun SPOT and each of the base stations that are located in the room. Figure 4-11 shows the experiment environment where the experiments of location determination using the signal strength took place. Three base stations were placed in certain room positions. A person holds a Sun SPOT and

follows a path as shown in Figure 4-11. A person takes 50cm steps. After each step the Sun SPOT broadcasts a signal that is collected by the three base stations.

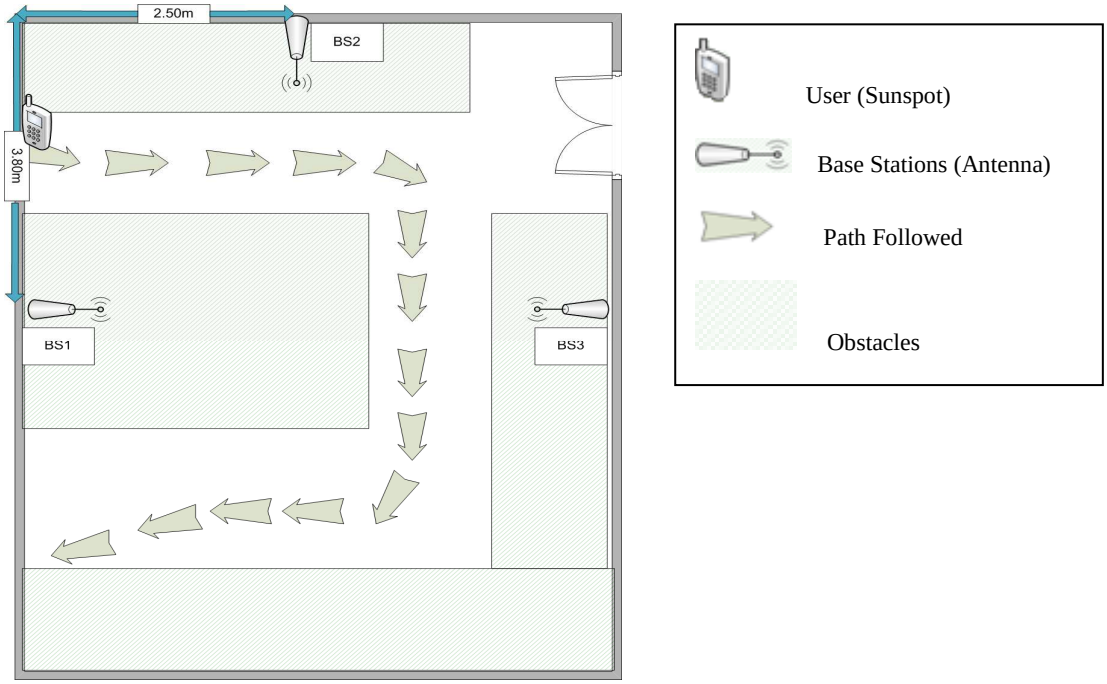


Figure 4-11: Experiment environment floor plan showing the base station position and the path followed by the Sun SPOT

The Sun SPOT platform upon receiving a radio data packet, can extract the signal strength information of that packet. The Received Signal Strength Indication (RSSI) value shows how strong or weak the radio signal is. The results collected by this experiment are presented on the next graph (Figure 4-12).

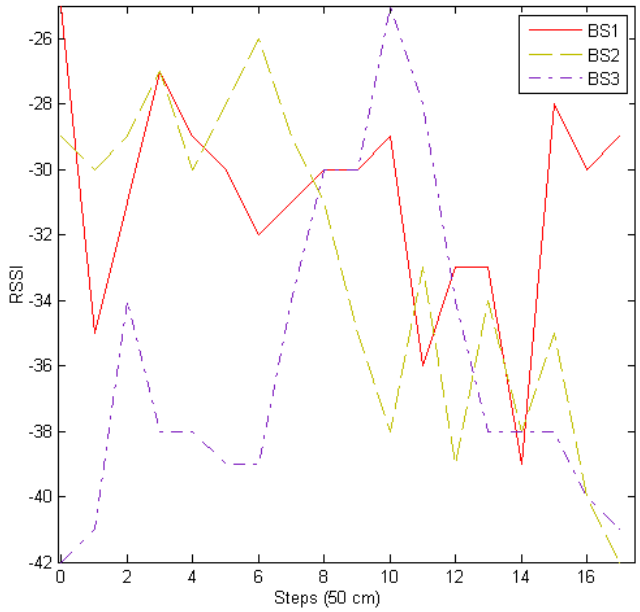


Figure 4-12: RSSI values collected from the 3 Base Stations

In order to calculate distances from the extracted RSSI values, a free space propagation model was used. This model assumes the ideal propagation condition that there is a line-of-sight path between the transmitter and the receiver (Sun SPOT and Base Station). It does not consider any effect of multipath fading and other signal strength loss. In indoor environments, where line-of-sight is not always available, multiple objects and humans may alter RSSI values, a free space model is not precise but it is still appropriate to determine distances if there are many base stations in the space.

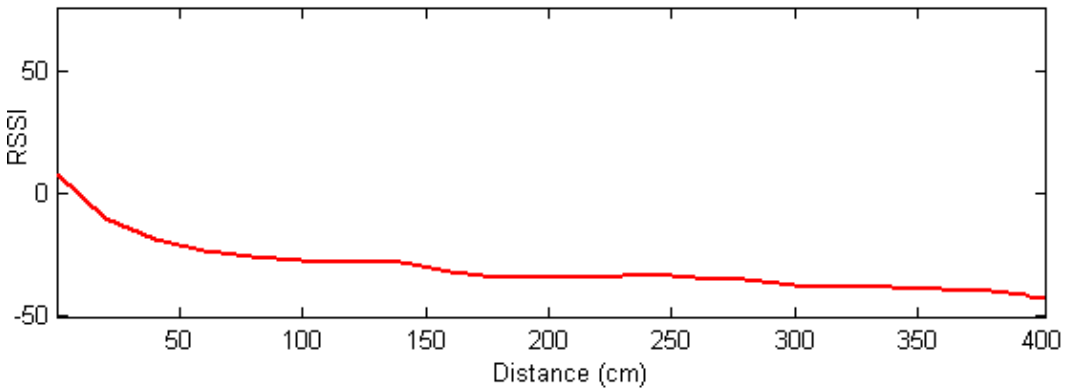


Figure 4-13: Relation between RSSI and Distance on the free space propagation model

In the free space propagation model the relation between RSSI and distance is measured (Figure 4-13) and is described as:

$$P_r(d) = P_0 - 20\log_{10}\left(\frac{4\pi d}{\lambda}\right)[dBm]$$

Where:

$P_0 =$ Empirical constant

$$\lambda = \frac{c}{f} = \frac{3\times10^8[m/s]}{2.4[GHz]}$$

$P_0 =$ 50 dB

Further experiments with RSSI values and combinations with different techniques and methodologies for extracting the position of an object in indoor environments led to a joint project [63] that better determines the user location and also the orientation of a user in an indoor environment.

4.6.2 Indoor User Mobility and Activity Determination Using Accelerometer Sensors

RSSI measurements during the location awareness experiments show that RSSI location determination techniques give accurate values in open space environments when no people and objects are between the transmitters and receivers. However, in more complex environments containing multiple objects, obstacles and people such as home environments, radio signals can get reflected or deflected by obstacle objects making location tracking inaccurate.

As humans move in and between house rooms, they produce acceleration which can be used to calculate position and to track them. Obtaining this information can be done through the use of three-axis accelerometer sensors that are attached on the body that needs tracking. Signals obtained by accelerometer sensors require pre and post processing as there is no direct conversion between acceleration and position.

Acceleration a is the rate of change of the velocity v of an object. Also, the velocity is the rate of change of the position s of the same object:

$$a = \frac{dv}{dt}$$

$$v = \frac{ds}{dt}$$

The velocity is the derivative of the position and the acceleration is the derivative of the velocity.

$$a = \frac{d(ds)}{dt^2}$$

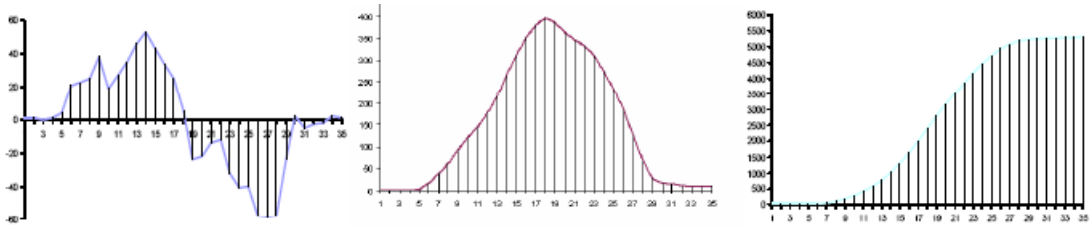
When the acceleration of an object is known, the position of this object at each time can be measured by applying a double integration:

$$a = \frac{dv}{dt} \rightarrow v = \int (a)dt$$

$$v = \frac{ds}{dt} \rightarrow s = \int (v)dt$$

Thus:

$$s = \int \left(\int (a)dt \right) dt$$



a. acceleration b. velocity c. position

Figure 4-14; a. acceleration: actual acceleration data as obtained using an accelerometer sensor, b. velocity: velocity after single integration, c. position: distance travelled from zero after a double integration

Figure 4-14 shows the accelerometer data of an object that accelerates and decelerates (a), its calculated velocity (b) and its calculated position (c) from a zero starting point.

An implementation of the double integration algorithm is used to process the obtained accelerometer data in order to apply the algorithm in a real world scenario. This processing of accelerometer data includes calibration and filtering.

Filtering is needed as accelerometer sensors may acquire noisy data. A digital filter is applied based on a moving average: the average value of a certain number of samples is processed.

Calibration also takes place before the processing of the accelerometer values (pre-processing). The calibration is needed to eliminate the sensor offsets at the beginning of a measurement when there is no movement. In other words, during the calibration, the algorithm finds the sensor values when the object is in a stand still position. After the calibration takes place, the real acceleration data is the data obtained by the sensors minus the calibration offset.

Experiments from section 4.6.1 and this section (4.6.2) led on a better position determination system that adapts to both dynamic physical environmental conditions and human movement changes in order to find estimated user locations and their orientation.

4.6.3 User Identification

Other information that can be extracted from the user context is the identity of the user. The identity of the user can provide information to a number of different applications in the smart home, in which, a user can be identified by the context and these application adapt to the user, e.g. only the adults of the family have access to the house garage; healthcare applications can sense who the user that performs an action is such as falling down and where this occurs and store this information or raise an alarm.

In this project a simple user identification algorithm identifies the user by the way he walks in the house. Gait biometrics can produce an accurate classification if the data is obtained accurately for a small group. The gait recognition algorithm proposed in this project uses accelerometer data along a 3-axis. It models the gait of each person with respect to his weight, step frequency and step speed.

Each user is equipped with an accelerometer sensor (e.g. Sun SPOT). A user spends some time to teach the algorithm by performing everyday activities in the home. Then the algorithm models the user's gait and stores it in a gait biometric database. When a user starts walking in the house, the identification algorithm compares its gait with the already stored biometric information and finds the best match.

4.6.4 User Status Recognition

The user activity level can also be extracted from the user context. The term "user activity level" in this project includes a number of actions that show how active is the user in the home, e.g. sleeping, relaxing, walking in a hurry, etc.

The user activity level recognition algorithm determines the user state by sensing accelerometer values from sensors attached to the user (e.g. accelerometer values change rapidly may indicate that the user is walking in a hurry). Through sensing the device context domain for changes in devices (e.g. the volume of the TV recently changed means that the user is still watching TV and has not fallen asleep).

4.7 Policy-based and Workflow-based Management of Devices

In this framework, service oriented life-cycles of devices and services in a home network are supported by workflows. This framework is placed between the device application frameworks and the OSGi messaging framework. This way the OSGi gateway is extended by enabling device and service orchestration using dynamic workflows adapted to the current home environment (context) using rules and policies. Rules are static and pre-defined set of restrictions or commands while policies are dynamic rules. Policies may be changed by users or other services during the service life-cycle in order to adopt with the current context.

4.7.1 Framework Requirements

The requirements of this prototype are described in Table 4-1.

Requirement	Description
Device discovery	Devices that enter the home network are discovered and their capabilities and settings are made available to the system
Device selection	Based on the device capabilities and settings, a device is selected if it is needed by a service
Device registration	After device selection, the device must be registered with the ICT home service directory. It is configured based on the current system configuration and context
Service selection and registration	A service is complete and can be registered only when the required devices for that specific service have been registered. During service registration, the user is asked if a service can be registered or dropped. Then devices that support these services registered are re-configured (on demand) based on the service configuration demands
Service execution - orchestration	After service registration, a service execution workflow is loaded that is specific for each service. The service execution workflow (jobflow) supports the execution life-cycle of the service by orchestrating the devices registered with it using sets of rules and policies.

Table 4-1 System requirements of System 4

4.7.2 Architecture Overview

This framework consists of three main workflow processes that support the installation and maintenance life-cycles of the SOA architecture and are displayed in Figure 4-15. The installation life-cycle is supported by the “device registration” and “service registration” processes and the maintenance life-cycle is supported by the “execution” process.

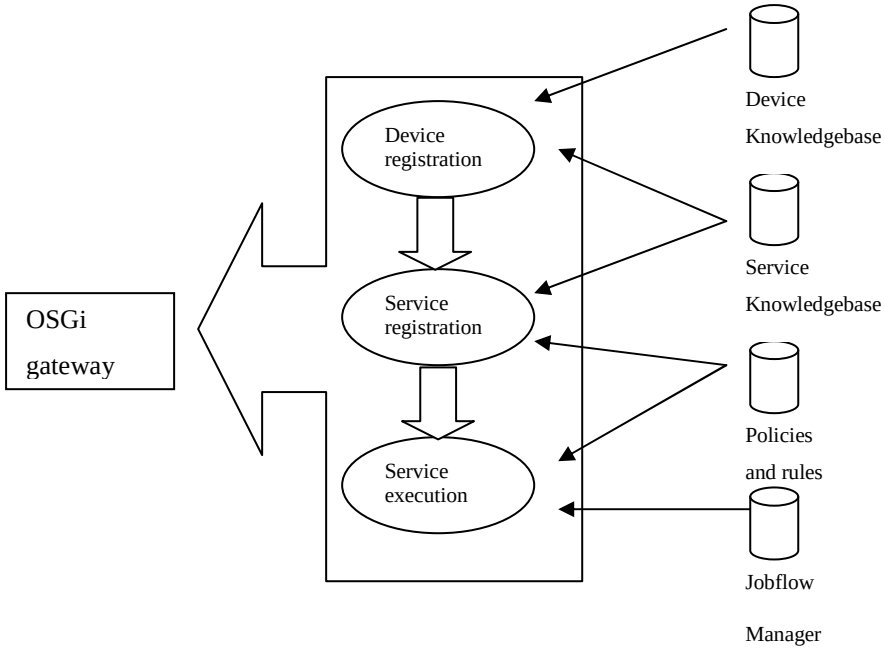


Figure 4-15: Overview of the 3rd Framework architecture displaying the three main processes and their main subsystems.

These three workflows are driven by four sub-systems described in the next sections: a device knowledgebase including information for device capabilities, a service knowledgebase including information required for service adaptation and generation, a policy manager holding and executing the rules and policies and a jobflow manager that stores the execution workflows for each of the available services.

The intelligent model proposed in this framework contains the following properties:

- Reactive: The environment events are sensed. Events then trigger action selection that may lead to actuators changing their environments.
- Rule or Policy-based: The framework uses a rule engine and a set of policies to orchestrate the operation of its distributed devices.
- Adaptive: It adapts to environmental changes and improves its own performance.
- Collaborative: Multiple devices and services share tasks and information in order to achieve shared and common goals.

- Orchestrated: Distributed services are controlled and ordered by designating some leader who acts as a central-planer.
- Shared Knowledge: The context information and system knowledge is shared between the distributed services.

4.7.3 Device and Service Repository

The device repository stores the device information, capabilities and settings that are queried from a device when the device is discovered by the system. This information is usually queried by a device using the UPnP discovery mechanism. In this project the UPnP profile of each device can be extended by the user. For example, when a device is discovered in the home network, the user enters more information that does not exist in the UPnP, such as device location, device friendly name, etc.

Services are defined in the service repository which stores the service names linked with the required and the optional capabilities that are needed to generate a service. When the combinations of devices that meet the required capabilities for a service are registered, then this service is ready for execution, while adding devices that offer optional service capabilities can extend this service configuration and execution.

4.7.4 Policy Management Subsystem

The policy manager stores the policies to drive the service registration workflows and the service execution workflows. For the service registration workflows, the policies are general and orchestrate the workflow by allowing the registration of a device when these device capabilities meet a service requirement (see section 5.5.3). In the service execution workflows the policy manager holds different policies and rules for each of the service that is being executed (section 5.5.4).

4.7.4.1 Rules

The Rule-based model of this framework is a mechanism for utilising a knowledge base's information and then applies logic reasoning to service operations. This model contains a rule engine determining how rules constructed for an IF-fact THEN-fact. When new events are generated, they are represented as new facts. The knowledge base uses the fact in the IF portion of the rule and matches this with current facts contained in the working memory part of the knowledge base. When a match is

confirmed, the action rule gets activated and its THEN statements are added to the working memory. These new facts added to the working memory can also cause other rules to fire.

In this framework the Rule-based model is combined with a workflow model in order to allow orchestration and knowledge sharing between distributed services in a complex topology, for example where a number of different services interoperate to generate a new virtual service.

4.7.4.2 Rule Generation and Execution

A rule engine searches the knowledge base using forward or backward chained searches. Backward chaining starts with a list of goals (or a hypothesis) and works backwards from the consequent to the antecedent to determine if there is information available that will support any of these consequents. On the other hand, forward chaining starts with the available information (data in knowledge base) and uses inference rules to extract more data (from an end user for example or the physical environment) until a goal is reached.

Forward-chaining may be regarded as progress from a known state (the original knowledge) towards a goal state. Backward-chaining means that no rules are fired upon assertion of new knowledge. The proposed framework uses a forward chaining search because the current activities and situated actions in a home environment are driven by the context (e.g. sensors). This fires rules from a known state, i.e., the current context. The final goal may not also be pre-determined, it may be opportunistic. Furthermore, in forward-chaining the reception of new data can trigger new inferences, which makes the engine better suited to dynamic situations in which conditions are likely to change.

4.7.4.3 Rete Algorithm

Forward-chaining systems can become cumbersome if the problem space becomes too large. As the rule-base and working memory grow, a brute-force method that checks every rule condition against every assertion in the working memory can become quite computationally expensive.

There are ways to reduce this complexity, thus making a system of this nature far more feasible for use with real problems. An effective solution to this is the Rete

algorithm [67]. The Rete algorithm reduces the complexity by reducing the number of comparisons between rule conditions and assertions in the working memory. To accomplish this, the algorithm stores a list of rules matched or partially matched by the current working memory. Thus, it avoids unnecessary computations in re-checking the already matched rules (they are already activated) or un-matched rules (their conditions cannot be satisfied under the existing assertions in the working memory). Only when the working memory changes, does it re-check the rules, and then only against the assertions added or removed from working memory.

4.7.4.4 Rule Properties

The rule-based model of the proposed framework has the properties of Table 4-2.

Property	Description
Passive or Active rule generation	A rule can be generated either when a rule administrator defines a rule using a rule interface (e.g. GUI) or automatically by the rule engine
Passive or Active rule execution	The execution of a rule may happen when a context event is triggered (passive) or when a user or a device activate the rule
Re-usable rule templates	Rules generated automatically (passive rule generation) are stored in the knowledgebase for re-use
Rule priorities	In order to avoid rule engine operation conflicts and user personalisation, rules are prioritised
Permitted or non-permitted rules	This property enables the permission to execute a rule based on the current context

Table 4-2 Rule properties

The properties of each rule are stored in the knowledge base along with the rule itself and are executed in the Home Gateway (OSGi). Thus, the Home Gateway is the planer. A generic knowledge base is used. This method decreases the number of rule conflicts as the Home Gateway:

- Has access to the whole knowledgebase and may use more specific matches rather than more general matches
- May use a high priority rule over a lower priority rule
- Use a conflict-resolution template

4.7.5 Workflow Process

The three main processes of this framework, device registration, service registration and service execution, are orchestrated by two types of workflow: the registration workflow and the service execution workflow.

4.7.5.1 Service Workflows versus Device Workflows

It is important to make the following assumptions at this point. A *device* is defined as an actual piece of hardware in the ICT home environment. Devices are divided into two subtypes: a dumb device and a smart device. Dumb devices do not have any networking capabilities, usually have an already programmed PIC (Peripheral Interface Controller) microprocessor with very limited or no memory and can execute basic operations and cannot be altered. ICT or smart devices are networked devices usually running an operating system (e.g. OpenBSD, Linux, or Windows ME, etc.) or a virtual machine. They have a faster processor and more memory than dumb devices and thus they are able to execute code as add-ons to their already installed firmware.

Examples of dumb devices include lights and light switches, microwave, some video players, etc. Examples of smart devices include mobile phones, set-top boxes (e.g. Dreambox, IPBox running Linux), PCs, some LCD TVs, etc.

A **service** is the result of one or more processes that executes in one or more devices. A service has 4 typical steps or loops as they are defined in the SOA: The service registration, service execution, service maintenance and the service de-registration. One or a set of devices need to be combined to complete a virtual service. In turn, this virtual service must follow the SOA specification.

A **device workflow** is an ordered list of processes that a device is programmed to do. A device workflow usually is triggered by a switch or button (e.g. start heating food in a microwave), by a sensor (e.g. motion sensor in an alarm system) or by a timer (e.g. alarm clock). Device workflows are usually hard-coded in the actual device firmware or PIC.

Service workflows are an ordered set of actions that run in an ICT home gateway that the home devices are network connected to. As the service workflow is not placed inside the actual device hardware, it is considered as a virtual workflow.

4.7.5.2 Workflow Types

Workflows include actions, event triggers and rules or policies. They can have three types as they are defined in the next figure (Figure 4-16).

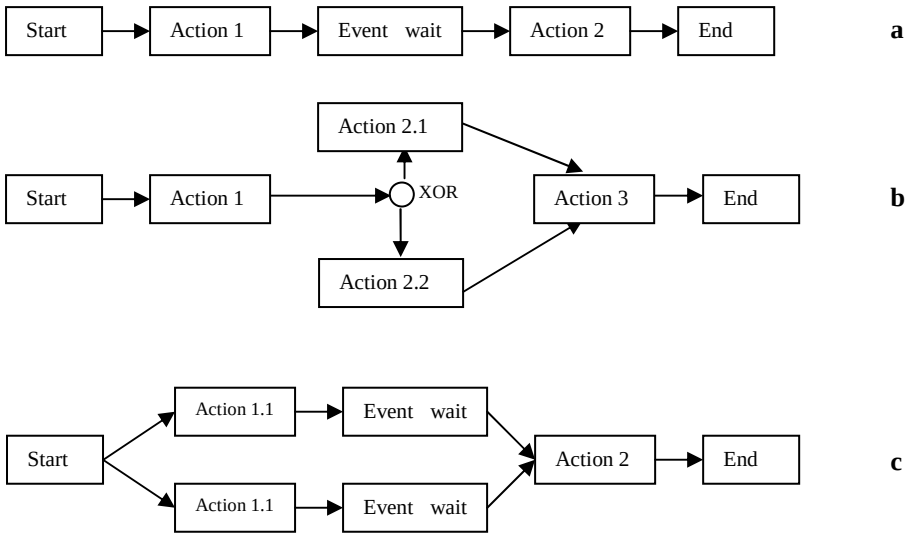


Figure 4-16: a. Simple sequence workflow, b. Branched workflow, c. Parallel-process workflow

In Figure 4-16, three workflows are shown: a simple sequence workflow (a); a workflow also may have one or more branches that change the activity flow based upon events coming in and other variables (b); activities may also spawn multiple thread on a part of the activity, e.g. parallel process workflow.

As services usually include a number of devices, a service workflow (virtual workflow) is used to combine device workflows. Thus a service workflow can be a set of device workflows (e.g. D1 and D2 workflows) where the flow is distributed in

each of the device workflows. There are three dimensions in flow distribution between two or more device workflows (Figure 4-17).

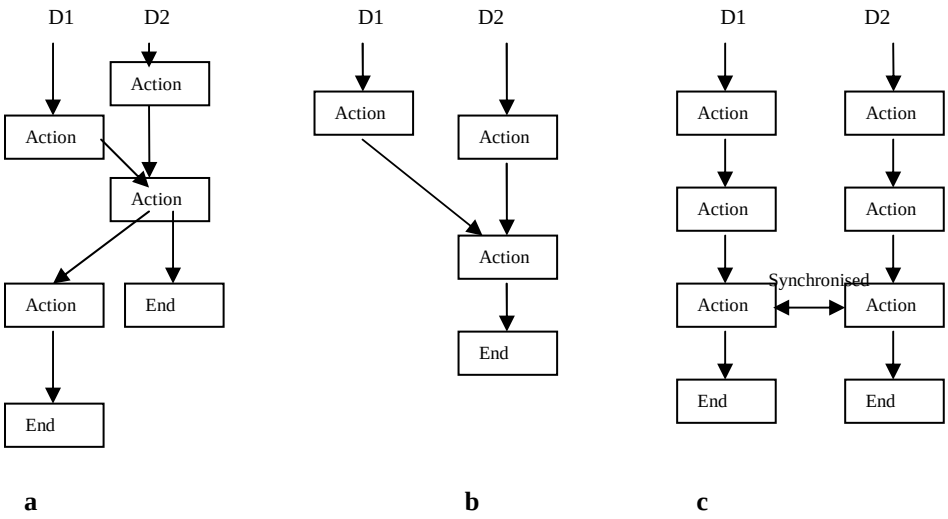


Figure 4-17 a. Nesting synchronous service workflow, b. service workflow through pipe, c. multi-threaded synchronised service workflow

In a nested synchronous service workflow, one device may request the results of an activity or executes an activity on another device and then continue its original flow (a). The service workflow pipe directs the flow of a device workflow to continue on another device (b). Finally, the synchronised service workflow allows activities between two or more devices to be executed and to exchange information in a synchronised way.

4.7.5.3 Registration Workflow

In the registration workflow there are two main loops: the “device registration/service complete” loop and the “service selection” loop. The “device registration/service complete” loop is on a wait state until a device enters the home network. Then the loop takes three actions: 1. executing a number of pre-actions for configuring this device before it is registered, 2. evaluating a device registration rule, that is checking the device capabilities and registers the device with a service only if the device is needed, and 3. executing a number of post-actions. The pre and post actions of each device type are stored in the knowledge base. A simple representation of the rule that evaluates the registration of a device is:

```
When
       device.capabilities[] are needed by service s
Then
       register device to s
else
       drop device
```

Another rule in the registration loop is to evaluate if a service has been completed and is ready for execution. A complete service is the service that has all of its required devices registered with it. For example a light service is complete only when a light device and a motion sensor device have been registered under this service. This rule checks if the service needs more devices to be ready for execution and if all the devices are registered with it, then it asks the user to accept or deny this service. This is the last step of service registration workflow and when the user accepts a newly created service, the "service accept" rule loads a job workflow (will be called jobflow) that has actions and rules for executing the created service.

4.7.5.4 Service Execution or Jobflow Workflow

A jobflow is a set of actions, event notifiers and rules that are used to orchestrate the execution time of a service. Each service has its own jobflow which is stored in the Knowledge Base. When the registration workflow registers and the user accepts a service, the "jobflow" is then loaded and starts. The event notifiers and actions in the jobflow are active during the time the service is in its "running mode". A simple jobflow for the light service is:

```
Start ---> Event trigger: Motion detector device is on -->
Action: Turn light device on --> Timer: wait 60 seconds -->
Action: turn light device off --> End
```

The above jobflow executes a light service with two devices: the light and the motion detector device. The jobflow starts executing when an event from the motion detector is triggered. It triggers an action to turn the light on and then an internal timer waits for 60 seconds before the device is turned off again. Then the job flow enters the event wait state again.

4.7.6 Implementation and Evaluation

This prototype was implemented in Java using the Drools rule management system. Drools is a business rule management system (BRMS) and an enhanced Rules Engine implementation based on Charles Forgy's Rete algorithm. Benefits of using a Drools implementation includes: declarative programming, graphical editing tools and use of an open source license.

- It implements a forward-chaining search model
- It uses the Rete algorithm.

A graphical representation of the registration workflow is shown in Figure 4-18.

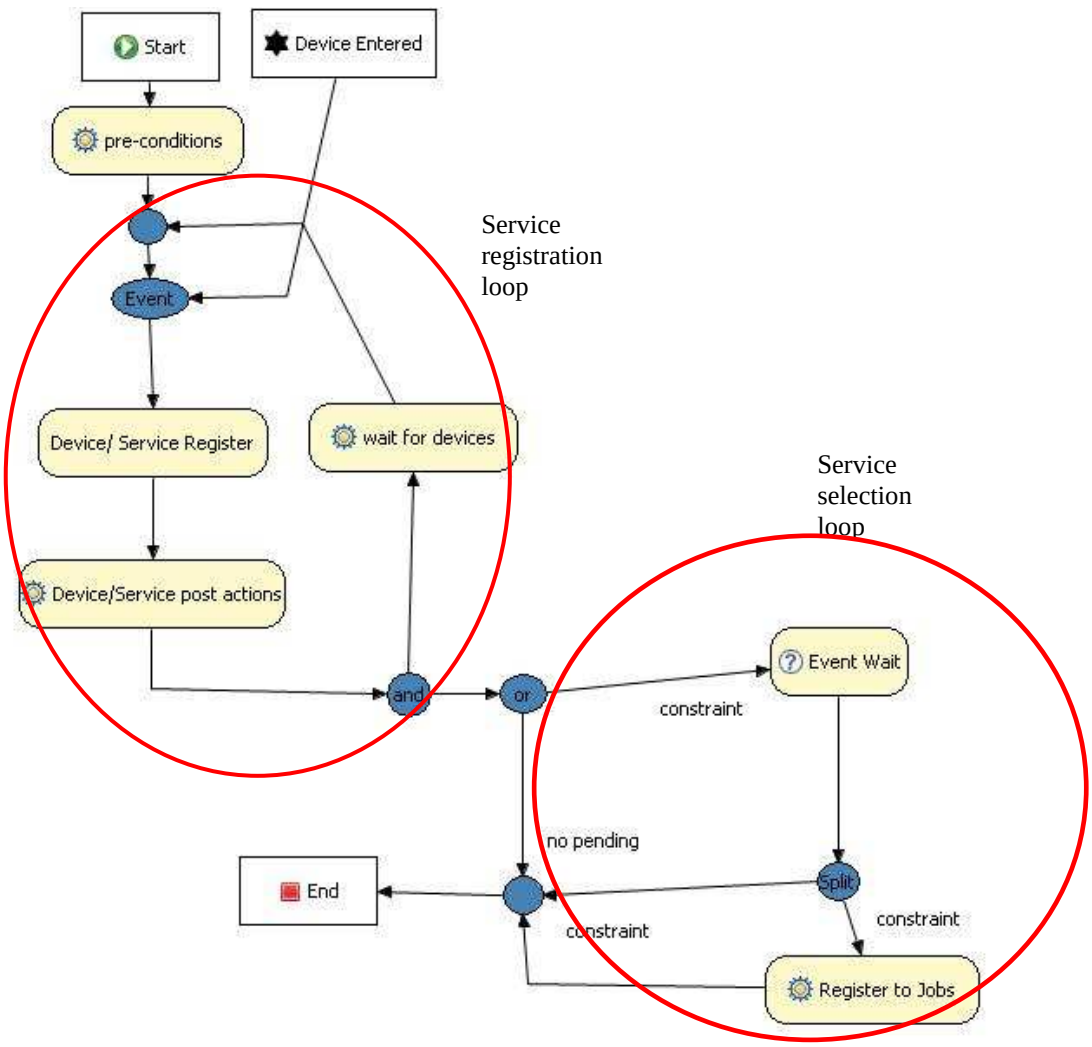


Figure 4-18 Graphical representation of the registration workflow

The service registration process is a workflow loop that includes an event wait, a rule management object and action objects. The event wait “Device Entered” is triggered when a new device is entered and discovered by the discovery mechanisms. Then the workflow continues to the “Device/Service register” rule manager object. At this point, a set of rules examines the new device’s capabilities and if they match the capabilities required by a service in the service knowledgebase, the device is selected and registered with the system. The workflow then moves to the “Device/Service post actions” action object where a configuration message is sent to the device if needed to be configured for the particular service. This loop runs continuously as the devices may enter the home environment at any time.

When a service’s required capabilities met by devices have been registered, the workflow moves to the service selection loop without terminating the registration loop. The selection loop has an “Event Wait” action that requires the user to either select and accept the new service or dissolve it (drop it). Finally, if the user accepts the service, the “Register to Jobs” action loads the execution workflow for this particular service and begins its process.

The service execution workflow (jobflow) for a typical light service is presented in Figure 4-19. This service comprises three devices: a motion detection sensor (user context – user movement), a timer (physical world context) and a light. At the first action of the workflow loop, the “pre-actions” re-set the timer and moves to the “Event Wait” state where the workflow halts until an “Event” is triggered. In this jobflow the “Event” is registered with the motion detection sensor and it is triggered when the sensor is activated. Then, the workflow moves to the “LightRules” rule management object where a set of rules orchestrate this simple service:

```
WHEN sensor is triggered AND timer > 0
THEN set light = 1 AND wait for timer.seconds
```

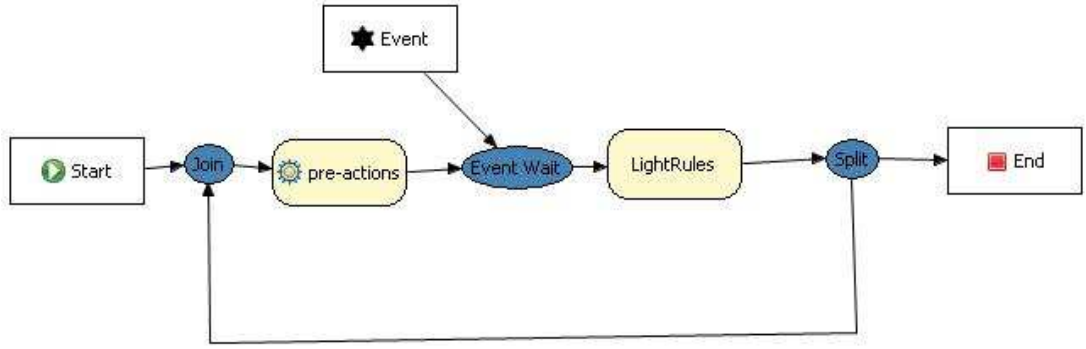


Figure 4-19 Light service execution workflow (job flow)

In the fourth framework and prototype, workflows are combined with rules to support home services life-cycles that follow SOA specifications. Service registration is based on rules directed by a device via a service knowledgebase. This enables automatic service registration. Service selection requires the user to select or drop a generated service.

Service execution uses workflows to orchestrate the various devices that comprise the service. Workflows contain: actions, switch (if/otherwise) statements, joins and event listeners. The context is acquired by these event listeners. A hardware sensor is sensing the physical world (in case of the physical world context), or a device/ service message triggers an event (in case of ITC Context) or a set of sensors in combination of a set of rules and policies understand the goal of a user and triggers an event (in case of the user context).

At this point the execution workflows are based in simple home service scenarios and demonstrate simple service execution including multiple devices. Also the context acquired is limited to find the user movement rather than understanding the user-goal, or to sense the current weather using a weather station rather than downloading the forecasting data from weather web services and learning the weather forecast.

5 Discussion

In this research project a number of other research projects and technologies were surveyed in order to design and implement a ubiquitous, context-aware framework that orchestrates devices and services in a home environment.

The proposed solution and framework used the OSGi service framework to support the life-cycles and messaging between the home services, as it provides better service discovery mechanisms and is much faster than the typical web services technologies. The framework proposed in this research project was built on the top of the OSGi layers. That way it extends the OSGi functionality by adding support for autonomous device combination and interoperability, dynamic user interfaces, context awareness and finally device orchestration using planers and pre-defined rules.

The following sections discuss the achievements and novel parts of this project in more detail.

5.1 Achievements & Novelty

During this research project progress the following were achieved.

5.1.1 Device Integration

A middleware framework has been developed that enables device combination using a web interface allowing users to enter simple pseudo-English sentences (e.g. Turn on the lights at 10.00 in the morning) to configure and control a combination of two devices. This framework orchestrates device configuration and operation using the OSGi framework.

Service adaptation and combination (novelty): Services are provided by devices. A device provides at least one service but there are devices that can provide two or more services. For example, a digital satellite box can provide 6 different kinds of services: accurate time service (each satellite transponder sends time signals taken from a satellite's atomic clock), video service, audio service, record/playback service, Teletext service and electronic program guide service. A service combination process enables two or more different services to adapt and combine to create one or more other services.

Demonstrator Evaluation: The demonstrator 1 implemented during this project, implements a method where different services provided by multiple devices can interact each other by exchanging messages. OSGi framework was used and services were easily converted into OSGi bundles, however in this demonstrator someone has to pre-define each service capabilities and its compatible services. This demonstrator's experiments showed that UPnP was not very efficient. For this reason UPnP tends to be used only during the only service discovery phase of the service life-cycle.

5.1.2 Dynamic Interface Generation – Device Portals

The above framework was extended to support more than two device combination and an interface generator that creates a dynamic control interface for the device integrated service.

Dynamic interface generation for device integration (novelty): Multiple devices combined into a new virtual service raises the problem of how a user interacts with a new service as there is no pre-defined interface. This framework solves this problem by providing methods to generate a new dynamic interface based on the capabilities of each of the devices needed to generate the virtual service.

Demonstrator Evaluation: The second demonstrator implements this framework successfully with the limitation that each device user-interface must be pre-designed. A first approach to this framework was to automate the virtual interface creation by creating a service-to-interface data source that would match a service with a number of user-interface components. For example the video player service would match with the play, stop, pause, next, previous, interface components. Then the UPnP discovery mechanism would acquire the service name and capabilities and the service-to-interface data source would push only the valid interface components for each service combination. However, this was not efficient as the UPnP information of each service was not complete enough in order to correctly match a service with its interface components.

The final virtual interface of the device combination is not pre-defined and is constructed at the time the devices are combined.

5.1.3 Context-aware Device Control

Sensors located on a user's body allow tracking of a user in a 3-D space. 3-axis accelerometers provide acceleration, velocity and position of the user. Processing of these measurements can sense additional activities such as sleeping, relaxing, walking fast, running, etc.

User tracking in indoor environments (novelty): The user location measured by combining the signal strength of networked sensors carried by the user with acceleration data produced by the user as he moves around the home. RSSI (received signal strength indicator) comparison and ranking techniques and acceleration data alone do not provide precise results for obtaining the user location. However, RSSI can be used to locate the user in certain house locations (e.g. places close to the base stations) and then accelerometer tracking techniques can be used to track the user's movement and provide better tracking information as the user moves away from the base stations where RSSI value readings get sensitive to object reflections. A virtual map of the room contains a number of areas where it is known that RSSI value readings are correct without getting errors from object reflections. When the RSSI readings match these areas a new tracking session is initiated from that point. Then the user is being tracked by the accelerometer readings until another known RSSI value is reached.

Demonstrator Evaluation: This approach was implemented in the third system (demonstrator). This system implements user-cantered context awareness methods to find and track user movement indoors. Sensor limitations and efficiency led to a number of different approaches in order to complete this framework. Beginning from RSSI readings and triangulation using three antennas to extensions of up to 8 antennas and combinations of RSSI with compass and acceleration sensors, the final demonstrator was efficient and capable of locating the user in a room within a radius of less than a meter.

Gesture-based scalable physical device control (novelty): Devices to be controlled may be identified into a 3-D space when the user is pointing to it. The selected device then can be controlled by predefined gestures (e.g. a tilt on the left of the user's hand may increase the volume of an audio device).

An extension to the third system was another system that gets the position of the hand of a person along with its gesture. A simple approach to this is implemented. It uses the location determination technique of section 4.6.1 together with a compass sensor that acquires the hand's direction.

5.1.4 Policy Based Device Control

A policy based device control framework, identifies the user activities and goals based on context driven and situated action models. User and device actions are captured by sensors as events that trigger actions to control devices based on policies in an Event-Condition-Action model.

5.1.5 Device Orchestration using Workflows and Pre-defined rules

Proposed SOA lifecycle extensions develop and manage the orchestration of reusable services in home environments. A set of pre-defined rules and workflows drive a device-service lifecycle from the moment a device enters the home environment until it is removed from it.

Planning device orchestration using rules/workflows (novelty): Service discovery, selection and configuration, during the first phase of a service life-cycle, involves querying UPnP information and pre-defined rules. Different workflows manage the service operation and maintenance (e.g. update, reconfigurations) where the user's input is necessary only when something unexpected happens (e.g. something that is not defined in a rule). Finally, when the service is not needed any more, another workflow dissolves it from the home environment and re-configures dependent services.

Demonstrator Evaluation: The last demonstrator was a combination of the frameworks defined plus a new framework adding orchestration of discovered services. The orchestration workflow was designed, built and evaluated in two phases: the service registration / de-registration and the service operation and maintenance. The scenarios that were evaluated for this demonstrator successfully show that rules and policies are essential to orchestrate and automate multi-services in the home with minimum user-direction. The demonstrator's experiments show that complexity is increased as the services workflow combines more services, however fault control frameworks can be used to decrease the number of errors this complexity occurs.

5.2 Future Work

In this section the future work of this research project is presented.

5.2.1 Service Orchestration and Planning

The service orchestration planning process described in this document has been limited to goal identification, plan creation phases and simple plan execution. There are two missing phases that the service orchestration planner requires in order to be complete: a plan execution phase and a re-planning phase. In current work, the orchestration planner identifies the user goal and creates a plan with the actions that should be performed in order to reach that goal. This plan includes actions such as: automatic or manual device configuration, device combination, service operation and user tasks. When the plan has been built it then needs an executor, a set of operations that executes each action of the path (e.g. the action "turn heating off" should be translated into a controlled message that will instruct the OSGi gateway to turn off the heating). Re-planning is another phase that is missing from the current service orchestration planning. It is the process that is triggered when a configuration in the environment has been altered (e.g. a service used to participate in the plan execution is no longer available) or an unexpected situation has arisen.

Some challenges that arise in a complete service orchestration planning process and will be addressed in the future work are listed in the following table (Table 5-1).

Service Orchestration Planning Phase	Challenges
Plan Execution	During the plan execution a device may not be OSGi enabled and cannot be accessed by the OSGi gateway (e.g. the user may need to operate the device manually)
	A device may have crashed and provide information that is not correct
Re-Planning	Sensing that a device or service configuration has been altered and there is a need for re-planning.

Table 5-1: Future Service orchestration planning challenges

While service choreography across multiple devices requires planning algorithms to execute in each device or each home network's access node, service orchestration requires the planning algorithms to execute in a gateway from where the service is

controlled and from where the properties are visible. Considering that service choreography planning algorithms require an "open" OS, some storage and a processing power and, as many devices in the home have limited processing and storage capabilities and they may not allow third party applications (such as planning algorithms) to be executed, it is difficult for choreographic algorithms to run in every home device. Thus, the best approach in managing multiple devices in the home is device orchestration.

Service orchestration across multiple devices needs an execution environment, a place where the orchestration algorithms will run and where plans will be created, executed and stored. For the future work of this project, service orchestration algorithms could be built and evaluated in a home service gateway. This gateway must follow these requirements:

- "Open" OS architecture: The OS of the gateway must allow third party applications to run.
- Storage: The gateway should contain sufficient memory for plan storage.
- Accessibility. The gateway must be visible to the devices and the devices must be visible to the gateway. ICT resource demanding devices include various networking technologies such as Bluetooth, ZigBee, Wi-Fi, Ethernet and USB. They can communicate with the gateway assuming that both the device and the gateway have the same networking technology. However, other home devices that have low ICT resource requirements do not include networking. The future work of this project investigates techniques for planning execution with these devices.
- Service discover: The gateway should be able to discover devices that enter the home environment and remove devices that exit, as there are mobile devices in the home.

Previous work in this project used an emulator to experiment with simple service planning algorithms. The plan was created after the user has input a rule or policy for one or more specific services. The plan then was executed on an OSGi emulator running on a computer and was stored until the emulator is shutdown. This work could be extended in further work through the use of a more complex planning algorithm that allows automatic plan creation based on policies and pre-defined

constraints. The orchestration planning algorithm evaluation will take place in an OSGi gateway situated in a house environment (testbed).

5.2.1.1 Orchestration using Business Process Execution Language

A **business process** is a collection of related, structured activities or tasks that produce a specific service (serve a particular goal) for users. Considering this definition of a business process, home tasks and activities can be considered as business processes that can be orchestrated using the Business Process Execution Language (WS-BPEL) [68].

In a future work scenario, an OSGi gateway could invoke Web services for each device event or action that is imported into a BPEL orchestration flow. The flow then orchestrates a set of devices by invoking other Web services that are received from the OSGi gateway and translated into device control messages.

5.2.2 Universal Service Controller

Conventional methods to control devices require users to interact with the device using its build-in user interface or using a remote control to access the device functions. Although there are universal remote controls introduced in the market that can be used to remotely operate a group of devices, these universal controls can access only a specific type of devices (mostly AV devices).

A universal portal controller framework is introduced to this project. This not only enables remote control of a wide area of services and devices, but also enables gesture-based control of devices and queries of the physical environment. A portal controller could be a device that includes a number of sensors and buttons that can be wirelessly connected to a home gateway (e.g. Sun SPOT).

For gesture-based scalable physical device control, the device to be controlled may be identified in 3-D space when a user is points to it using the controller. The selected device then can be controlled by gestures, for example a tilt on the left of the user's hand may increase the volume of an audio device. Another application of a gesture-based controller is the device combination in a "drag n drop" manner. For example a user points with the controller to a room light, presses and holds the button on the controller and move the controller towards the TV while holding the button. Then he

releases the button while pointing the TV and a GUI on TV to display the house lights configuration.

Table 5-2 lists some of the challenges that arise when using a device control as a gesture identifier.

Challenge	**Examples**
Location of the controller in the environment	X, Y and Z location of the controller
State of the controller	Is it pointing up or down? What is the angle? Where is it heading? North or South?
Accuracy of the controller	When the user points to an object, how accurate is this?
	What happens when two or more devices are very close and the controller accuracy is not very good?
	What happens when a device is on the back of another device?

Table 5-2: Gesture-based universal controller challenges

Some of these challenges can be addressed by calculating the location variables out from the controller's (Sun SPOT) sensors, others can be calculated and others need additional sensors that are not found on the Sun SPOT. The Y axis is easily retrieved by the Sun SPOT accelerometer by measuring the gravity force G on the Y axis. The Sun SPOT's location can be found by using 3 Sun SPOT base stations and determining its position by measuring the signal strength. The heading of the Sun SPOT can be found by using a magnetic compass as an extra sensor attached to the spot's input ports. These magnetic compass sensors can provide us with accurate measurements of the spot's angle with an accuracy of 0.1°.

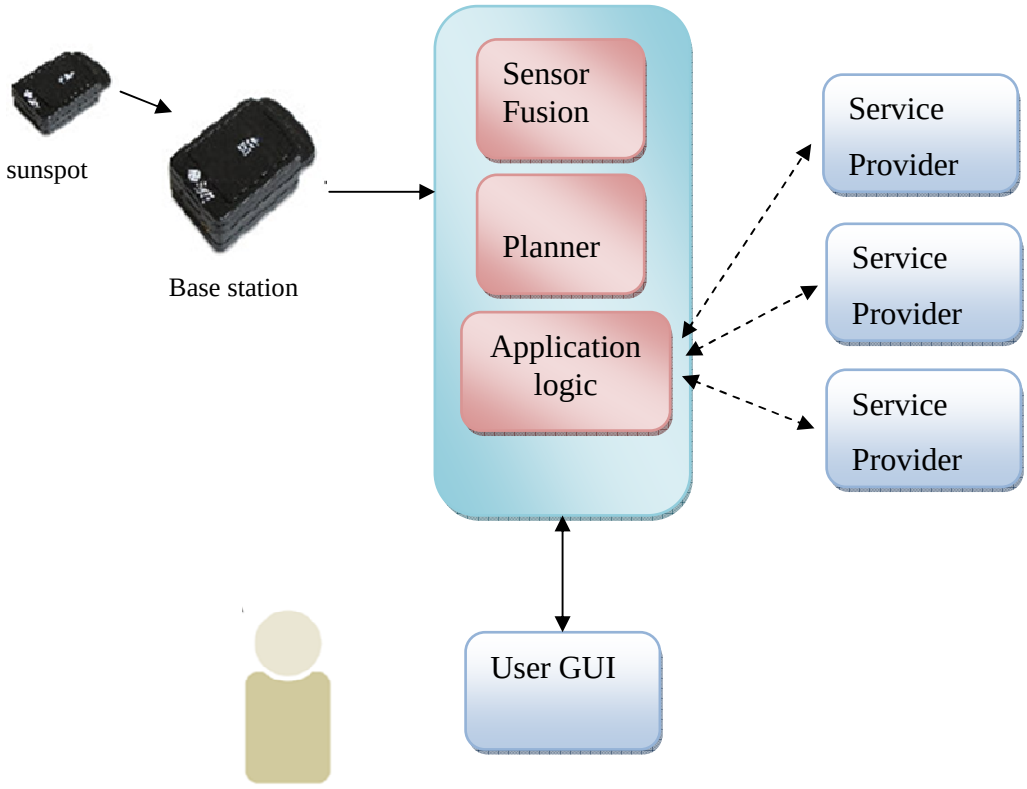


Figure 5-1: Universal device and service controller model

The Figure 5-1 shows some of the components that could be built in a future work of this project. Sun SPOTs send raw data to their bases stations where the sensor fusion component filters and computes context information from the sensor raw data (e.g. location). This context data can help determine user goals. For example the user points to window, presses and holds the Sun SPOT button, moves the Sun SPOT to point to TV and releases the button can lead to goal: show the weather forecasting on TV screen.

These goals then are sent to the Planner module that will create a set of actions to reach the user goal (e.g. show the weather forecast on the TV screen). These actions finally are sent to the application logic module which communicates with the device services.

6 Conclusion

This thesis presents the research work towards a ubiquitous context-aware management of services in the ICT home. Survey of the current pervasive and context-aware computing environments presents the state-of-the-art technologies and architectures. However, the survey shows that limitations exist in supporting device life-cycles, device collaboration and orchestration. In addition, the technologies for context acquisition, for example for user context acquisition, do not provide accurate results or they require expensive equipment. Devices are not aware of each other and are not aware of the services available in the home environment, making it impossible for devices to adapt to a dynamic user goal.

To solve the above problems some methods have been proposed in this document. These methods can be divided into two categories, the first category proposed methods to identify a user goal based on the current context and any corresponding situated actions while the second category proposed methods to make a plan of actions distributed in devices, services and users in order to reach the goal specified from the first category. A number of algorithms have been proposed that support device orchestration, device selection and dynamic interface generation that will lead to more ubiquitous operation of current home services. These algorithms include the rules/workflow design and implementation of device orchestration framework, the XML-XSD Schema for device discovery and interoperability and the UPnP extension with custom user interface components that lead to virtual user interfaces. Also, some methods have been proposed regarding the context acquisition mostly on the user's domain. Some examples include methods to identify the user identity and user location. The future work of this project presents the system architecture and the framework that will support the current architecture in order to meet the requirements of this project.

Appendix A

Demonstrator 1:

The first demonstrator of this project presents a device interoperability framework where a description of services exposed by devices is stored into a simple XML knowledge base. The system queries this knowledge base to make service-to-service matches. The next figure shows how the lights service can be matched with the timer service.

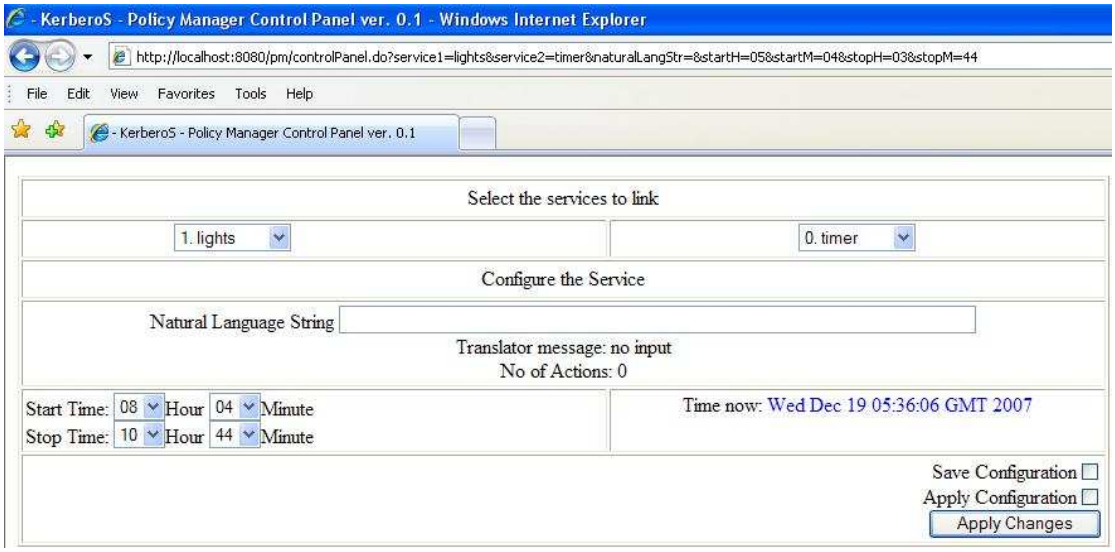


Figure A-1: Demonstrator 1. Two services are combined and configuration information has been entered

On another run of the demonstrator, a service that has not compatibility with any service is selected. The system interface does not allow any match.

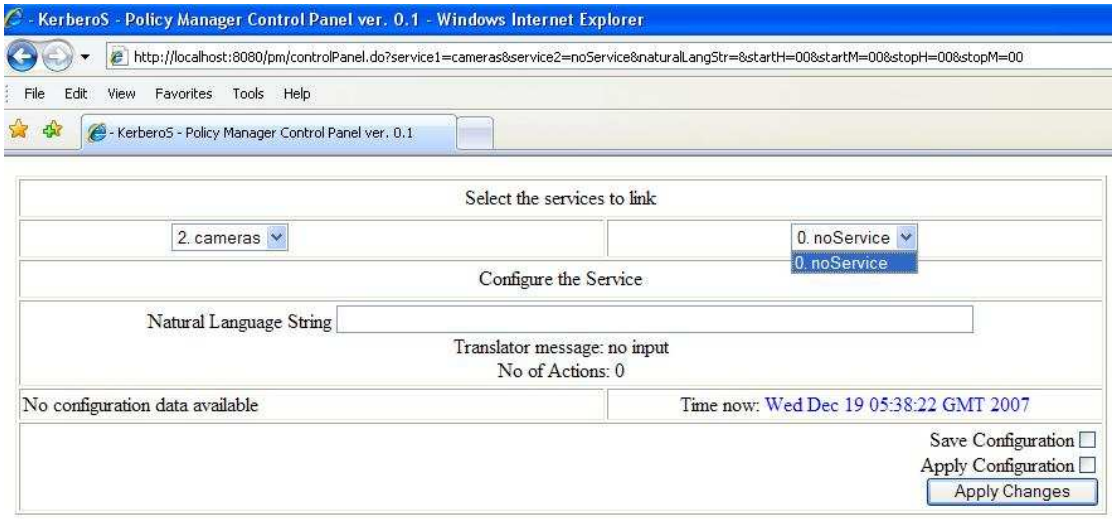


Figure A-2: Demonstrator 1. Camera service cannot be combined with other services

Finally, another approach for a more automated interoperability system was to allow a user to input a command that matches the two services as a phrase in English. The following figure shows a simple scenario of this approach.

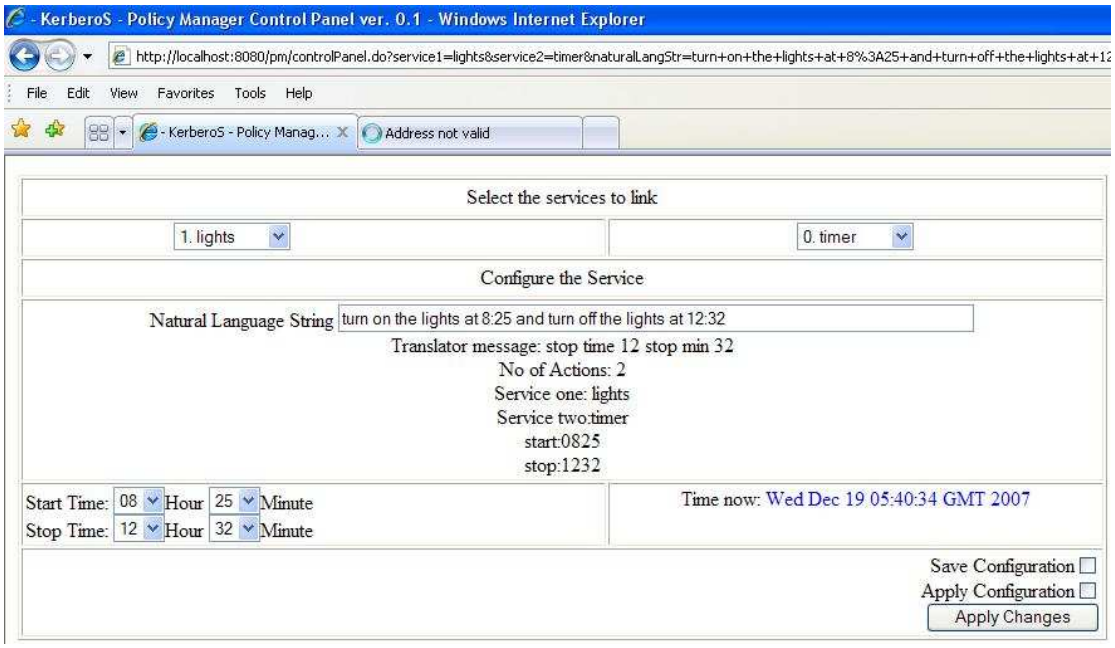


Figure A-3: Demonstrator 1. Input service configuration expressed in natural language.

Prototype 2:

This prototype extended the Prototype 1 by adding UPnP discovery methods and virtual user-interface capabilities after the multi device combination. As the next figure shows, the Service Discovery section presents every service publishes itself through UPnP. By adding the service its UPnP description (XML message) is stored in memory and the service is moved to the Configuration section.

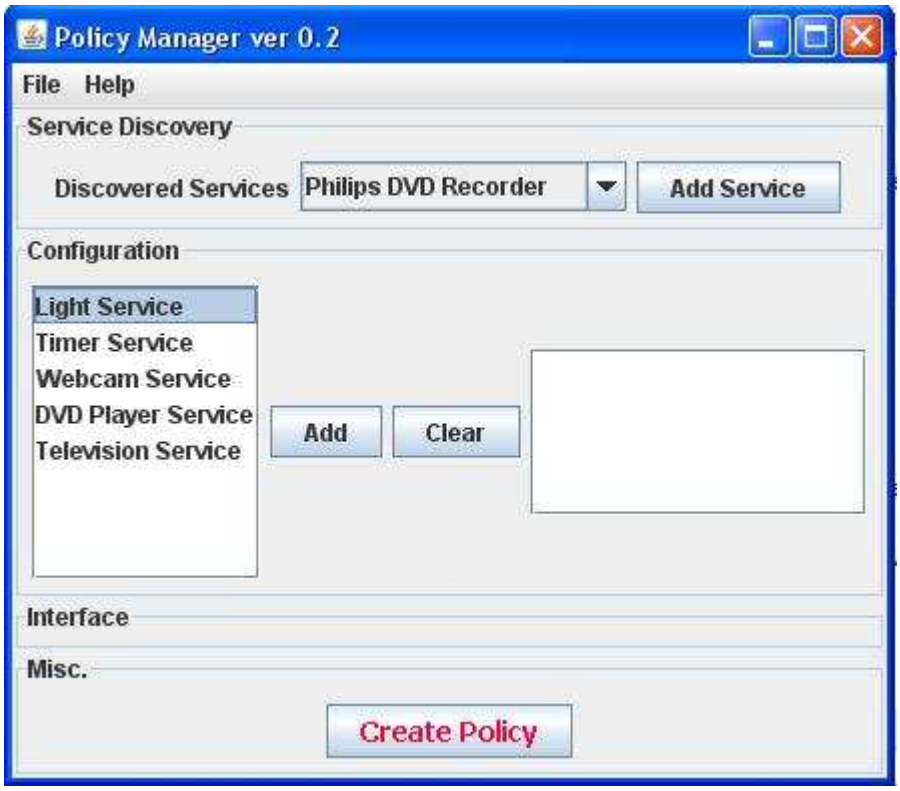


Figure A-4: Demonstrator 2. Initial state where the services have been discovered

When one of the available services is added to the right list, its status changes to active and the virtual-interface mechanism draws its interface. On the next figure only one service is selected thus only its interface is presented on the interface section.

Figure A-5: Demonstrator 2. One of the discovered services is selected and its interface is displayed.

Later the Television Service is selected together with the DVD Player service. The TV Service adds more settings and capabilities to the already existing DVD user-interface. Next figure presents a screen capture of the demonstrator after two compatible services are selected.

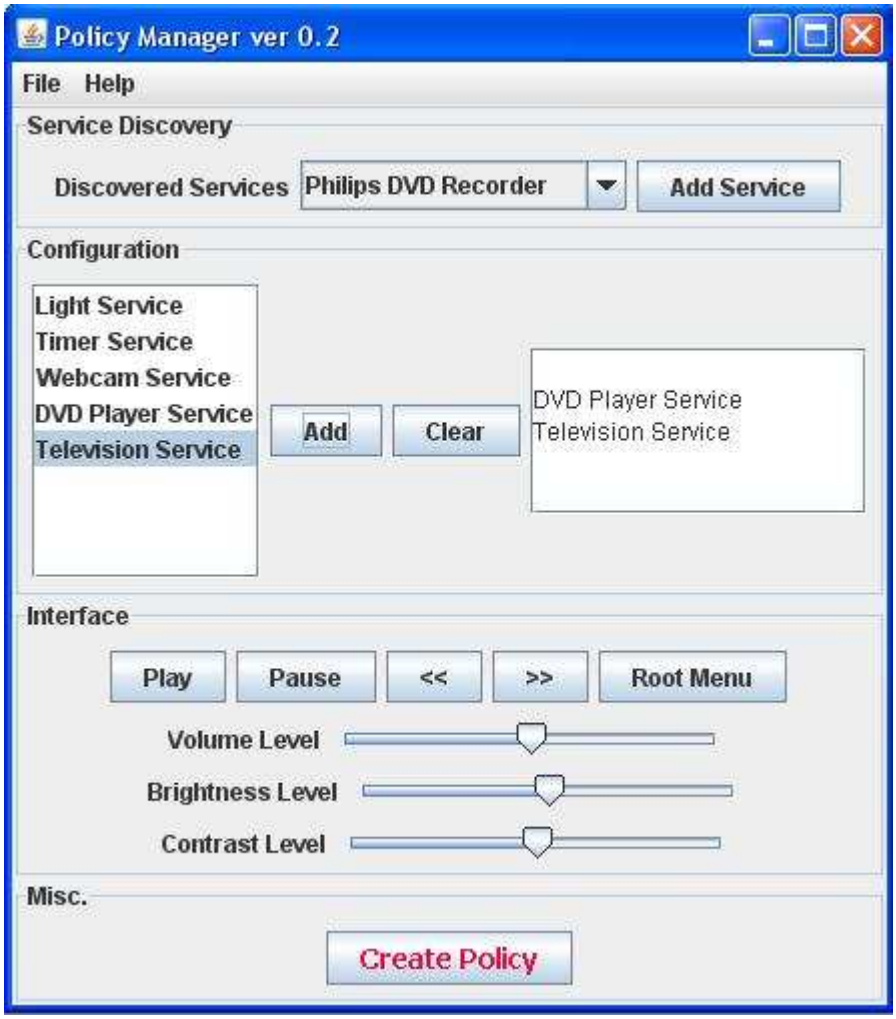


Figure A-6: Demonstrator 2: When two of the discovered services are selected, a dynamic interface containing components of both the two services interfaces is displayed.

Glossary of Terms

Bundle A bundle is an OSGi term for an application or component. It is a discreet software component. Some of this project frameworks' application layer is a set of OSGi bundles.

CCI Computer-to-Computer Interaction. It is the interaction process that takes place between two services.

CHI Computer-to-Human Interaction. The interaction between users and computers. Users may interact with the computer by using a user-interface or the computer may sense the user's actions using sensors.

Orchestration / Choreography Co-ordination of events in a process. Overlapping with the related concept of choreography, orchestration directs and manages the on-demand assembly of multiple component services, to create a composite application or business process. Orchestration tends to imply a single co-coordinating force, whereas choreography also applies to shared co-ordination across multiple autonomous systems. After evaluating several competing specifications, mainstream sentiment is now converging on BPEL4WS as the core standard for web services orchestration.

OSGi Open Services Gateway Initiative. A Java-based specification that describes a runtime and component model. This research project's prototypes run on an OSGi implementation called Knopflerfish.

RMI Remote Method Invocation — a specification for RPC (remote procedure calls). The client can invoke methods on objects remotely residing in the server, possibly passing it primitives or objects as parameters and receiving a primitive or object as a result

RPC A method a program can use to make a call to another program across a network without specifically dealing with network protocols. It is often used for printing across a network.

SOAP SOAP is a protocol defining how XML-encoded information can

	be passed using the web-standard hypertext transfer protocol (HTTP).
Sun SPOT	Sun Small Programmable Object Technology is a wireless sensor network (WSN) developed by Sun Microsystems. The device is built upon the IEEE 802.15.4 standard. Unlike other available mote systems, the Sun SPOT is built on the Squawk Java Virtual Machine.
UDDI	Universal Description, Discovery and Integration. A specification for development of global online directories and registries of Web services. UDDI allows organizations to register their technical specifications (such as integration profiles and capabilities) and then identify the specifications of others.
WSDL	Web Services Description Language. A standard by which a web service can tell clients what messages it accepts and which results it will return. WSDL is an XML language which is used by service interfaces and protocol agreements, among others, for describing access to Web Services. WSDL is independent from the underlying service implementation language or component model.
XML	Extensible Mark-up Language. XML is a specification developed by the W3C and is a mark-up language for structured documents
XSD	XML Schema Definition (language). Used to define the schema for an XML document. An XSD file will have a .xsd file extension.

References

[1] M. Weiser, “The Computer for the Twenty-First Century,” Scientific American, vol. 265, 1991, pp. 94-104.

[2] S. Poslad, *Ubiquitous Computing: Smart Devices, Environments and Interaction*, Wiley, ISBN-13 9780470035603.

[3] Sun SPOT (Sun Small Programmable Object Technology), http://research.sun.com/spotlight/Sun SPOTSJune30.pdf, retrieved on 10/05/08

[4] R. Want, A. Hopper, V. Falcao and J. Gibbons, “The active badge location system,” *ACM Trans. Inf. Syst.*, vol. 10, 1992, pp. 91-102.

[5] D. Gerlan, D.P. Siewiorek and A. Smailagic, “Project Aura: towards distraction-free pervasive computing,” *IEEE Pervasive Computing*, vol. 1, 2002, pp. 22-31

[6] N. Gershenfeld, R. Krikorian and D. Cohen, “The Internet of Things,” Scientific American, vol. 291, 2004, pp. 76-81.

[7] R. Harper, T. Rodden, Y. Rogers and A. Sellen, *Being Human: Human Computer Interaction in 2020*, Microsoft Research Ltd, ISBN-13 9780955476112.

[8] G. D. Abowd, “Classroom 2000: an experiment with the instrumentation of a living educational environment,” *IBM Syst. J.*, vol. 38, 1999, pp. 508-530.

[9] W.K. Edwards and R. E. Grinter, “At Home with Ubiquitous Computing: Seven Challenges,” *3rd International Conference on Ubiquitous Computing, Atlanta, Georgia*, vol. 2201, pp. 256 – 272.

[10] D. Dearman and J.S. Pierce, “It's on my other computer!: computing with multiple devices,” *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, Florence, Italy: ACM, 2008, pp. 767-776.

[11] M. Mateas, T. Salvador, J. Scholtz and D. Sorensen , “Engineering ethnography in the home,” *Conference companion on Human factors in computing systems: common ground*, Vancouver, British Columbia, Canada: ACM, 1996, pp. 283-284.

[12] M. Mozer, “The neural network house: An environment that adapts to its inhabitants,” *Proceedings of the American Association for Artificial Intelligence*, 1998.

[13] P. Cohen , J. Clow, M. Johnston, D. McGee, J. Pittman amd I. Smith, “Quickset: A Multimodal Interface for Distributed Interactive Simulation,” *Proceedings of the UIST'96 demonstration*, 1996, pp. 217-24.

[14] P.R. Cohen, M. Wang and S. C. Baeg, “An open agent architecture,” *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 4, 1994, pp. 1-8.

[15] O. Lemon, A. Gruenstein and S. Peters, “Multi-tasking and collaborative activities in dialogue systems,” *Proceedings of the 3rd SIGdial workshop on Discourse and dialogue - Volume 2*, Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 113-124.

[16] D. Marples and P. Kriens, “The Open Services Gateway Initiative: An Introductory Overview,” *IEEE Communications Magazine*, vol. 39, issue 12, 2001, pp. 110-114

[17] M.P. Papazoglou and W. Heuvel, “Service oriented architectures: approaches, technologies and research issues,” *The VLDB Journal*, vol. 16, 2007, pp. 389-415.

[18] C. Elting, “Orchestrating output devices: planning multimedia presentations for home entertainment with ambient intelligence,” *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, Grenoble, France: ACM, 2005, pp. 153-158.

[19] C. Kray, A. Krüger and C. Endres, “Some Issues on Presentations in Intelligent Environments,” *Ambient Intelligence*, 2003, pp. 15-26.

[20] SMIL, “Synchronized Multimedia Integration Language”; http://www.w3.org/AudioVideo, retrieved on 12 March 2009.

[21] W3C, *Web Services Choreography Description Language, Version 1.0*; http://www.w3.org/TR/ws-csl-10, retrieved on 07 February 2009

[22] B. Schilit and U. Sengupta, “Device ensembles [ubiquitous computing],” *Computer*, vol. 37, 2004, pp. 56-64.

[23] M. W. Newman, T. Smith and B. Schilit, “Recipes for digital living [ubiquitous computing in consumer electronics],” *Computer*, vol. 39, 2006, pp. 104-106.

[24] OSGi Alliance, *OSGi Service Platform*, IOS Press, 2003.

[25] Gat's ATLANTIS: http://ai.eecs.umich.edu/cogarch2/specific/gat.html, retrieved on 19 February 2009.

[26] T.M. Mitchell, J. Allen, P. Chalasani, J. Cheng, O. Etzioni, M. Ringuette and J.C. Schlimmer, "Theo: A framework for self-improving systems," in K. VanLehn (ed.), *Architectures for Intelligence*, pp. 323-355, Lawrence Erlbaum Associates, Hillsdale, NJ, 1991.

[27] J. E. Laird, A. Newell and P. S. Rosenbloom, “SOAR: an architecture for general intelligence,” *Artif. Intell.*, vol. 33, 1987, pp. 1-64.

[28] H. Lieberman and J. Espinosa, “A goal-oriented interface to consumer electronics using planning and commonsense reasoning,” *Knowledge - Based Systems*, vol. 20, 2007, pp. 592-606.

[29] H. Lieberman, A. Faaborg, J. Espinosa and T. Stocky, “Commonsense on the Go,” *BT Technology Journal*, vol. 22, Oct. 2004, pp. 241-252.

[30] A. L. Blum and M. L. Furst, “Fast planning through planning graph analysis,” *Artificial Intelligence*, vol. 90, 1997, pp. 281-300.

[31] R. Fikes and N. Nilsson, “STRIPS: A new approach to the application of theorem proving to problem solving,” *Artificial Intelligence*, vol. 2, 1971, pp. 189-208.

[32] S. J. Russell and P. Norvig, *Artificial Intelligence: Modern Approach*, Prentice Hall, 1995.

[33] C. Elting and G. Michelitsch, “A multimodal presentation planner for a home entertainment environment,” *Proceedings of the 2001 workshop on Perceptive user interfaces*, Orlando, Florida: ACM, 2001, pp. 1-5.

[34] W3C Web Service Activity home page, http://hwww.w3.org/2002/ws/, retrieved on 02 May 2010.

[35] Web Service Description Language (WSDL), http://www.w3.org/TR/wsdl, retrieved on 02 May 2010.

[36] Simple Object Access Protocol (SOAP), http://www.w3.org/TR/soap/, retrieved on 02 May 2010.

[37] E. André, J. Müller and T. Rist, "WIP/PPP: automatic generation of personalized multimedia presentations," *Proceedings of the fourth ACM international conference on Multimedia*, Boston, Massachusetts, United States: ACM, 1996, pp. 407-408.

[38] M. D. Brouwer-Janse, R. W. Bennett, T. Endo, F. L. van Nes, H. J. Strubbe and D. R. Gentner, "Interfaces for consumer products: "how to camouflage the computer?"," *Proceedings of the SIGCHI conference on Human factors in computing systems*, Monterey, California, United States: ACM, 1992, pp. 287-290.

[39] D. Davis and M. Parashar, "Latency Performance of SOAP Implementations", *CCGRID '02 Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, Washington, DC, United States, 2002, pp. 407.

[40] J. Piesing, "The DVB Multimedia Home Platform (MHP) and Related Specifications," *Proceedings of the IEEE*, vol. 94, 2006, pp. 237-247.

[41] J. Nichols, B. Myers, M. Higgins, J. Hughes, T. Harris, R. Rosenfeld and M. Pignol, "Generating remote control interfaces for complex appliances," *Proceedings of the 15th annual ACM symposium on User interface software and technology*, Paris, France: ACM, 2002, pp. 161-170.

[42] T. D. Hodes, R. Katz, E. Servan-Schreiber and L. Rowe, "Composable ad-hoc mobile services for universal interaction," *Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, Budapest, Hungary: ACM, 1997, pp. 1-12.

[43] J. Dan R. Olsen, S. Jefferies, T. Nielsen, W. Moyes and P. Fredrickson, "Cross-modal interaction using XWeb," *Proceedings of the 13th annual ACM symposium on User interface software and technology*, San Diego, California, United States: ACM, 2000, pp. 191-200.

[44] S. R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan and T. Winograd, "ICrafter: A service framework for ubiquitous computing environments," *In Ubicomp*, vol. 2201, 2001, pp. 56-75.

[45] H. Lieberman and J. Espinosa, "A goal-oriented interface to consumer electronics using planning and commonsense reasoning," *Proceedings of the 11th international conference on Intelligent user interfaces*, Sydney, Australia: ACM, 2006, pp. 226-233.

[46] B. Schilit, N. Adams and R. Want, "Context-aware computing applications," *In Proceedings of the Workshop on Mobile Computing Systems and Applications*, 1994, pp. 85-90.

[47] K. Finkenzeller, *Rfid Handbook: Radio-Frequency Identification Fundamentals and Applications*, John Wiley & Sons, 2000.

[48] A. Harter, A. Hopper, P. Steggles, A. Ward and P. Webster, "The Anatomy of a Context-Aware Application," *In Mobile Computing and Networking*, 1999, pp. 59-68.

[49] A. Ward, A. Jones and A. Hopper, "A new location technique for the active office," *Personal Communications, IEEE [see also IEEE Wireless Communications]*, vol. 4, 1997, pp. 42-47.

[50] P. Bahl and V. Padmanabhan, "RADAR: an in-building RF-based user location and tracking system ," *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, 2000, pp. 775-784.

[51] V. Seshadri, G. V. Zaruba and M. Huber, "A Bayesian Sampling Approach to In-Door Localization of Wireless Devices Using Received Signal Strength Indication," *Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, IEEE Computer Society, 2005, pp. 75-84

[52] N.B. Priyantha, A. Chakraborty and H. Balakrishnan, "The cricket location-support system," *null*, 2000, pp. 32-43.

[53] R.J. Orr and G.D. Abowd, “The smart floor: a mechanism for natural user identification and tracking,” *CHI '00 extended abstracts on Human factors in computing systems*, The Hague, The Netherlands: ACM, 2000, pp. 275-276.

[54] S. Yeh, “GETA sandals: a footstep location tracking system,” *Personal Ubiquitous Comput.*, vol. 11, 2007, pp. 451-463.

[55] A. Gaffar, D. Sinnig, A. Seffah and P. Forbrig, “Modeling patterns for task models,” *Proceedings of the 3rd annual conference on Task models and diagrams*, Prague, Czech Republic: ACM, 2004, pp. 99-104.

[56] E. Furtado, V Furtado and J. Vanderdonckt, “KnowiXML: a knowledge-based system generating multiple abstract user interfaces in USIXML,” *Proceedings of the 3rd annual conference on Task models and diagrams*, Prague, Czech Republic: ACM, 2004, pp. 121-128.

[57] Q. Limbourg, B. Michotte, L. Bouillon, M. Florins and D. Trevisan, “UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces,” *in Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages*, 2004, pp. 55-62.

[58] J. Farringdon, “Wearable Sensor Badge and Sensor Jacket for Context Awareness,” *Proceedings of the 3rd IEEE International Symposium on Wearable Computers*, IEEE Computer Society, 1999, p. 107.

[59] S. Lee and K. Mase, “Activity and location recognition using wearable sensors,” *IEEE Pervasive Computing*, vol. 1, 2002, pp. 24-32.

[60] Seon-Woo Lee and K. Mase, “Recognition of walking behaviors for pedestrian navigation,” *Control Applications, 2001. (CCA '01). Proceedings of the 2001 IEEE International Conference on*, 2001, pp. 1152-1155.

[61] L. Bao and S.S. Intille, “Activity recognition from user-annotated acceleration data,” 2004, pp. 1-17.

[62] N. Kern, B. Schiele and A. Schmidt, “Multi-sensor activity context detection for wearable computing,” *In Proc. EUSAI, LNCS*, vol. 2875, 2003, pp. 220-232.

[63] Zekeng Liang, Ioannis Barakos and Stefan Poslad, “Indoor Location and Orientation Determination for Wireless Personal Area Networks,” *MELT 2009,* 2009, pp. 91-105

[64] K. Van Laerhoven, A. Schmidt and H. Gellersen, “Multi-sensor context aware clothing,” *Wearable Computers, 2002. (ISWC 2002). Proceedings. Sixth International Symposium on*, 2002, pp. 49-56.

[65] K.K. Khedo, “Context-Aware Systems for Mobile and Ubiquitous Networks,” *Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies*, IEEE Computer Society, 2006, p. 123.

[66] A. Schmidt, “Ubiquitous Computing- Computing in Context,” 2002.

[67] R.B. Doorenbos, “Production matching for large learning systems,” Carnegie Mellon University, 1995

[68] B. Margolis, *SOA for the Business Developer: Concepts, BPEL, and SCA (Business Developers series)*, MC Press, ISBN-978-158347-065-7