Bluetooth Hotspots for Smart Spaces Interaction

Mestrado de Engenharia Informática

Universidade do Minho

Advisor: Prof. Helena Cristina Rodrigues

Miguel Craveiro Martins de Almeida

October, 2011

Acknowledgements

I express my deep gratitude to Professor Helena Cristina Rodrigues for her guidance, valuable scientific support, helpful advises and for her constant patience.

I also would like to thank other members of the Ubicomp team, at Universidade do Minho, namely Prof. Rui José, Arlindo Santos and Bruno Silva for their support, which was very important in all the steps of this dissertation.

I am thankful to Pedro Farinha for his precious help in part of this work, particularly in what concerns the development of the Wiimote Extension's prototype.

I wish to thank Mariana Pulido and my friends Afonso Arriaga, Mário Pinhal, and my brother Nuno for all their support, enthusiasm and friendship and for helping me revising this document.

Thanks are still due to Departamento de Sistemas de Informação of Universidade do Minho for all facilities.

Finally I would like to thank my parents and brothers for all their familiar permanent support. Without it this dissertation would not be possible.

Abstract

In scientific literature and industry, mainly in the area of Pervasive Computing and Smart Spaces, a variety of applications and systems may be found which are based on both explicit and implicit user interactions with physical resources in the environment, such as Wi-Fi spots, GPS receivers, Bluetooth components, RFID readers, mobile phones or cameras. Bluetooth is a short-range wireless technology which is present in a wide number of handheld devices and requires zero configuration. By this mean, it becomes a powerful tool to be used for interaction with physical environments and consequently has been adopted by different applications and systems as a privileged interaction technology. Currently, those are implementing their own Bluetooth components that perform application-specific tasks, like getting information about user devices, sending or receiving files.

In this dissertation, we argue that Bluetooth components may be managed as interaction resources in the physical space, which can be shared and reused by different third-party applications and systems. This situation would free application developers from Bluetooth management-related issues and would allow them to focus in application objectives. Additionally, our approach may also contribute for the sustainability of the Pervasive Computing industry from an environmental perspective, as it will enable sharing and reusing of physical resources and potentially reducing the number of local computing devices.

In this work we studied relevant projects and applications for Pervasive Computing and Smart Spaces that use Bluetooth an interaction mean. Common characteristics of these projects are identified - an important step to systematize those user interactions. This work is a basis for the design of a system component Bluetooth resource. Prototypes were developed and deployed in multiple real scenarios to validate, not only the feasibility of using such a component on Smart Spaces, but also the integration model with applications. Such validation is presented on this document and works as a proof of concept for this component.

Resumo

Tanto na literatura científica como na indústria, especificamente na área da Computação Disseminada e Espaços Inteligentes (*Pervasive Computing and Smart Spaces*), são encontradas muitas aplicações e sistemas baseados em interacções (implícitas e explícitas) com recursos físicos no ambiente, tais como pontos Wi-Fi, receptores GPS, componentes Bluetooth, leitores RFID, telefones móveis ou câmaras. Bluetooth é uma tecnologia sem fios de curto alcance, que não requer configurações e que está presente num elevado número de dispositivos móveis. Tornou-se, assim, numa poderosa ferramenta para interacção com ambientes físicos, sendo consecutivamente adoptada por diferentes aplicações e sistemas como uma tecnologia priveligiada de interação. Actualmente estas aplicações e sistemas implementam os seus próprios componentes Bluetooth, capazes de executar tarefas especificamente relacionadas com as aplicações em causa, tais como a obtenção de informação sobre os dispositivos dos utilizadores ou trocas de ficheiros com estes.

Nesta dissertação argumenta-se que os componentes Bluetooth podem ser tratados como recursos de interação do espaço físico, com a possibilidade de serem partilhados e reutilizados por diferentes aplicações e sistemas. Desta forma liberta-se os seus programadores de questões relacionados com a implementação e gestão específicas da tecnologia Bluetooth, incentivando-os a focarem-se nos objectivos da aplicação. Além disso, a nossa abordagem poderá também contribuir para a sustentação da indústria da Computação Disseminada sob uma perspectiva ambiental, dado que irá permitir a partilha e reutilização de recursos físicos, reduzindo o número de dispositivos computacionais.

Neste trabalho estudam-se projectos e aplicações relevantes para a indústria da Computação Disseminada e Espaços Inteligentes que recorrem à tecnologia Bluetooth como uma forma de interacção com os seus utilizadores. São identificadas as características comuns destes projectos, tratando-se de um passo importante para a sistematização destas interacções. Este trabalho constitui a base do desenho de um novo recurso Bluetooth. Foram desenvolvidos e instalados protótipos em vários cenários reais, a fim de se validar não apenas a viabilidade de tal componente em Espaços Inteligentes mas, também, o modelo de integração com as aplicações. Tal validação é apresentada neste documento, funcionando como uma prova de concceito desta componente.

Contents

Chapter 1

Introduction

Pervasive Computing is as a new paradigm for computing systems, where computation is spread through everyday objects able to communicate with each others and the users [39]. One of the main characteristics and challenges of Pervasive Computing systems is the physical-virtual integration: the increasingly pervasive presence of all sorts of sensors, such as Wi-Fi spots, GPS receivers, Bluetooth components, Radio-Frequency IDentification (RFID) readers, mobile phones or cameras, in our physical world. The capability to network those sensors and infer information from their data is greatly expanding the ability of the virtual world to perceive the physical world and react accordingly.

The existence of all sorts of those sensors in our physical world, have promoted the emergence of different trends in Pervasive Computing. Context-aware computing was the main trend deriving from this characteristic in the sense that context-aware computing is a field on which software applications behaviour is defined by contextual information. *Such context-aware software adapts according to the location of user, the collection of nearby people, hosts, and accessible devices, as well as to changes to such things over time* [4, 21, 31]. Smart Spaces, as public spaces augmented with local computation [28, 23], have been a test bed for experimenting smart-meeting rooms [24], smart-houses [22], smart-classrooms [3, 6] or smart-museums [10] applications. Work on Reality Mining [16], Urban Computing [33, 35] or Mobile Social Software [15] has broadly explored available sensors in the physical space and related sensing capabilities for *applying and refining methods of observing, recording, modelling and analysing the city, physically, digitally and socially* [33]. In marketing, mobile advertising has broadly explored pervasive computing environments for delivering permission-based location-aware mobile advertisements, and other content, to mobile users [2, 36, 38].

Focus on the Bluetooth technology

Bluetooth is a short-range wireless technology which is presented in a wide number of handheld devices and requiring near zero configuration. By this mean, it became a powerful tool to be used for the interaction with physical environments and has consequently been adopted for different applications and systems, as a privileged technology of interaction [11]. In fact, many of the different cited works in the area of Pervasive Computing have broadly explored and evaluated the use of Bluetooth technology as a mean for implicit or explicit interaction with users. Particularly José *et al* [27, 26] and Davies *et al* [14] have explored an approach to support user interaction with public displays based on the Bluetooth Device Names. A different kind of Bluetooth interaction, based on OBject EXchange (OBEX) file exchanging, is used [38, 9, 37] to disseminate contents to mobile users. A mean for intuitive interaction using the Nintendo's Wii remote controller equipped with a Bluetooth interface is presented by various authors [18, 42, 40, 41]. Other types of interactions are oriented to a connection that is established between the user device and some remote application or service, using the Bluetooth infrastructure (a Bluetooth interface or Access-Point available on a space) as an intermediary or broker between these two end-points.

We argue that Bluetooth components may be managed as interaction resources in the physical space, which can be shared and reused by different third-party applications and systems. This situation would set application developers free from Bluetooth management-related issues and allow them to focus on application objectives. Additionally, our approach may also contribute for the sustainability of the pervasive computing industry from an environmental perspective, as it will enable sharing and reusing physical resources and potentially decreasing the number of local computing devices.

1.1 Objectives

The main scope of this dissertation is the proposal of a new system component centered on Bluetooth-based user interactions, to be used on systems and applications centered on situated applications. The focus of the work is on these interactions. Our goal is to design more than just an interface with the user: we want this component to be self-contained and able to be easily installed, configured and used - the concept of a small box that, after being installed, becomes a resource available for different applications. With this component, it should be easy to create diverse interactive spaces, which uses

Bluetooth as a mean of interaction. For instance, as a support for visits to museums, thematic parks or entertainment in urban computing.

In the area of Pervasive Computing, many works are being developed using this mean of interaction, which has lead us to establish three main objectives for our work:

To study the user interactions of relevant Bluetooth-based systems. This study aims to analyze patterns in user interactions. Deeply understanding how related works are implementing their own Bluetooth interfaces is crucial to systematize our work. This is the first step towards the implementation of the next two objectives. Furthermore, this study is relevant by itself, as it offers a state-of-the-art survey for works centered on user-interaction over Bluetooth.

To design the architecture of a Bluetooth Hotspot. We aim to specify an architecture for a new system component that works as a bridge between Smart Spaces applications and user's Bluetooth devices. Our intention is to gather on a single component the possibility of performing all the interactions that the actual Bluetooth interfaces are currently implementing, in a dedicated way: 1) application's developers become free from Bluetooth-related issues and 2) physical spaces may become free from numerous Bluetooth devices when multiple applications share the same physical place. Generally, we call this component a Bluetooth Hotspot. The final objective is to develop a fully-capable prototype that runs on Linux-enabled Ethernet routers.

To propose an integration model for Bluetooth user interaction systems. The integration of our Hotspot with situated applications is based on a model designed to support the Bluetooth-related interaction requirements of those applications. This model is designed in order to serve multiple applications that may coexist on the same physical space, in the most optimized way possible.

1.2 System overview

The environment that evolves our Bluetooth Hotspots is composed by four essential entities, illustrated in Figure 1.1. They are 1) the set of user's Bluetooth devices, 2) applications that expect some kind of user-centered interaction using the Bluetooth technology, 3) Bluetooth Hotspots and 4) a Main Controller that manages the Hotspots.

User's Bluetooth devices. Any kind of device equipped with Bluetooth can be included on this set, although cellphones or Personal Digital Assistants (PDAs) are

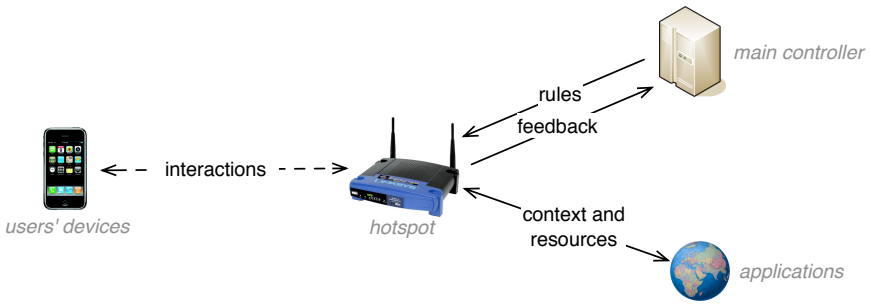

Figure 1.1. An abstraction of the main components of the system designed.

the most usual ones. These devices act as a user's terminal, which exchanges information with the rest of the infrastructure.

Applications. Applications are pieces of software that perform user-centered tasks based on some kind of interaction with users over Bluetooth technology. Despite usually running on a computer, the physical location where these applications run is irrelevant for the whole system, if they can be reachable on an Internet Protocol (IP) network. A typical application is the one running on a public display that shows useful information (such as news, videos or photos) and expects the input from users, with suggestions of content. In this case, the public screen acts as an output device for the application while the user Bluetooth device acts as an input device.

Bluetooth Hotspots. Hotspots are small boxes which act as a bridge between user devices and the applications. In order to interact with users, these Hotspots perform Bluetooth tasks such as detecting the user's devices presence, retrieving their device's name, receiving or sending files, among other similar tasks. These boxes are equipped with embedded devices (Ethernet routers running OpenWrt Linux[1]) which run the software that was developed based on the architecture proposed in Chapter 4.

Main Controller. The Hotspots work in a stand alone mode thanks to a list of rules which defines their behaviour. For each Hotspot, this list can be previously installed (turning it on a standalone Hotspot) or managed in real-time by a

[1]OpenWrt (`www.openwrt.org`) is a Linux distribution optimized to run on embedded devices like Ethernet routers. This distribution have the ability to run efficiently even with low resources like slow CPU (e.g: 100MHz), short memory (e.g: 16MB of Random Access Memory (RAM)) and tiny storage capacity (e.g: 1MB).

central Main Controller. Thus, this component is responsible for managing the Hotspots, by telling them how to behave.

If multiple Hotspots are deployed (for example, to cover a wide space) they need to cooperate with each other, otherwise their behaviours can be randomly erroneous (e.g, when a file is sent by a Hotspot to a specific user device, it should not be sent again by a second Hotspot that may also reach the device). In this kind of scenarios the cooperation work between the Hotspots is responsibility of this Main Controller[2].

The operation of this component is beyond the scope of this work. Its architecture is not covered by this document. Nevertheless, the way it is integrated with the Bluetooth Hotspots and Applications is addressed.

The list of rules describes the Hotspots' behaviour, allowing them to be deployed and to run autonomously. The list of rules is composed by a set of identifiers. Each identifier can be followed by some parameters, as shown in the following example[3]:

```
- scan(scaninterval=30, getnames=true, getservices=true)
- postsighting(url=http://app1.com/postsightings.php)
- acceptfiles(devclass=CellPhone, url=http://app2.com/postfiles.php)
- deliverfile(url=http://app2.com/welcome.txt, devaddress=00:A5:BF:*)
```

In this example, the list of rules tells the Bluetooth Hotspot to: 1) scan for the presence of devices every 30 seconds, retrieving their human readable names and the list of which services are available; 2) for each scan, send the information above to an application available at a specific Uniform Resource Locator (URL); 3) accept files from devices (only cellphones) and send them to a remote server, available at the specified address and 4) also send a welcome file to all the devices detected, which addresses start with `00:A4:BF`.

1.3 Methodology

Along this document, our research is reported in three main steps, always aiming to achieve the stated objectives (detailed in Section 1.1):

Related work. The first step of this work was a survey on relevant projects that use the Bluetooth technology as a mean of interaction with users. Our objective

[2]The architecture of the Main Controller is not described on this dissertation as it is not the focus of our work. Only the protocol of communication between it and the Hotspot is designed and detailed.

[3]The example is presented in YAML format, exactly as it is used by Hotspots and applications.

is twofold. First we intend to survey related projects in order to describe the different requirements and architectural approaches for supporting Bluetooth interactions. Then we intend to identify common characteristics across different scenarios and systematize the main types of Bluetooth interactions. This is the basis for defining the main interactions between the whole group of system components, such as the Bluetooth Hotspot, the client applications, the web resources and the controller.

Proof-of-concept prototypes. In a second phase, we have deployed basic prototypes of the Bluetooth Hotspot in order to evaluate technical decisions that have been made, such as the choice of the target hardware and the development language. The prototypes were deployed together with on-running projects that already use Bluetooth interactions. This evaluation is done in the context of the on-running projects at Departamento de Sistemas de Informação[4], such as Instant Places [27].

Fully capable prototype for strong validation. In a third phase, a more complete and robust prototype was developed and deployed (see Chapter 5) to validate not only the whole designed architecture but also the proposed integration model with client applications. Several Bluetooth Hotspots were deployed and configured in multiple scenarios, re-used for different types of interactions and shared by different client applications. Performance data were collected to report the case-studies.

1.4 Structure of the dissertation

This document is structured in five chapters: Chapter 1 justifies this work, resumes it and briefly explains how it is organized in different phases. Chapter 2 includes a survey about related work on relevant studies, applications and projects. We intend to identify needs and requirements, focusing in the context of Bluetooth interactions. Chapter 3 presents a more detailed analysis of the patterns of interactions identified in the previous chapter and describes the architecture developed for the proposed system component. The design of the system component we propose and how it can be implemented constitutes Chapter 4. The validation of this architecture is achieved with prototypes and their development. This validation is described in Chapter 5.

[4]Mobile and Ubiquitous Systems Group, Universidade do Minho - `http://ubicomp.algoritmi.uminho.pt/`

Chapter 6 presents the most relevant conclusions and proposes future work, derived from the work developed here.

Chapter 2

Related work

In this chapter we survey relevant work in the area of Pervasive Computing that strongly builds on user interaction based on Bluetooth technologies. The objective is to understand what kind of interactions with users are being used by Smart Spaces applications, identifying patterns on those interactions. This survey supports decisions about the design of the Bluetooth Hotspot we are proposing. The identification and study of these patterns will also justify the design of a modular architecture for this component.

Based on the research, we grouped those kinds of user interactions in five main sets: 1) simply scan for the present devices 2) get the name of user's devices 3) send or receive a file to/from user's devices 4) gesture recognition using the Nintendo's Wiimote accelerometer and 5) connection-based interactions.

Due to its relevance and based on the analysis of Chapter 3, most of these interactions will be natively integrated on the architecture of the Bluetooth Hotspot that is proposed in this document. Many other interactions could be included (like print or audio over Bluetooth) but they were excluded as they does not show, currently, to be relevant for the context of Smart Spaces and Pervasive Computing.

A section for each kind of interaction we have studied is presented below. Each section briefly introduces how that kind of interaction is often used and then present some of the most relevant works, studies, systems and applications for our research. For each of them we will detail as much as possible, not only the interactions over Bluetooth between the user and the infrastructure, but also the main system components and the most relevant issues about their system architectures.

2.1 Device scan interaction

The Bluetooth Inquiry Protocol enables a device to discover other devices in the proximity, usually to later establish some type of connection for data transfer. This procedure - called the Inquiry Procedure [1] - consists in broadcasting Inquiry Packets and then waiting for a response from the other devices in the neighbourhood. All devices that are listening to those packets will reply with an Inquiry Reply packet that includes their own hardware address - identical to an Ethernet Media Access Control (MAC) address. This allows the inquirer to uniquely identify the inquired device and later reach it to establish a connection.

In order to be able to receive Inquiry Packets, devices should enable the Inquiry Scan State mode. In this situation, the device is in the discoverable state. Failing to enable the Inquiry Scan State mode, Bluetooth devices are hidden from other devices, even if the Bluetooth radio is enabled.

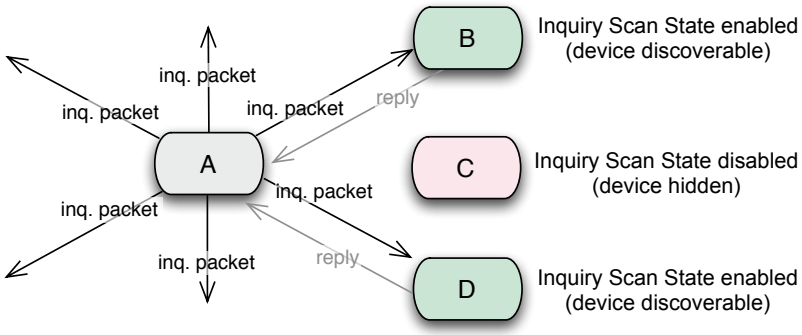

Figure 2.1. Essential steps of an Inquiry Process where three devices are present, two of them are discovered and one is hidden.

Figure 2.1 illustrates a situation where a device (A) searches for the present devices. Despite existing three devices in the neighbourhood, only two (B and D) will be discovered, due to their responses with Reply Packets.

Cityware applications

Many works like [33, 35, 32] use the Bluetooth Inquiry Protocol for proximity detection of users without any other objective than the detection itself. The objective of such work is to collect the maximum amount of data concerning the presence of users in urban spaces, like shoppings, schools or car parks, or to elaborate the notion of *social context* based on the analysis of interpersonal proximity data collected in a particular

situation. Bluetooth scanners [33] are installed in different places of a city, periodically scanning for the presence of devices and storing information about them. Despite this paper is not specifying technical details about how this scanner is implemented and how the data is stored, it does specifies which data is collected for each device: 1) the hardware address, which is unique and identical to an Ethernet MAC address), 2) the human-readable name, which is a name usually set by the user for easy identification of the device, 3) the Bluetooth device class[1] and 4) the list of services available on the device (e.g. OBEX push, modem, fax).

Car traffic monitoring

Device discovery is also used with other purposes like vehicle traffic monitoring for statistical purposes. One in twenty vehicles have at least one Bluetooth-enabled device inside it [43]: a cell phone, a GPS receiver, an headset or even the car itself. As seen in Figure 2.2, with multiple Bluetooth scanners distributed along a high-way or road, it becomes possible to calculate the average speed of a car on that road section, if the car is observed by two scanners. The Center for Advanced Transportation Technology of University of Maryland[2] has developed a box which can be deployed on the street, which periodically searches for devices around it and stores the collected data internally, for posterior download and analysis. The work presented in [5] describes an identical system that uses a Wi-Fi network to access the Bluetooth nodes (device scanners), and remotely retrieve the Bluetooth collected data.

2.2 Device name-based interaction

Each Bluetooth device can handle a short human-readable name for easier identification. Bluetooth Human Interface Devices (HIDs) like audio headsets, Bluetooth GPS receivers, mouses or keyboards are generally assembled with a static name. However, computers and handheld devices like PDAs, cellphones and smartphones allow the user to configure its Bluetooth name. A typical situation where this name is useful is when a user wants to transfer a file to another device or connect to an audio headset: 1) first, the user device scans for the presence of other devices (situation identical to the interaction of the previous section, 2.1); 2) then, for each device found, the name is

[1]The Bluetooth device class is a number that corresponds to the type of device, e.g: a cellphone, a laptop, a desktop computer.

[2]Center for Advanced Transportation Technology of University of Maryland - `http://www.catt.umd.edu/`.

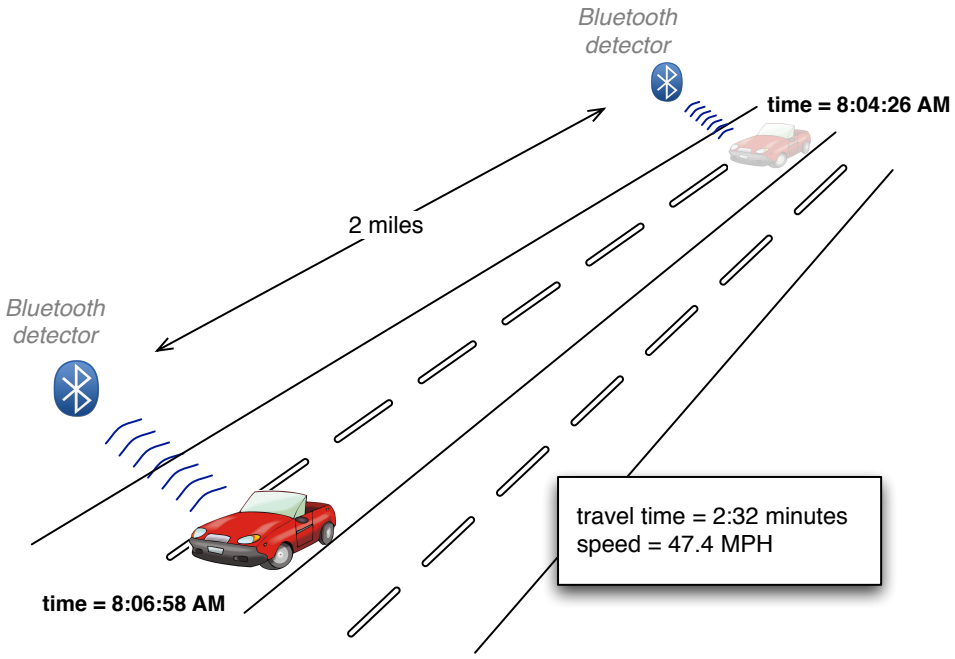

Figure 2.2. Multiple Bluetooth scanners along a road to determine the average speed of cars with a Bluetooth device inside it. Image adapted from [43].

obtained and shown on the screen for the user to select which one to connect.

Some works [14, 27, 7] in the area of Pervasive Computing use this kind of interaction not only for proximity sensing but also to obtain information about users. The idea is to use the device name to suggest information on the context of some activity. The user just needs to set the name of its device to some text that is recognized by the system. With this method the user can deliver a simple text to the system without installing any kind of application, turning on a easy to use and cross-platform method.

Name abbreviation

A problem inherent to this method is the limited space available of the device name. Despite of the Bluetooth specification limit of 248 bytes with `UTF-8` encoding for the device name [1], most of the device brands strict to their own limit - sometimes below 32 characters. This turns abbreviation almost an obligation [14]. This problem was a motivation for the creation of Bluetooth Extended Naming (BEN) [26] which describes a method, and its implementation, to turn Bluetooth device names into a command line interface for interactions with near systems. They propose a syntax[3] [13] that

[3]Bluetooth Extended Naming - `http://ubicomp.algoritmi.uminho.pt/ben/`.

allows users to describe user-related information, using a low number of characters. For example, to share their interests on specific news topics (e.g: `tag.radiohead`), to vote on polls (e.g: `vote.braga`) or to share their ID of services like Last.fm[4], Youtube[5] or Flickr[6] (e.g: `id.flk.mary`).

InstantPlaces

In a public space - like a bar, a school or a hotel hall - public displays can be installed containing news, weather, advertising, quizzes/polls, photos or other kind of useful information for that places. Instant Places is a project that explores how Bluetooth can be used for interaction with public situated displays [27]. Users can suggest what kind of information they are interested to see on the screen, vote on a poll or provide personal information to show on the screen like their own name. This is done by setting the human-readable name of their Bluetooth-enabled devices using the abbreviation syntax of the BEN [26] technique. Each display is installed with a Bluetooth scanner, which is constantly getting information about the present user devices. This cyclic procedure consists of two phases: 1) a phase of scanning (using the Bluetooth Inquiry Procedure, already described in Section 2.1) which results on a list of hardware addresses and 2) a phase of name discovery for each seen device on the first phase, resulting on a list of strings. After each cycle of scanning the list of addresses and names is delivered to the service that fills the screen with corresponding information.

In what concerns the Bluetooth-related components, the Instant Places architecture - represented in Figure 2.3 - is based on three main components: 1) the public displays software, which renders content to users, 2) the Bluetooth scanners and 3) a central server that manages displays, scanners and the context of the information that is displayed. Despite these scanners currently being deployed as a piece of software running inside the same computer that controls the displays, the system was designed to later support the Bluetooth scanner as an autonomous component, running, for example, on a small computer or a embedded device. In this case, the Bluetooth server component could be reused by different applications running on the same physical space

All these components communicate over an IP network, following a RESTful paradigm [17]: 1) scanners periodically send the result of each scan to the server and 2) displays periodically query the server for content to be displayed. All this information

[4]Last.fm - `http://www.last.fm/`
[5]Youtube - `http://www.youtube.com/`
[6]Flickr - `http://www.flickr.com/`

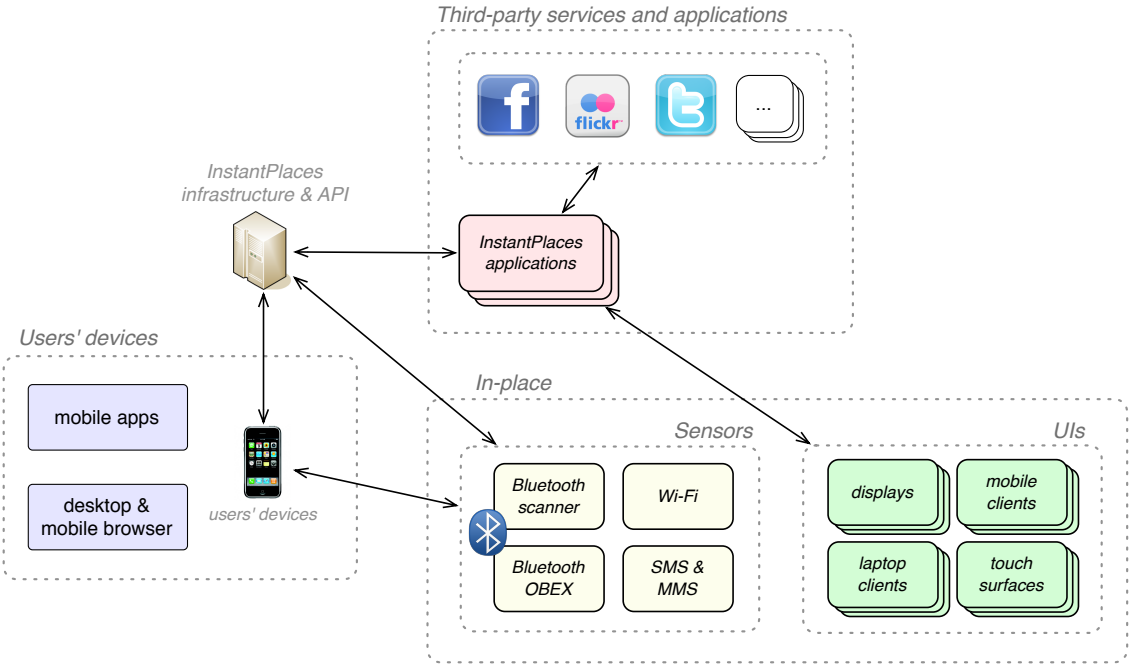

Figure 2.3. An abstraction of InstantPlaces architecture. Image adapted from technical documentation of Instant Places.

is exchanged on Extensible Markup Language (XML) and sent to the server over an Hypertext Transfer Protocol (HTTP) POST packet.

Bluemusic

In the Bluemusic system [30], the Bluetooth device name is used to personalize a public environment where users may suggest artists or musics to be played just by changing the Bluetooth name of their devices to specific tags. *If there are more than one user in the Bluetooth proximity area, the system identifies the preferences shared by most of the users.* This paper briefly introduces other possible scenarios that can be based on the same kind of interaction, like adjusting the temperature of the office by user suggestion (or by suggestion of a group of users) or suggesting news for a public screen. In all the above scenarios, users are able to interact with system without the need to install an application on their devices.

The Bluemusic software runs on a computer equipped with a Bluetooth interface (an internal interface or a Universal Serial Bus (USB) dongle) and running the prototype software. Bluemusic periodically scans and retrieves the name of Bluetooth devices in the proximity, parses them for the name `Bm+` followed by the name of the

chosen music and considers to play it.

e-Campus

In [14], Nigel Davies *et al.* have proposed Bluetooth based interaction communication with public displays. These displays present information from services search results from Google[7], videos from Youtube[8] or photos from Flickr[9]. Using a technique similar to the one presented in [26], users submit commands to the display over their device names. To interact with the display, users define their device's name to tags with the search term they are interested, like `google <search term>` or `youtube <search term>` or `flickr <search term>` and wait for the results to be shown on the public display.

Another application available on this system is an Audio Jukebox, identical to the previous work (Bluemusic): users may suggest the music they want to be played on that place, just by defining their device's name to `juke <song id>`. This action adds the song associated with that id to the queue of musics that will be played.

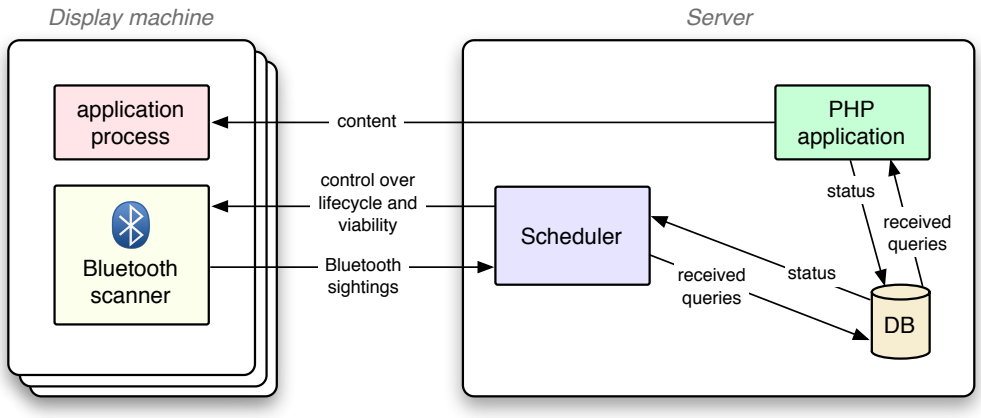

Figure 2.4. System architecture of e-Campus. Image adapted from [14].

Many prototypes of these screens are deployed in the Lancaster University[10] (in the United Kingdom) on a network of screens which they call the e-Campus. The architecture that supports this system - represented in Figure 2.4 - is identical to Instant Places, with a central server that manages multiple screens and each screen is equipped

[7]Google search engine - `http://www.google.com/`
[8]YouTube video service - `http://www.youtube.com/`
[9]Flickr photo stream - `http://www.flickr.com/`
[10]Lancaster University, in UK - `http://www.lancs.ac.uk/`

with a Bluetooth scanner. For each display, the scanner - a piece of software running on the same machine of the display - constantly searches for the present Bluetooth devices and collects its current name. This information is sent to the central server using a low-level Application Programming Interface (API). The server decides what information is to be shown based on the data collected by the scanner and creates a "playlist" of data to be shown on that screen. Following a publish/subscriber messaging pattern, the screen periodically consults this queue and shows the available information to users. This way, scanners act as input devices for users and screens as output devices for the system.

2.3 File exchange-based interaction

Bluetooth is often used to exchange files between devices like computers, PDAs, cell-phones and smartphones. These transferences are usually performed over the Bluetooth OBject EXchange protocol (OBEX).

When a user wants to send a file to another device, for example from a phone to a computer: 1) the phone scans for other Bluetooth devices using the inquiry mechanism [1] - mechanism already described in Section 2.1 and 2) it obtains the name of each device, exactly as described in the previous interaction (refer to Section 2.2). Then, 3) the user selects the device to which to send the file. The phone queries the computer about the services it has available, in order to know the availability of the OBEX file transfer service. If the service is available, 4) the phone tries to send the file to the computer using the channel associated to the OBEX service.

Some works in the area of Pervasive Computing are based on the idea of content delivery in other situations than the standard user-to-user one. The delivery of files to user devices in the proximity can be used for delivering informations in public spaces, like cities or museums, for tourists, or even for commercial purposing or advertisement [29]. With a Bluetooth-based infrastructure that is constantly aware of all the Bluetooth devices, it becomes possible to deliver files to all of them (or just to specific ones) [37, 38].

2.3.1 Content-delivery applications

In this section we describe some works in the area of Pervasive Computing, based on proximity-sensing for content delivery, that we consider relevant for this research.

BlueMall

BlueMall is a Bluetooth-based advertisement system for marketing purposes on large commercial areas, *delivering information based on the clients' current location* [38]. This paper describes a system architecture based on three software entities: 1) client mobile devices, 2) BlueMall Access-Points (APs) and 3) a central server. This server is responsible for the management of Bluetooth APs, maintaining the state and context of every action of the whole system. APs have the ability for: 1) constantly searching for Bluetooth devices and 2) delivering files to those devices.

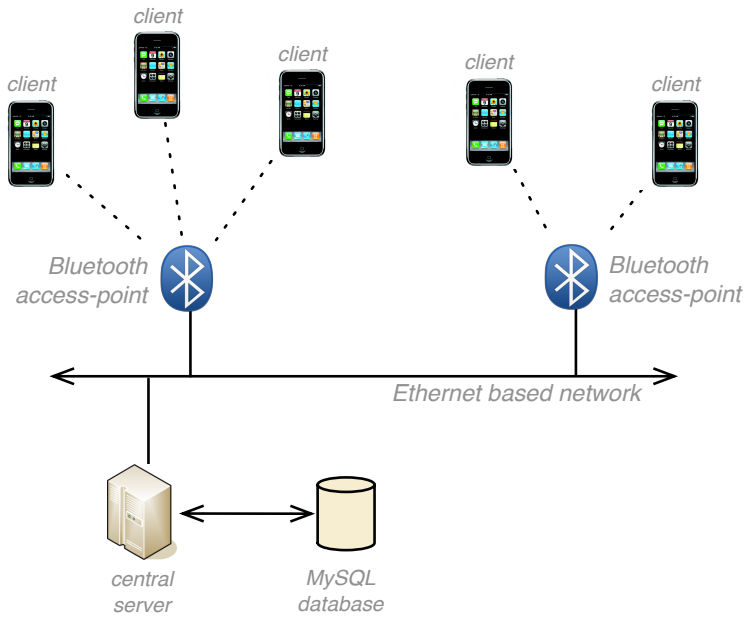

Figure 2.5. An abstraction of Bluemall architecture for content-delivery. Image adapted from [38].

As represented in Figure 2.5, BlueMall APs are configured by the server using a XML file containing variables like the AP location, time elapsed to consider a client's visit to be a new visit, server's address and the amount of time to ignore a device until it is considered as a new visit. After configured, the AP starts scanning and delivers to the server the list of device sightings on each scan. Every time a device is seen (identified by its hardware address) the server's database is updated, also registering if the `OBEX Object Push` service is available. Knowing such information speeds up further scans since there is no need to ask the device for the service availability again.

To avoid flooding users with files (which can be very negative if it leads the user to turn off his device) BlueMall takes care of "what" information is sent to "whom" and

"when". The system also supports a white list of devices not interested in receiving files, intended to be filled with the MAC addresses of the employees' devices. *Experimental results show that this system provides a viable solution for permission-based mobile advertising* [38].

Bluegiga access devices

In [38] the author refers the utilization of a commercial component - the Bluegiga's APs - as an alternative for the BlueMall APs. Bluegiga Technologies Inc.[11] is an enterprise that develops Bluetooth-based products. The most relevant ones are the "Bluetooth Access Point" and the "Bluetooth Access Server". Both products are embedded devices that perform Bluetooth tasks oriented to the industry, like: eHealth, Point of Sale (POS), proximity marketing and Internet sharing. These APs are designed to be installed on public spaces and to act as an infrastructural anchors to be connected to devices like medical metering devices on a clinic or an hospital, or to handheld POS devices on restaurants or cafes. They can also be used for content delivery applications for marketing purposes, delivering files to devices present near the APs.

The significant difference between the Bluegiga Access Point and the Bluegiga Access Server is the number of connections they can handle. While the first one is equipped with just one Bluetooth interface - it can keep 6 simultaneous connections with 6 different devices - the second one is equipped with three interfaces, allowing 18 simultaneous connections. Both Bluegiga devices are equipped with an RJ-45 Ethernet port to connect to a TCP/IP network, but the second one is also equipped a Wi-Fi card and an USB port, allowing it to be equipped with a General Packet Radio Service (GPRS) or a 3G USB card.

Bluegiga's APs software is implemented with the Linux operating system, and offers its own Software Development Kit (SDK). This gives the possibility for developing specific applications for specific scenarios. However, basic commons applications are natively supported, like:

OBEX sender. With this application the AP can deliver files to groups of devices. Constantly scanning for the present devices, the AP will deliver a file (locally or remotely stored, somewhere on the network) to all devices that match with the groups' filter (depicted in Figure 2.6). These groups are defined with filters for: device class, name, hardware address and even the distance from the AP to the device. All events are logged for further analysis.

[11] Bluegiga Technologies Inc. - `http://www.bluegiga.com/`

OBEX receiver. The AP may also receive files over Bluetooth, locally storing them or uploading them to a remote server.

SPP-over-IP. Connections for specific tasks may be performed over this protocol. It allows to establish an end-to-end connection between the user's devices and a remote server, accessed over TCP/IP. With this protocol, the AP acts like a bridge between these two ends.

PAN. The AP can be configured to share an Internet connection to Bluetooth devices, over the Bluetooth Network Encapsulation Protocol (BNEP), extending a TCP/IP network. With this protocol, the AP accepts connections from a group of devices or establishes it when a specific device is seen near it.

Bluegiga's APs can be remotely configured over the network using a Secure Shell (SSH) console or using a Graphical User Interface (GUI) served over HTTP. However, it is designed to be configured interactively by the administrator (via console or via GUI). A RESTful integration would be extremely important to improve its reusability.

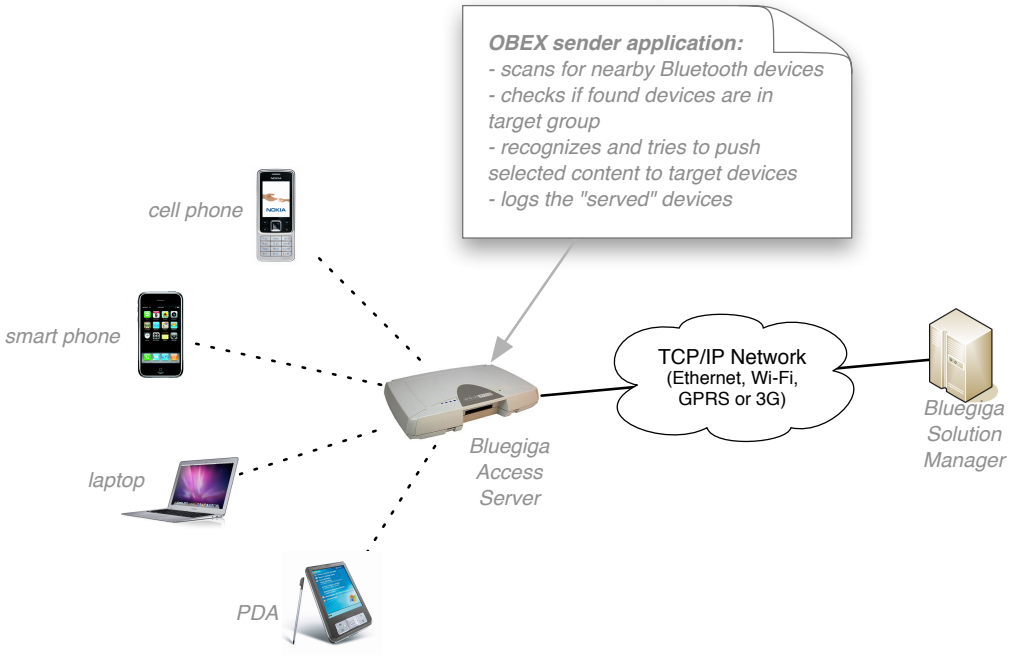

Figure 2.6. A network of devices connected to Bluegiga's Access Server by its built-in "Obexsender" application. Image adapted from Bluegiga's User Manual.

Location-based advertisement

In the work proposed in [37], identical to Bluemall, Omer Rashid *et al.* propose a system for content distribution based on the location of users, using Bluetooth. Figure 2.7 depicts an abstraction of this system. With "Bluetooth push servers" (Bluetooth spots capable of delivering files to users) installed on a space and aware of the presence of devices it becomes possible to deliver one or more files to potentially interested users. In this paper, they propose a system based on multiple Bluetooth spots and a central server that manages the decisions of delivery and the state of the whole system. The spots are constantly scanning for the present devices and sending this list to the central server. For each scan, the server checks if the file(s) was already sent to each device. If it was not delivered yet, the server contacts the Bluetooth spot again and gives order to deliver it. However, if it was already delivered, the server will wait for a pre-specified amount of time until trying to deliver the file again to that device. This allows to deploy multiple spots without the problem of having a user roaming from a spot to another and receiving the same file multiple times.

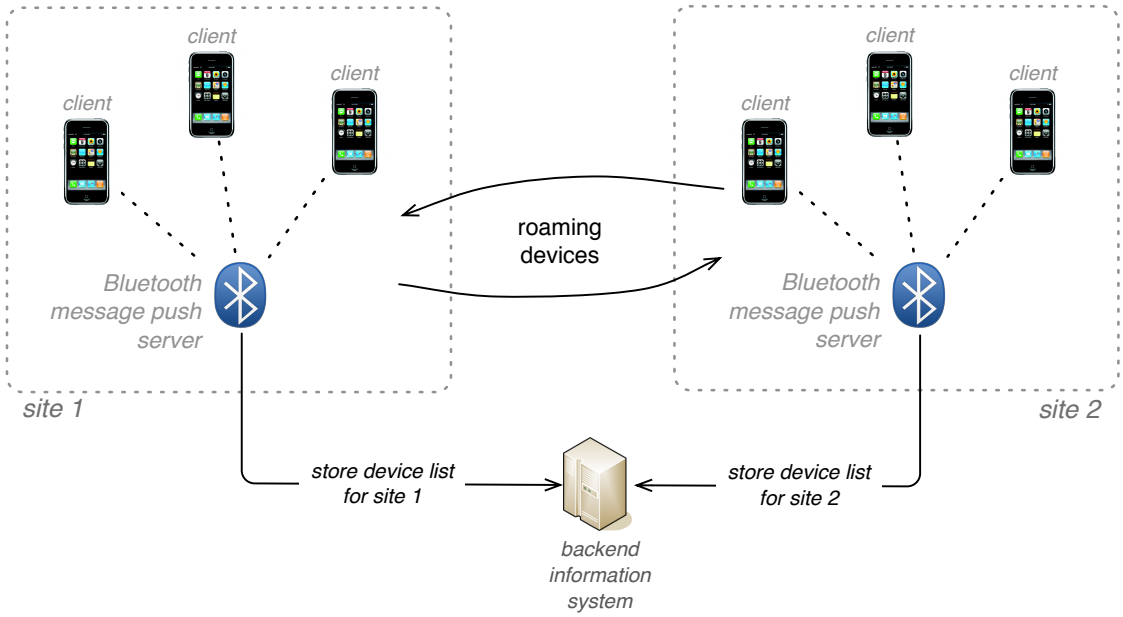

Figure 2.7. Roaming devices switching between different Bluetooth sites. Image adapted from [37].

OpenProximity

OpenProximity[12] is an open-source application oriented for Bluetooth content-delivery, identically to Bluemall and Omer Rashid's project. However, OpenProximity uses a single standalone computer to deliver files instead of dedicated Bluetooth APs. The computer owns the Bluetooth interface but also acts as the state manager, keeping information about which files were sent to which devices. It can be configured graphically via an HTTP interface or configured via Remote Procedure Call (RPC) requests.

OpenProximity works with at least two Bluetooth interfaces: one interface is used to constantly scan for Bluetooth devices and the other(s) interface(s) are responsible for delivering files. The software supports up to 15 Bluetooth interfaces simultaneously installed, sending 7 files at the same time with each interface. This means that, theoretically, it can handle up to 105 connections with 105 different devices at the same time.

With multiple configurations, multiple files can be delivered by OpenProximity for different contexts. The configuration of a file to be delivered is called a "campaign". If a campaign is created without any filter, the file is delivered to all present devices through the available OBEX service. However, filters can be defined to create time restrictions (when to start and when to end the campaign) or filters for devices. Devices can be filtered by name, class (type of device) or hardware address.

There are identical commercial products on the market, like BlueMarket[13], BlueSender[14] and Fexmax[15].

Bluestation

In his master thesis, Tiago Camacho proposes a system with Bluetooth stations capable of scanning for the presence of devices and delivering files to them [9]. A Bluetooth station is a computer running the "Bluestation" software, pre-configured with a list of rules containing the description of the station behaviour. These rules allow to filter the list of seen devices by hardware address and to define time restrictions before delivering the files. This makes possible to configure the stations to deliver different files to different devices in different moments. Each Bluetooth station have at least two Bluetooth interfaces installed: 1) one that is constantly scanning for the present devices and 2) the other(s) are reserved for sending files to user devices using the OBEX service.

[12]OpenProximity - `http://www.openproximity.org/`
[13]Bluemarket - `http://www.bluemagnet.com/`
[14]BlueSender - `http://www.bluesender.com/`
[15]Fexmax - `http://www.fexmax.com/`

This work also details many different tests done with the stations deployed in many different situations (in the university bar, in bus stations or shops). Despite the focus of our work not being the acceptance of this interaction by different publics, the work of Tiago Camacho becomes valuable as it determines the viability of content delivery using Bluetooth. It also raises important issues for the design of our component, which will be explained in Chapter 4.

2.3.2 Both-way file exchange

Keith Cheverst *et al.* propose a system [12] that uses Bluetooth as a mean of interaction with a photo display (Hermes Photo Display). Users can 1) submit photos to the display using their handheld devices (typically mobile phones) or 2) download photos that are being displayed, submitted by other users (touching them on the screen) - both using the OBEX service.

The system is composed by four main components: 1) a Linux server that implements the Bluetooth functionality of delivering files and receiving files to/from devices 2) a shared file space for storing the photos to display 3) a presentation server that generates the form how pictures and which pictures will be displayed and 4) a presentation client - the photo display. This display connects to the presentation server via a wireless network and presents the photos. The screen of the photo display is touch sensitive to allow simple interactions with users.

2.4 Connection-based interaction

The Bluetooth technology is used by many Human Interface Devices like mouses, keyboards or audio headsets to connect to user's computers or handheld devices. Other devices like printers, game console controllers or remote controllers also use Bluetooth to communicate. This type of communication is usually performed over the RFCOMM, Serial Cable Emulation Protocol (RFCOMM) protocol (a transport layer protocol) which emulates an `RS-232` serial port. Being a reliable protocol, it easily supports the development of Bluetooth applications as it allows the creation of data sockets identical to IP sockets, over this protocol. However, not all the applications want a reliable connection, like audio or real-time applications. So, may also connect over a lower layer protocol called Logical Link Control and Adaptation Protocol (L2CAP) (which is the link layer protocol that supports RFCOMM). Many other protocols can be implemented for many application-specific tasks, depending on the context of the

application in question.

The most common scenario of a Bluetooth connection is a connection between devices of two different users. Other typical scenario is a connection between two devices of the same user. However, some works are based on applications that use a connection between the user device and an infrastructure. During the time that the user device is present in a place, it becomes possible to communicate with the infrastructure until the user leaves the space. We will now focus on two works that we consider to be relevant because they represent a set of identical applications oriented to this type of interaction.

LectComm

LectComm is an open-source software used to support lecturers during classes [6]. Both the lecturer and students install the software on their devices. Students run a client version of the software (or access a web site with an on-line version) where they may answer questions and quizzes made by the lecturer, who uses a server version of the software.

The communication between the client and the server is made over TCP/IP. As most students do not have access to the Internet on the classroom, LectComm provides a Bluetooth connection, with an extended version of the client software application. With this application, students are able to interact with the lecturer's server software using a socket over a Bluetooth connection.

The paper also describes the possibility to use a "generic Bluetooth AP". The paper does not detail what is in fact a generic Bluetooth AP, but it describes it as a component which maintains a connection with the clients, that may be deployed on a class room without the need of a complex infrastructure or a dedicated computer. The only requirement is a Ethernet connection or wireless link to connect the Bluetooth AP to the Internet.

BluetunA

The Human Connectedness Research Group[16] at MIT has developed "tunA" - an application that runs on PDAs and allows users to share music with other nearby users that run the same application on their PDAs. The application uses a Wi-Fi ad-hoc network to reach the nearby tunA's users. Each one can see the other's profiles,

[16]Human Connectedness Research Group - `http://web.media.mit.edu/~stefan/hc/`

consults their playlists and listen what they are listening over a peer-to-peer audio streaming.

Stephan Baumann *et al.* developed BluetunA [8], an application identical to tunA that uses Bluetooth to reach the other users, instead Wi-Fi ad-hoc networks. Despite the application being designed to run on handheld devices and to connect to other devices, the paper briefly introduces the concept of a BluetunA Hotspot (under development). The idea with this Hotspots is to provide more reliability to the system and to allow the implementation of situated applications. There are no details about how this Hotspot is implemented, but for our research we just need to focus on the idea that a Bluetooth connection is established between the Hotspot and every user device that runs the BluetunA software.

Internet connection sharing

Being a short-range protocol, Bluetooth is widely used with personal devices like mouses, keyboards, audio headsets, PDAs, cellphones and computers. It is possible to interconnect most of these devices on an Ethernet-like network with IP addressing. This is possible due to BNEP - a protocol that allows the creation of Personal Area Network (PAN) between Bluetooth devices [25]. It also becomes possible to establish a network not only between user devices, but also between user devices and an infrastructure of Bluetooth Hotspots. This protocol is useful to share an Internet connection to Bluetooth-enabled handheld devices that are not equipped with Wi-Fi.

The Bluegiga's Access Points and Access Servers, already described in Section 2.3.1 also offer this service to the nearby devices.

2.5 Wiimote-based interaction

In 2006, Nintendo Co., Ltd[17] released the Wii, a new home video game console, differentiated from similar consoles by its innovative wireless remote controller - the Wiimote. Beyond the classic game controller's arrow and auxiliary buttons, this controller is equipped with Bluetooth technology to communicate with other devices, four light-emitting Diodes (LEDs), a vibrator, a small speaker and two sensors that improve the user interaction: 1) An Infrared (IR) sensor enables the controller to perceive its location relative to two IR emitting diodes installed above or bellow the television and 2) an accelerometer that measure peaks of movement on three axes `(x,y,z)`.

[17]Nintendo Co., Ltd - `http://www.nintendo.com/`

The Wiimote is represented in Figure 2.8 - *providing the basis for computer gesture input and recognition* [18]. The recognition of well defined gestures like squares, circles, rolls or other types of similar movements makes the Wiimote a powerful tool for many applications that support human interaction based on gestures [40].

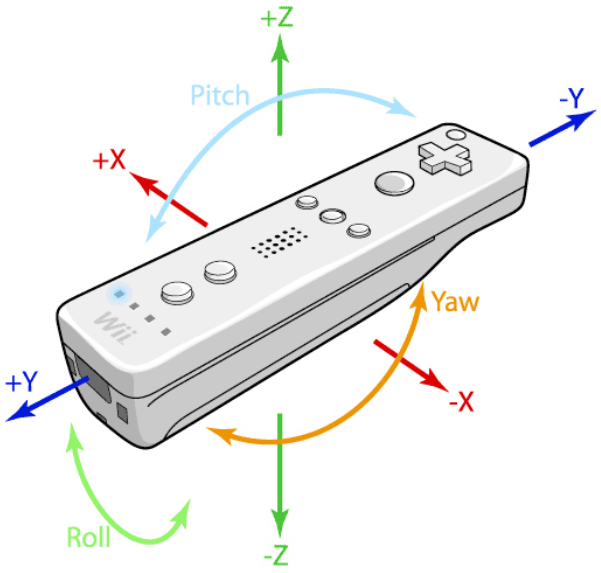

Figure 2.8. The three axes and three rotation movements that can be detected by the Wiimote. Image source: Osculator `http://www.osculator.net/`.

When a Wiimote is turned on, it enters on the Inquiry Scan State, which means that the device can be detected by other devices (situation already described in Section 2.1). At this moment, the controller also announces the service `Nintendo RVL-CNT-01` and starts listening for new connections. At the same time, the Wii console, which is periodically searching for Wiimotes (devices with the name `Nintendo RVL-CNT-01`), will find it and establish a serial-emulated link with it using the RFCOMM protocol. From this moment on, every event that occurs on the controller - a variation on any accelerometer axis, a variation on the IR sensor position or a button that is pressed - is sent over this link to the console. On the other hand, the controller also accepts three kinds of input from the Wii console [41]: 1) enable/disable each of four LEDs, 2) enable/disable the internal vibrator that is used to give feedback to the person who is holding the controller and 3) send a short sound to be played on a small speaker installed inside the controller.

As Wiimote communication is based on Bluetooth technology, as opposition to the use of a proprietary protocol, it can be used with any Blutooth-enabled software component. The work in [41] introduces a new way of interaction in Smart Spaces using

Wiimote as a controller for public displays. Holding a Wiimote, users can navigate through pages of information shown on the screen using the arrow buttons. The paper focus on the following scenario: an interactive museum with multiple displays installed around, showing useful information about the artworks and offering users the possibility to bring their Wiimotes from home and interact with those displays.

The prototype of the application that runs on the displays - implemented in Java - constantly scans for Wiimotes and establishes a connection with each device it discovers. This application only interprets inputs from the controller's buttons. However, gesture recognition (accelerometer) and motion tracking (IR sensors) can be added in the future, using gesture recognition libraries like Wiigee[18].

2.6 Discussion

All the works covered by this survey use the Bluetooth technology as the main interaction mean between users and applications available on a particular space. To support this interaction, system architectures are heavily based on a Bluetooth component that is installed at the same physical space as users. This interface runs on a dedicated hardware device (usually called a Bluetooth node or Hotspot [38, 8, 27, 33, 37, 9]) for some systems and as a software that runs on the same machine that controls the place's applications for others [14, 30, 27, 6, 41]. Generically we will call them just Bluetooth nodes.

Each system we have surveyed implements its own Bluetooth nodes, just for their purposes and not designed for an optimized reusability environment. Moreover, these nodes (either software-based or hardware-based) become tightly-coupled with the remaining infrastructure and can hardly be reused by other applications that share the same physical place. Even wider featured solutions like the commercial Bluegiga (see Section 2.3.1) raise the problem of being hardware-dependent, based on proprietary hardware and software and not oriented for a coherent and autonomous integration as they need to be manually configured by the administrator, via a console or a GUI.

There is no study of integration models between Bluetooth components and the remaining architectural components. The focus of our study goes to the specification of a loosely-coupled architecture, oriented to user interactions and capable of turning these Bluetooth nodes on a new reusable system component that can be shared by multiple different applications. To support such architecture, this new Bluetooth nodes

[18]Wiigee (`http://wiigee.org/`) is a Java-based gesture recognition library for the Nintendo's Wii remote controller.

are 1) designed with support for the most scenarios as possible, 2) allowing their configuration at deployment time and 3) defining application-independent protocols for communication with the other infrastructure components. In order to achieve a fundamented design of such architecture, a systematized analysis of the survey of this chapter will be presented on the following one.

Chapter 3

Analysis for Hotspot design

This chapter reports the analysis made over the survey of the previous chapter, in the context of Bluetooth-based interactions. We have identified the relevance of a Bluetooth Hotspot system component. In this analysis, we are aiming at justifying and supporting on decisions for the design of the Bluetooth Hotspot component.

Firstly, in section 3.1, we systematize the types of interaction that may happen between this system component and the users, grounded by the related work surveyed on the previous chapter. Now, with a new perspective, we assume that the Applications' original Bluetooth components are now replaced by the Bluetooth Hotspot. This chapter is organized in four sub-sections that correspond to four groups of Bluetooth interactions that share identical characteristics. We will focus, not only on interactions with users but also on interactions with applications that will be using this new system component.

Secondly, in section 3.2, we identify relevant key design issues for the architecture of the Bluetooth Hotspot, gathered from the analysis of Section 3.1 and from the survey presented in Chapter 2.

Thirdly, in section 3.3, we report the most relevant Bluetooth-related issues collected from the survey in what concerns the system scalability, contributing for a more sustained implementation.

3.1 User-interaction patterns

With this analysis, our objective is to identify types of Bluetooth interactions between users and the system components. In short, the common components widely present in all systems or projects we have studied are:

- User's devices that are able to or waiting to interact with an application or with a system near them, in the same physical space;
- A Bluetooth interface (one or more) that is able to perform Bluetooth tasks, acting as a bridge between users and applications;
- Applications willing to interact with users, over Bluetooth.

Usually, those Bluetooth interfaces appear integrated with the applications in cause, anchored to specific tasks. When multiple applications share the same space with users, those interfaces could be replaced by a common one. The interface would be shared between applications and would be reused by future applications that use the space. If we make an analogy with the term "Wi-Fi Hotspot" used for Wi-Fi APs that are publicly available, and because this component becomes available to the present devices on a space, we will call it, from this moment on, a "Bluetooth Hotspot".

The Bluetooth Hotspot as a new Bluetooth component

In this chapter we will instantiate previous user-interaction patterns in our high-level architecture. The main components of our architecture are: 1) the users' devices 2) the Bluetooth Hotspot and 3) the applications that use the Hotspot as a Bluetooth resource. During this analysis, original characteristics of the Bluetooth components will be preserved. New characteristics are only introduced from a management point of view: those we understand to be essential for Hotspot sharing and reusability. This step turns on to be the most important step in the design of the Hotspot architecture, later presented in Chapter 4.

We have grouped Bluetooth interactions in four different types: 1) obtaining the address and name of present devices, 2) sending a file to a device, 3) receiving a file from a device, 4) to establish a connection with a device

Assuming that the original Bluetooth components are now replaced by the Bluetooth Hotspot, each of these types of interaction will be detailed in the next four sections. The last two types (4 and 5) are grouped in the same sub-section.

3.1.1 Getting the address and name of a device

Probably the most important type of interaction - as it is the most common, used in all scenarios - is the detection of a Bluetooth device. All the applications that require to know the present devices have to periodically scan the environment. This

process may include a phase for obtaining the devices' name or simply their hardware address. In some cases, obtaining the available services (e.g: OBEX Object Push, Dial-Up Networking, Hands-Free Audio Gateway) of each device is also required, as for example in the case of file transfers (see next sub-sections).

We have identified the common phases during this type of interaction: 1) In the configuration phase, the Bluetooth Hotspot is configured for periodically scanning for present devices; 2) In the second phase, the Bluetooth Hotspot is responsible for getting the address and Bluetooth names of present devices; and finally 3) In the third phase, the Bluetooth Hotspot is responsible for sending device information to the applications. This scenario is illustrated in the sequence diagram of Figure 3.1.

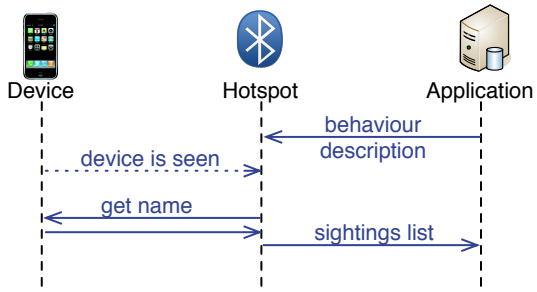

Figure 3.1. Sequence diagram illustrating interactions of a Bluetooth Hotspot that searches for the present devices, gets its names and send this information to a remote application.

3.1.2 Sending a file to a device

In content delivery scenarios, files are sent to user devices as soon as they are detected. BlueMall [38] or OpenProximity are examples of applications that fit in this type of scenarios.

For these scenarios, we envisioned the following interactions, as illustrated in Figure 3.2: 1) Applications should configure the Bluetooth Hotspot with the location where to get the files - an HTTP or File Transfer Protocol (FTP) location - and the list of devices to which send the files, commonly called a "white list". 2) Every time a device from this list is present 3) the file must be downloaded from its URL. Simultaneously, the device is inquired about the availability of the OBEX service. If available, 4) the file is sent to the device using the OBEX protocol.

To be able to keep track of the Hotspot behaviour, the application requires feedback about success or failure of these tasks. So, logs should be reported to the application

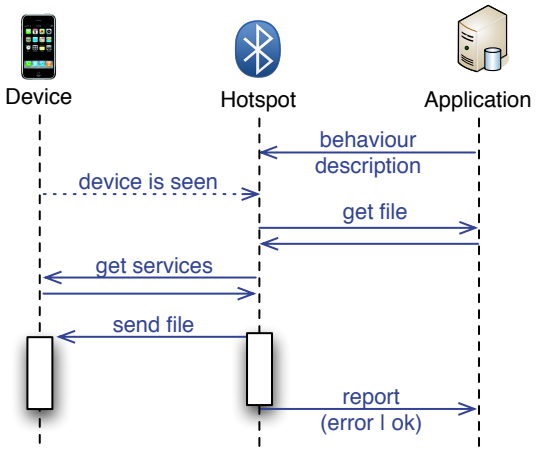

Figure 3.2. Sequence diagram illustrating interactions for a Bluetooth Hotspot that sends a file from an application to a Bluetooth device.

(e.g: inform the application about a downloaded file, a file successfully sent to a device or an error downloading or sending a file to a device).

3.1.3 Receiving a file from a device

Despite being a less usual scenarios, some applications may also receive files from mobile devices. In this scenario, users send a document to applications which then process it, same as showing an image on a public display.

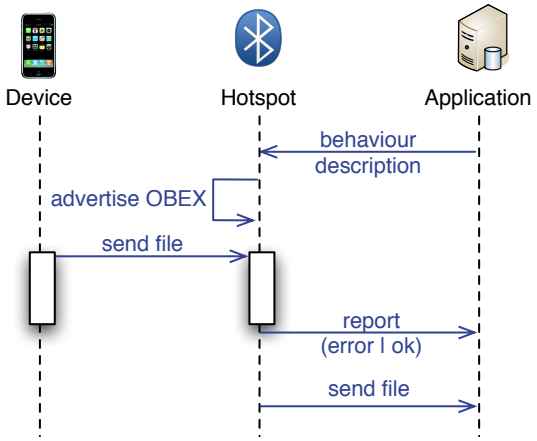

Figure 3.3. Sequence diagram illustrating interactions for a Bluetooth Hotspot that receives a file from a device and uploads it to an application.

As illustrated in Figure 3.3, this type of interaction can be divided in the following

phases: 1) Applications should configure the Hotspot with the list of devices allowed to send files (white list) and with URL where to store the received files. 2) The Hotspot advertises the OBEX service and starts listening for new connections. 3) Every time a user intends to send a document to an application, the mobile device scans for the appropriate OBEX service (previously advertised) and initiates the documents OBEX transference. 3) When the Hotspot receives a document and forwards it to a specific predefined URL.

Again, the feedback about the success or failure of these tasks should be reported to the application (e.g: received file, sent file, error receiving or error sending a file).

3.1.4 Establishing a generic connection

We have described in Section 2.4 and 2.5 other Bluetooth-based interaction patterns which mainly rely on the establishment of a generic connection between the mobile device and the application. On top of such connection, application-specific interaction patterns are implemented by both ends of the connection.

The establishment of a connection may be either initiated by the mobile device or the Hotspot, on behalf of the application.

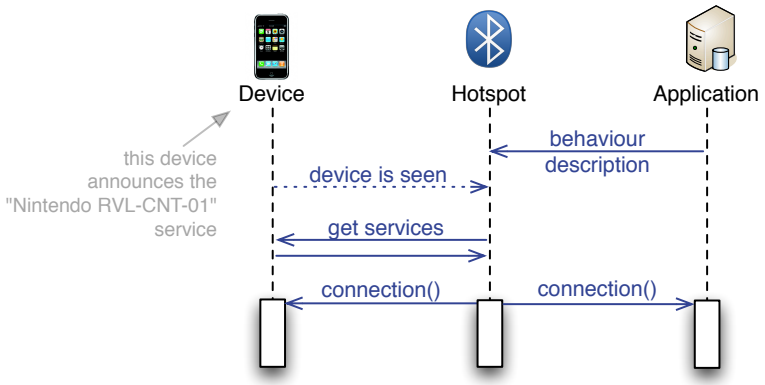

Figure 3.4. Sequence diagram illustrating interactions for a Bluetooth Hotspot that establishes a connection with a device.

Figure 3.4 depicts the establishment of a socket connection over an RFCOMM connection between the Hotspot, on behalf of an application based on Wiimote interactions. This scenario was described in Section 2.5 and will be used as a prototype in the validation phase.

The establishment of the connection will follow the following steps: 1) Applications

configure the Hotspot to scan for present devices and to establish a RFCOMM connection. 2) For every scanned device, the Hotspot inquires the device for the available services: 3) If the Wiimote service (`Nintendo RVL-CNT-01`) is available, the Hotspot establishes a RFCOMM connection with the device and 4) a socket connection with the application. 5) Once the connections are established, the Hotspot acts as a bridge between the mobile device and the application.

The application should be informed about the success of fail of any task performed, during the entire process.

Connection establishment from device side

As depicted in Figure 3.5 (a), in some scenarios, the connection establishment may be started from the device side instead from the Hotspot side. In this case, the device is not discovered by the Hotspot, but the Hotspot should previously advertise the service and listen for incoming connections. This scenarios follow the following steps: 1) the Hotspot is configured to advertise a specific service and to accept connections from devices for that service. 2) Every time a user intends to establish a connection with the Hotspot, the mobile device scans for the appropriate service (previously advertised) and initiates the connection 3) When a new connection arrives to the Hotspot on the service in question, the Hotspot accepts it and also establishes a concurrent socket connection with the remote application that interprets the information. 4) After established the both connections, the Hotspot redirects the data received from the Bluetooth connection to the application connection and vice versa.

Interaction's context independent of applications

Some situations does not require the application to manage the context of the interaction. In this kind of scenarios, the connection is only established between users and the Hotspot and not with the application. A typical situation is for sharing an Internet connection with a device over BNEP to establish a PAN connection, like Bluegiga does (refer to Section 2.3.1). Figure 3.5 (b) illustrates this type of scenarios, where: 1) The Hotspot should be configured to accept BNEP connections. 2) When a device tries to establish a connection with the Hotspot 3) a new PAN connection must be established between the Hotspot and the device.

The application should only be informed about the result of this action, if succeeded or failed, but does not receive any content from the device.

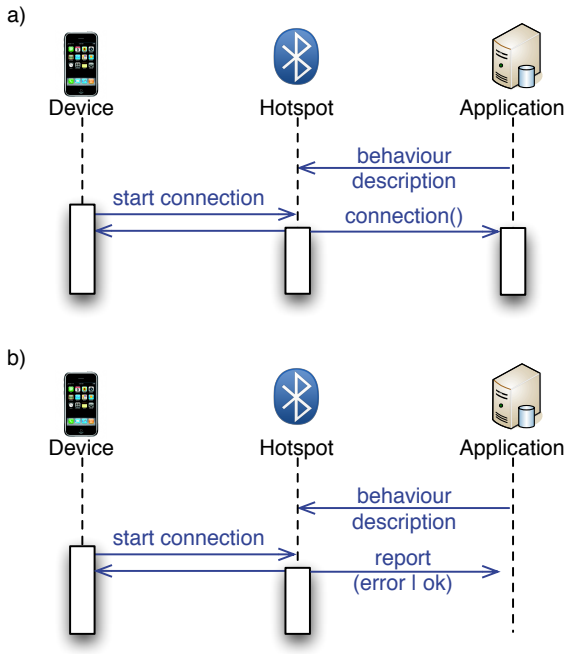

Figure 3.5. Sequence diagram illustrating the existing interactions for a Bluetooth Hotspot that accepts incoming connections over RFCOMM: a) managed by an application or b) managed by the Hotspot.

3.2 Key design issues

In this section, we will enumerate and describe the main key design issues for the design and development of the Bluetooth Hotspot system component. Our work is based on the analysis of the main interaction patterns described in the previous sections.

3.2.1 Integration

The Bluetooth Hotspot component should be designed as a new space resource, which may be integrated in client applications offering different services for that space. Its function is to manage Bluetooth interactions between mobile devices and client applications. The processes of integration must, as much as possible, existent interaction requirements in order to minimize any changes to existing interaction models. We will now enumerate the most relevant issues for the process of integration.

Hotspot's behaviour description

In order to achieve the correct interactions at the right moment, accordingly to the applications requirements, applications need to somehow configure their Bluetooth interfaces. In [14, 33, 30, 43] Bluetooth components are pre-configured to send the list of present devices to a remote server on each scan. In [9], a list of rules is used to describe 1) when and 2) what to deliver 3) to which devices. An extended version of this type of rules must describe the behaviour of the Hotspot to achieve those interactions identified as being the most relevant (refer to Section 3.1).

If we replace current Bluetooth interfaces with our Bluetooth Hotspot, those rules will need to be described and uploaded to the Hotspot by applications or by some other component acting as a controller. This controller must configure the Hotspot with such rules, but also retrieve reports generated by the Hotspot to be up to date with every task performed by it. These reports include the time when a task was performed, an identification of the task and extra related information. Because multiple applications may be sharing the same Hotspot, from now on we will assume the controller as an autonomous component focused on controlling the Hotspot.

As this component controls the Hotspot we identify it as "The Main Controller", or simply "The Controller".

Device Sightings

Different systems surveyed are based in different kinds of Bluetooth interaction with users. However, almost all of them refer to the existence of a "device scanner" - a component that periodically searches for the present Bluetooth devices and then collects more or less information about them (depending on the objective of the applications). This information can be as simple as the device hardware address or more complex as the human-readable name, the class (type of device) or the available Bluetooth services on each device. For each scan, the list of present devices is referred in some articles as a "Sighting" [27, 14]. A Sighting can be locally stored [43] or sent to a remote site over the network [27]. In this case, this action is usually performed through the serialization of an XML file that contains all the information collected about the discovered devices.

Download and upload of files from/to devices

Scenarios of content delivery or content pull deal with files which are sent or received to/from user's devices. These files can be locally stored [9] inside the Hotspot or remotely available over the network [38]. This means that a file can be identified by an

URL, both if it is locally or remotely stored. In this case, before being sent to a user's device, it will be downloaded over the network and temporarily stored on the Hotspot to be sent. Also the reverse happens, when the Hotspot receives a file from a user's device: the Hotspot receives the file and stores it. If the URL is a remote location, the file is sent over the network to the appropriate application and then it is removed from the Hotspot.

3.2.2 State management

In most of situations, a Hotspot is enough to cover a space of interaction. However, to cover larger physical spaces, multiple Hotspots may be deployed. In this scenarios it may be required the Main Controller to manage the context between all the Hotspots. A task executed by a Hotspot may require any type of synchronization actions with other Hotspot in the same space. Typically, this situation is observed in scenarios where a Hotspot delivers a file to a device and all the others must assume that the file should not be delivered again to the same device. For example, when a user enters a museum equipped with two Bluetooth Hotspots and an informative file is delivered: the Hotspot A detects the device and successfully sends it a file. Then, the user moves across the museum and a Hotspot B also detects his presence, but the file is not delivered because it already has been sent (by the Hotspot A). As depicted in Figure 3.6, after the file being delivered by the Hotspot A, the Main Controller is informed and configures the Hotspot B to ignore the device. The next time any of the Hotspots detect the device, it is ignored and the file is not delivered.

3.2.3 Extensibility

In connection-oriented scenarios, the context and information shared with users depends on the application, differing from application to application (as analyzed in Section 3.1.4). This means that it is important to design a system capable of serving the most situations as possible. Because it is not possible to foresee all the possible scenarios, it makes sense to have an extensible architecture, allowing further implementation of other application-specific interactions. With such mechanism, it becomes possible to the application developers to implement extensions that achieve their tasks, to be installed on the Bluetooth Hotspots. One extension of this type will then responsible for maintaining a connection with a device and another with the corresponding application, turning the Hotspot in a bridge between both of them.

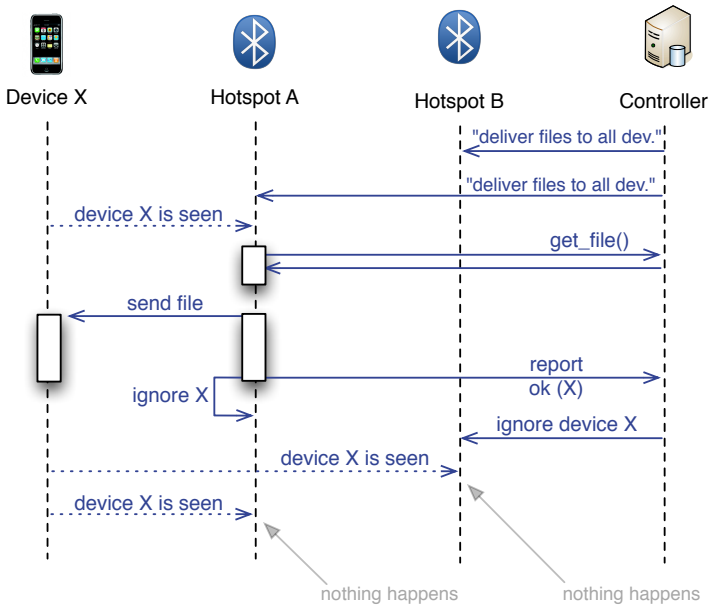

Figure 3.6. Sequence diagram illustrating a situation where two Hotspots cover a space. A file sent from a Hotspot to a user's device, but is not sent again by any of the two Hotspots.

When to start an extension

From the analysis of surveyed scenarios we identify that a Bluetooth interaction may be initiated in one of two situations: a) when a device is detected by the Hotspot or b) when a user tries to establish a connection with the Hotspot. The same pattern may be applied to the extension mechanism. If the Hotspot is installed with an extension, the extension may start in one of those two situations:

The Hotspot detects a device. When a device is detected by the Hotspot's scanner, the extension starts (if not yet started) and is informed about the detected Bluetooth devices, along with all the information collected about them (hardware address, device name and available services). With such information, the extension becomes able to behave accordingly to the desired objectives (e.g. to establish a connection with all the present devices).

A device establishes a connection with the Hotspot. In order to accept incoming connections from Bluetooth devices the Hotspot must be previously configured to be listening to these connections. So, if these connections are handled by an extension, the extension must be previously started, in order to be listening. Moreover, the Bluetooth services must also be announced in order to allow the user's devices to reach the Hotspot.

The extension behaviour

As described above, the behaviour of an extension will depend on the application that is using it. To be able of serving multiple applications, extensions have to be configured by them, in order to achieve the desired behaviour. However, the structure of such configuration completely depends on the extension and application. Extension configuration should then be of responsibility of applications. Applications are responsible for managing and store a configuration file, which is fetched by the Hotspot at the the moment the extension is initiated.

3.3 Bluetooth-related scalability issues

During our review of existent systems, we have identified a set of Bluetooth-related technical issues, which largely influence the technical behaviour of Bluetooth components and thus, also the technical behaviour of the Bluetooth Hotspot and Hotspot-based prototypes. In this section we enumerate those issues we understand to be more relevant. All the items questioned here will later be taken into account when designing the architecture of the Bluetooth Hotspot, reported in Chapter 4.

3.3.1 Scanning frequency

The three processes related with the scanning of Bluetooth devices (scanning for devices + retrieve their names + retrieve their services) may introduce unexpected performance issues if not performed with appropriate durations and times to timeout [34, 20]. On the one hand, if scans are performed too often in a short period of time, the majority of those scans may not find a new device. On the other hand, if we decrease the scan frequency, we may be creating an usability problem, as it may lead the user to give up and probably to quit and probably leave the space.

Limit the duration of full scan

Another situation that can produce unwanted behaviours is the presence of a large number of devices: if the Hotspot detects a large number of devices, and further executes the inquiry process for every device, the total duration of the scan process may increase too much and fail to meet the performance requirements of the system. This can lift the same usability problem referred above, leading the user to quit. This means that it is important to define a maximum time for each scan, independently of

the number of devices. At least, the list of all devices should be created and submitted to the application, even without some device names. This will give the opportunity to the application to use, at least, the number of devices present and their hardware addresses. Further information may be then submitted in subsequent scans.

The previous scenario is exemplified in Figure 3.7, with two hypothetical scanners:

- In the first situation (a), the scanner is not configured with any limit of time for scanning. Thus, the duration of a full-cycle scan[1] becomes unpredictable, because it is not possible to know how much time it will be needed to obtain the name of each device. In the example of Figure 3.7 (a), the scanner detects 7 devices but it only obtains the name of 5 of them. The process of getting the name of the other two devices (marked with an asterisk, in Figure) times out. A device Sighting (refer to Section 3.2.1) with the list of all detected devices may only be generated after 20 seconds (time=20).

- In the second situation (b), independently of the time needed to process the whole scanning process, this is aborted when the "maximum scanning time" is reached, even if the process is not complete. In Figure 3.7 (b), if we consider again the situation where 7 devices are detected, a device Sighting that contains the list of all devices and the names of three of them is generated after 10 seconds (time=10). The name retrieval of the fourth device (marked with an asterisk) is canceled and a new scan is immediately started.

Example: As an example we will assume a router that is constantly scanning for Bluetooth devices and then retrieving the Bluetooth name and available services of each one, with a full-cycle scan period of 20 seconds (in this case, for a full-cycle scan we understand the whole process of inquiry + name retrieval + services retrieval). This means that a new scan should be performed at least every 20 seconds. However, as the inquiry process takes around 10 seconds [20], the scanner will have only 10 seconds more to retrieve the names and get the available service of all devices. If it takes more than 10 seconds to perform these steps, the period of a scan will not be met.

Hypothetically, if retrieving the name of each device takes 3 seconds, as well as obtaining their available services, when a scan is executed and 6 devices are present,

[1]For "full-cycle scan" we understand the full process of getting the list of discovered devices + get the names of all devices

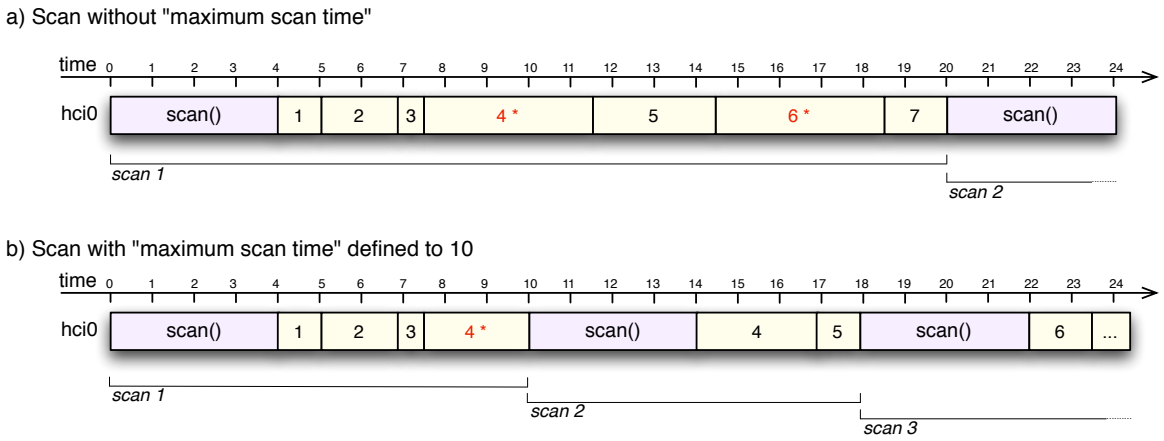

Figure 3.7. Comparison between a scanner with a maximum time for scan defined (b) and a scanner without it defined (a).

the whole process would take around

$$10 + 3 * 6 + 3 * 6 = 46 \ seconds$$

which is greater than the expected 20 seconds. To avoid this, a maximum scan time should be defined to 20 seconds: a new scan will be performed each 20 seconds, independently of the number of devices detected and the success of name retrieval and services retrieval. If there is remaining data to obtain from devices, after the end of a scanning period, that data will be retrieved in the following scan.

Conclusion

There is not a perfect solution that fits all kinds of scenarios. On the one hand, the scan period can be hardly affected if a large number of devices is present, but their names and services can be retrieved on time. On the other hand, to ensure that the scan period is respected (configuring a maximum time for scan) the performance of retrieving the devices' names and devices' services will be hardly affected. The decision of choosing one situation or the other will depend on the scenario. However, if multiple Bluetooth interfaces are installed on the same Hotspot, these tasks can be performed in parallel: if two interfaces are available, one can be used for scanning while the other can be used for retrieving device names, speeding up the entire process. This situation is analyzed below.

3.3.2 Multiple Bluetooth interfaces

Some real scenarios may involve a large number of simultaneous tasks on the same Hotspot. However, there are some technical limitations that can deteriorate the operation of the Hotspot. The performance of a Bluetooth connection will depend on issues like wireless interference, available bandwidth or limitations of the protocol. While implementing the Bluetooth Hotspot we do not have any control on the first one, as it depends on many factors like the distance between the Hotspot and user's devices or the presence of other radio devices interfering with the Bluetooth communication. Also the second one - that evolves management of bandwidth - is not a concern because the Linux Bluetooth stack manages the available bandwidth between all the Bluetooth connections established. However, there is a limitation that we must care about: the Bluetooth specification limits the number of connections with other devices to 8 connections [1]. This means that if we want to keep more than 8 connections at the same time, the Hotspot will need to handle more than one Bluetooth interface. Using more than one interface is also extremely important in scenarios of real-time interaction, where device scans must me performed simultaneously with other Bluetooth tasks, like we have described in the section above.

Having multiple Bluetooth interfaces in the same Hotspot raises a management problem: The load over all available interfaces must be equally distributed between all of them. When a Bluetooth task is going to be performed, the Hotspot checks which of the interfaces have the minimum number of connections established and assigns it to that task. Because of the Bluetooth stack limitation of 8 connections, the Hotspot must be able of tracking how many of them are being used on each interface.

The number of interfaces needed when a Hotspot is deployed will always depend on the situation where the Hotspot operate. The number of devices is increased according to the needs, offering scalability, until the limit of 15 devices (refer to OpenProximity in Section 2.3.1).

Chapter 4

Design for Hotspot implementation

In this chapter we present the Bluetooth Hotspot architecture. This chapter also explains how Hotspots integrate with other external system components. Both the architecture and the integration model build on the analysis of Chapter 3.

The chapter is formed by five sections. Section 4.1 introduces the three main entities of the ecosystem that interact with a Bluetooth Hotspot, explaining the role of each of them on the system. Section 4.2 describes the integration of the different components with the Hotspot and describes their main interactions. Section 4.3 details the main aspects of the Hotspot internal architecture. Section 4.4 describes the syntax and semantics of the Hotspot behaviour rules. Finally, section 4.5 details the structure of a Sighting (a report generated by the Hotspot on each device scan).

4.1 System components

The Bluetooth Hotspot ecosystem is composed by three main type of entities: 1) client Bluetooth devices 2) one or more Bluetooth Hotspot(s) and 3) a set of client applications/systems.

Figure 4.1 depicts the essential components of a Bluetooth Hotspot ecosystem. A Hotspot acts as a broker between Bluetooth devices and the remote applications/systems. The applications are willing to receive and provide data from/to remote user devices through Bluetooth interactions, e.g. to obtain Bluetooth device names through a Bluetooth scan or distributing a file through a Bluetooth OBEX Push interaction. The integration with client applications is done over an IP network. The interactions with Bluetooth devices are done over Bluetooth connections. In this section we detail which is the role of each component:

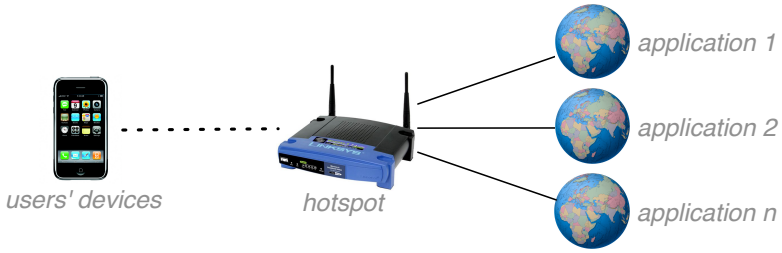

Figure 4.1. An abstraction of the essential components of the system that evolve the Bluetooth Hotspots.

Applications. Applications correspond to the services that can be offered to users when entering a physical space equipped with a Bluetooth Hotspot (see Chapter 2 with multiple real scenarios). These applications assume the presence of a Bluetooth interface able to interact with Bluetooth-enabled devices. In many cases, this interface is attached to the server that supports the application(s). However, in many scenarios, as we have discussed in Chapter 2, it may be advantageous to have the Bluetooth interface physically detached from the application server.

Bluetooth devices. Bluetooth devices are the mean for users to interact with available services in a space. Those include cell phones, PDAs, laptops or any other kind of portable device. The type of support for user interaction is dependent on the available services and on the type of interactions provided by the Hotspot. In order to be detected by the system, user devices should be set with the "discoverable mode" on (refer to Section 2.1 to understand what is this mode).

Bluetooth Hotspots. Bluetooth Hotspots have the ability to establish multiple Bluetooth connections with multiple devices simultaneously. They are designed to perform common Bluetooth interaction tasks as efficiently as possible. Using a Bluetooth Hotspot, applications become free from understanding how to establish Bluetooth connections with Bluetooth devices. Applications are responsible for configuring the Hotspot behaviour, using rules defined by the Hotspot Behaviour Rules set (in Section 4.2 we detail this mechanism). After having configured a Hotspot, applications will wait for, or initiate the interactions correspondent to the defined behaviour, e.g. sending a file to a device, or receiving a periodic list containing the Bluetooth device names from devices in the Hotspot's range.

4.2 Components integration

Bluetooth Hotspots are autonomous components in the way that they can indefinitely perform most of the interactions with users' devices, without depending on other external components. These interactions with devices are however defined in the context of an application. So, interaction data is transmitted from the Bluetooth Hotspot to applications and from applications to the Bluetooth Hotspot, for every defined interaction. When receiving the interaction data, applications will process that data in the context of the application logic. Application logic is also responsible for generating data that is communicated to the Hotspot in order to be delivered to users' devices.

We rely on the Representational State Transfer (REST) paradigm [17] to achieve communication between Hotspots and applications. Compared to Simple Object Access Protocol (SOAP), REST utilizes HTTP as an application protocol rather than as a transport protocol. For this reason, it introduces less overhead. All the communications are realized by POST requests. The messages formats are described in Sections 4.4 and 4.5.

The procedure for integrating the system components is formed by 1) the mechanism by which applications define the Hotspots behaviour, 2) the process for management the sharing of Hotspots between applications, 3) the protocol for managing multiple Hotspots by the same application and 4) the logs report protocol. The whole procedure is described in the following sections.

4.2.1 The Hotspot behaviour

The core process in the process of integrating applications with a Hotspot is the definition of the Hotspot behaviour. The Hotspot behaviour is defined by a set of rules known as the "Hotspot Configuration Rules", or simply the "Rules List". Each rule defines a task to be scheduled and executed by the Hotspot. The type of rule is identified by a unique string (e.g: `deliverfile`, `acceptfile` or `scan`), followed by a list of arguments that customize each task. Typically, these arguments specify characteristics of Bluetooth interactions, like the location of a file to be delivered or filters for the names of devices to interact. In Section 4.4 we detail the whole set of rules that can be installed on a Hotspot.

We rely on YAML Ain't Markup Language (YAML) to describe the Hotspot rules. Compared with other mark-up languages, YAML is very simple to be interpreted and manually written by humans. Furthermore, the performance is not affected [19] when compared with XML, and may even be substantially better for a small number of

elements. This is particularly advantageous considering that the Hotspot software is intended to run on embedded devices with slow processors.

The YAML file, describing the Hotspot rules, is then submitted into the Hotspot. This can be made directly by the application or by a configuration server, considering that applications can access the Hotspot through the Internet protocols. In this case, the application uploads the file with the new rules over an HTTP POST. In the case where the Hotspot does not have direct IP addressing, the Hotspot can periodically download the Hotspot Configuration Rules from an URL on the network[1].

In both cases, every time a new list of rules is submitted into the Hotspot, it will assume the new behaviour.

4.2.2 Hotspot sharing

A Hotspot may serve multiple applications simultaneously. We depict this situation in Figure 4.2. In this situation, we have defined a main point of control, which will be responsible to maintain and submit the Configuration Rules to the Hotspot. In this case the main point of control will be able to resolve any conflicts and provide the adequate degree of concurrency.

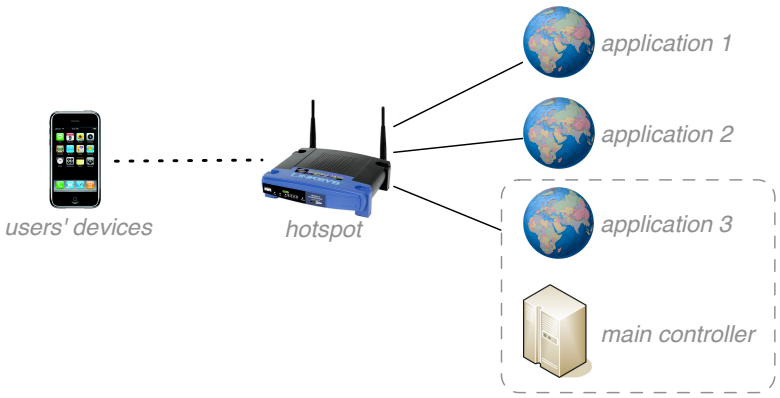

Figure 4.2. Representation of the Hotspot's system architecture where a main application (or a Main Controller) manages the Hotspot.

As seen in the figure, the main point of control, or master, may be 1) one of the sharing applications or 2) a dedicated service that we denominate as "The Main

[1]Typically, this happens when the Hotspot is behind a firewall or installed on network with Network Address Translation (NAT).

Controller". Despite our work is not focusing on the design and implementation of this component, we refer their characteristics and assume it is present in the architecture.

4.2.3 Multiple Hotspots

In many scenarios, a one-to-one association between an application and a Hotspot will be sufficient. This is true when the application's space may be covered by only one Hotspot. For example, an application running on a public display which interacts with nearby users. However, there will be other scenarios where an application (or the Main Controller, if it exists) will have to manage multiple Hotspots, as analyzed in Section 3.2.2. This situation is depicted in Figure 4.3.

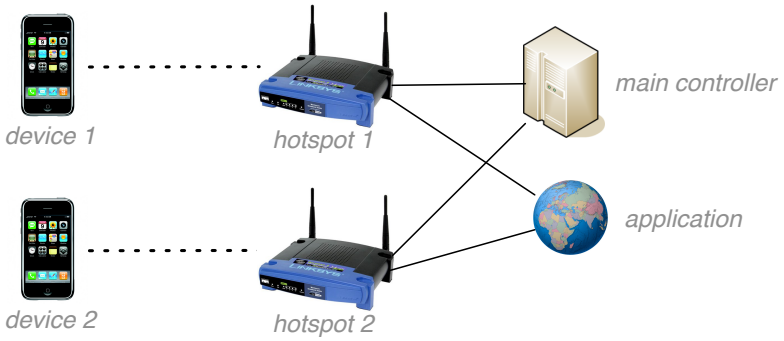

Figure 4.3. Representation of multiple Hotspots reused by the same application. The coordination of context between those two Hotspot is performed by the Main Controller.

4.2.4 Feedback protocol

The feedback protocol was designed to address fault tolerance and improve reliability into the integration process. Using the feedback protocol, applications may request to receive feedback logs about the execution of the desired tasks. The protocol is formed by two parts: 1) The first part consists on the specification of an URL associated with each Hotspot Configuration rule that identifies the application resource to be updated with the respective feedback log. 2) The second part consist on the specification of the feedback message. The feedback message is delivered over an HTTP POST on the specified URL.

Feedback logs may be delivered using two different methods: 1) rule by rule, using a different location for each defined rule or 2) in a unique URL that holds all the tasks performed by the Hotspot.

The first method is useful to deliver logs to different applications that share the same Hotspot. The second method is useful to deliver feedback logs to the Main Controller, if it is present. In terms of configuration, the difference between one method and the other is in the Hotspot Configuration Rules: the first method uses an argument rule by rule, while the second method uses a specific rule for this end. Refer to Section 4.4.6 for a more extensive explanation).

Feedback message

The feedback message is an YAML file. Its format is composed by three values: `status`, `message` and `rule`:

- The first value (`status`) contains a number that identifies the status error. If equals zero (0) it means that the task was successfully performed. If it differs from zero, it means that an error has occurred[2];
- The second value (`message`) contains a short message describing the result of the error (if the previous value is greater than 0);
- The third value (`rule`) contains the content of the rule that was triggered.

Typical errors that can occur are: files not delivered for any reason, incoming files not accepted for being too big or for having an unauthorized extension, URLs not found or empty generated Sightings.

Example: In the example below, the Hotspot was configured with a rule to deliver a file to all devices with their name matching the tag `*iwantfiles*`. The Hotspot tries to deliver the file, but the user leaves the space and the connection times out. Then, a feedback log is generated, with the following content:

```
status: 2
message: connection timed out for device 00:a3:12:3a:f4
rule:
    - deliverfile
        url: http://www.bavaria.com/images/boat.jpg
        devname: *iwantfiles*
```

[2]The status error numbers are not reported on this document because it is not relevant and because it depends on the implementation made. Also, messages generated are also not detailed. These messages are merely informative.

The feedback log indicates that the rule was performed with an error (identified by the number 2). The error message describes that the connection timed out for the device with the address `00:a3:12:3a:f4`.

4.3 Hotspot internal architecture

In this section we describe the Hotspot internal architecture, its main components and the interactions between each component.

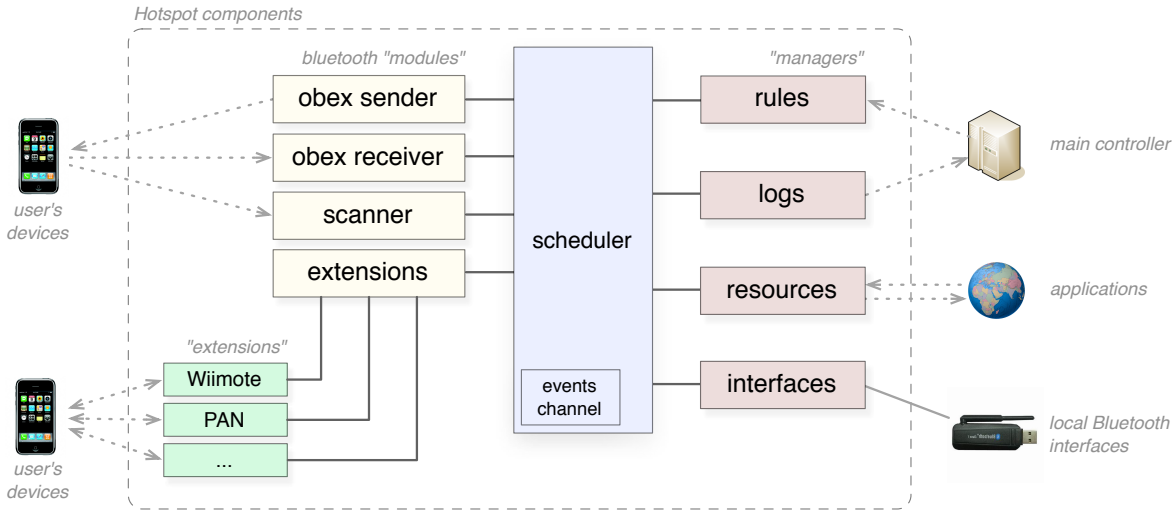

Figure 4.4. An abstraction of the internal components of a Bluetooth Hotpot.

The internal architecture of a Hotspot is organized in four groups of components, illustrated in Figure 4.4: 1) the Scheduler, 2) the Hotspot Managers, 3) the Bluetooth Modules and 4) the Hotspot Extensions. We briefly describe each group below and detail its role in the following sections.

Scheduler. The Scheduler is the component responsible for executing Hotspots tasks. A task can be a Bluetooth interaction or a management task, internal to the Hotspot (typically, an internal task is to enable/disable a specific Bluetooth Module or to registry an entry in the log). To achieve a correct execution of these tasks, the Scheduler holds an event mechanism: an events channel. Both Bluetooth Modules and Bluetooth Managers can register events on this channel. When an event is registered, the Scheduler decides what to perform accordingly to the Configuration Rules that are currently loaded. Then, the task is executed.

Hotspot Managers. These components perform specific internal tasks like 1) the management of the Hotspot Configuration Rules, 2) the management of feedback log reports, 3) the management of Internet resources, such as files and 4) the management of available bandwidth at each installed Bluetooth interface. Hotspot Managers functionality is used by the Scheduler component.

Bluetooth Modules. These components perform Bluetooth-specific interactions: 1) device scannings to obtain the list of present devices, device names and their available services, 2) sending files to devices, and 3) receiving files from devices. Each Module is responsible for one type of interaction, but a special fourth Module is dedicated to the management of Bluetooth Extensions.

Extensions. Extensions are software components that execute application-specific tasks, such as Wiimote controllers interactions or other Bluetooth profiles's interactions. An extension is intended to be developed along with an external application that uses a Hotspot. This specific interaction modules are integrated with external applications using the same model defined for Bluetooth Modules described above.

Extensions are installed on the Hotspot before its operation and thus they can be referenced at the Hotspot Configuration Rules. An extension can work in two modes: 1) always running - started along with the startup of the Hotspot. This allows the extension to be listening to new connections that are started from devices to the Hotspot or 2) be instantiated (started) just when a device or a group of devices is seen, taking advantage of the scan mechanism that the Scanner Bluetooth Module offers.

The next five subsections describe with more detail all these components, as well as the data structures that support them (Sightings structure and Hotspot Configuration Rules structure). The first subsection explains how the Scheduler manages the Hotspot tasks and the behaviour of the Hotspot. The second subsection explains Managers' functionality. The third subsection describes the Bluetooth Modules, focusing the most important issues of their design.

4.3.1 Scheduler

The Scheduler Module triggers the execution of Hotspot tasks, typically Bluetooth interactions. As it deals with asynchronous events, the Scheduler holds an Events

Channel where Modules and Managers can register their events. An event may have one or more associated actions to be triggered. The decision of what actions to be performed will depend on the currently loaded Hotspot Configuration Rules. Table 4.1 establishes the association that exists between registered Events and the corresponding actions to be executed. In this table, the wildcard (*) matches with any event.

Table 4.1. The list of actions that may be triggered by each event.

Events	**Rules / Triggered actions**		
newscan	deliverfile	postsighting	runextension
obexin	acceptfile		
newrules	scan	advertise	
*****	postlogs		

Actions act as small pieces of software included on the Scheduler. They use the available Bluetooth Module (to perform Bluetooth interactions) and Hotspot Manager (to perform internal management).

Using Rules as filters for Actions execution

Rules act as filters for the execution of actions. When an event is registered in the Events Channel, the Scheduler consults the Rules Manager for the list of currently loaded rules. If the list is empty nothing is done. If there are loaded rules, the Scheduler checks the Table 4.1 for actions associated with the event. Each action is executed the same number as the number of times a rule with the same name appears on the loaded' Rules List. This means that, if a rule X appears three times on the list and an event triggers an action X (the same name as the rule), the action X is executed three times, in parallel.

For example, when the `newscan` event is registered in the channel[3], the possible actions to be executed are `deliverfile`, `postsighting` and `runextension`, according to the table. Assuming that the Hotspot is configured with two `deliverfile` rules (with different arguments for different applications or contexts), then the action `deliverfile` will be performed two times in parallel.

[3]The event `newscan` is registered by the Scanner Module after performing a new device scan.

Action execution context

The actions execution contexts are defined by 1) the trigger event data arguments and 2) the associated rule data arguments. Consequently, correspondent tasks may be executed concurrently, in different contexts:

Event's arguments. An event may be defined by a set of arguments, which retain the data that is generated by the occurrence of the event (for example, the event `newscan` registered by the Scanner Module appends the Sighting generated, containing the list of seen devices). These arguments are passed to the corresponding action to be executed.

Rules's arguments. A rule may be constrained by a set of arguments (refer to Section 4.4 for the list of arguments accepted by each rule). Every time an action is executed (along with the occurrence of an event), the rule's arguments are also passed to the action. For example, the `deliverfile` rule has two arguments, which correspond to the name of the file to be delivered and the list of devices, which are passed to the corresponding action.

The action execution context is expressed with the following expression:

$$event(a), rule(b) \rightarrow action(a, b)$$

If a certain event is instantiated as X, a rule as Y and an action as Z (where Y and Z have the same name), this expression can be interpreted as:

```
when event X (with arguments A) occurs,
if the rule Y (with arguments B) is contained in the list of loaded rules,
the action Z is executed, receiving the arguments A and B.
```

4.3.2 Hotspot Managers

The Hotspot Managers (or just Managers) are responsible for the basic functional tasks of the Hotspot. This includes managing the Rules List, obtaining and caching Internet resources (like files), sending resources or managing the concurrency of the Hotspot's Bluetooth interfaces (USB dongles).

Rules Manager. This is the component that manages the Hotspot Configuration Rules (refer to Section 4.2.1 for details about this rules). This manager interprets

a file of rules that is loaded using one of the methods below referred. When loaded, the current rules can be requested by the Scheduler every time it needs them.

There are three possible ways to load a new file with a list of rules on the Hotspot:

- statically, defined on a local and static file;
- through the remote Main Controller at any time[4], in the case for scenarios where the list needs to be constantly updated;
- by periodically downloading it from a remote URL, in the case for scenarios where the Hotspot does not have a public IP address.

If the Hotspot software is running on the OpenWrt operating system, inside Ethernet routers, the configuration of which method is to be used is done using the OpenWrt's configuration mechanism - called Unified Configuration Interface (UCI)[5]. This configuration includes the URL of the rules' file. The URL can be a local path (for the first method) or a remote HTTP location (for the third method[6]). If the Hotspot software is running on another operating system, the same configuration is done on a static text file.

Every time a new rules' file is loaded, the Rules Manager registers an event `newrules` on the Scheduler. The new file of rules is appended to the event as an argument. When this event is registered, the Scheduler starts or stop the Modules in question, accordingly to the rules.

Logs Manager. The logs manager is the component responsible to build the Hotspot's feedback logs (this mechanism of feedback for application is described in Section 4.2.4). Every time an event is registered on the Scheduler a log entry is created in the Logs Manager. When an action is performed by the Scheduler, a log entry is also created.

Resources Manager. This component manages Internet resources, like files. All the other Managers, Modules and the Scheduler's actions use this manager to access external resources from the web. With this manager, files can be downloaded

[4]The Main Controller uploads the rules' files to the Hotspot's HTTP server. This server is provided by the Resources Manager, which is listening for the input of new files.

[5]OpenWrt Linux distribution uses a mechanism called UCI that allows the configuration of single values associated to identification variables. This variables are centralized on the system and can later be consulted by any application running on the system.

[6]In this case, the period of time (in seconds) between each download is included on the second line of the same file.

and uploaded from/to applications. File exchanges are performed over HTTP requests and each resource is identified by an URL. A cache of the downloaded files is maintained, speeding up dependent processes, avoiding unnecessary delays and excessive network usage.

This manager also holds an HTTP server. This server is used by the Rules Manager to listen for the input of new rules.

Interfaces Manager. This component manages the Bluetooth interfaces installed in the Hotspot, usually USB dongles. This implementation follows the requirements reported in the analysis of Section 3.3.2. A Hotspot may have installed one or more Bluetooth interfaces. This way, a Hotspot may execute concurrent Bluetooth interactions.

Bluetooth Modules request a Bluetooth interface from the Interface Manager before executing the Bluetooth interaction. At the moment of system startup, the number of Bluetooth interfaces installed is obtained. The bandwidth of each interface is abstractly assigned to 8 usage slots[7]. Each module or extension that wants to use an interface for a Bluetooth interaction will just have to request an interface, also informing about how many slots it needs. If there are enough free slots, the module (or extension) is locked until the Interfaces Manager releases them. After all the interactions performed, the Bluetooth Module is responsible for freeing the slots. If all the usage slots are in use, the request will be added to a round robin queue until a sufficient number os slots are freed.

4.3.3 Bluetooth Modules

The Hotspot includes a set of modules - called the Bluetooth Modules - that execute the most relevant[8] Bluetooth interactions. A coherent integration means that these Modules, 1) respect the available bandwidth for each Bluetooth interface, using the Interfaces Manager that decides what Bluetooth interface is to be assigned, 2) register the appropriated event on the Scheduler for each Bluetooth task to be performed and 3) free the Bluetooth usage slots after all the Bluetooth interactions are performed.

The Hotspot includes three modules that natively offer three types of Bluetooth interactions: 1) the OBEX Sender Module, 2) the OBEX Receiver Module and 3) the

[7]The reason to use 8 slots comes from a Bluetooth specification that limits a Bluetooth interface to use ate last 8 connections (or 8 devices) at the same time [1].

[8]The decision of what interactions to include was based on the related work survey of Chapter 2 and is justified in the analysis survey of Chapter 3.

Scanner Module. There is a fourth module with an extension mechanism that allows to extend the Hotspot's functionalities.

OBEX Sender Module. This module is dedicated to the delivery of files over Bluetooth to other devices. As depicted in Figure 4.5, when it is used by the Scheduler, a Bluetooth interface is requested to the Interfaces Manager using one usage slot to send the file. The file to be sent is downloaded from its URL, using the Resources Manager. After downloaded, the file is then sent to the Bluetooth device. If it is successfully sent, an event `obexout` is registered on the Scheduler and the usage slot is freed by the OBEX Sender Module.

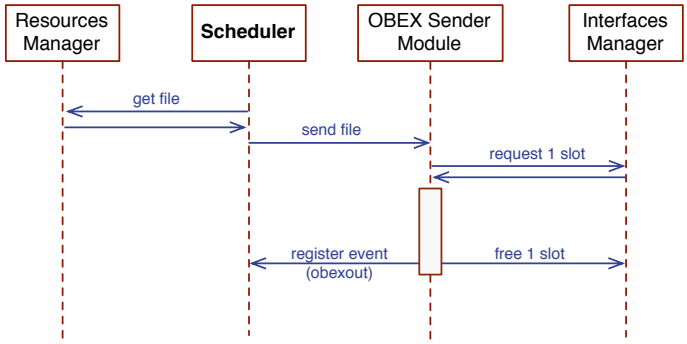

Figure 4.5. Sequence diagram illustrating a OBEX Receiver Module that sends a file to a Bluetooth device.

OBEX Receiver Module. This component receives files from Bluetooth devices. The module has two states: "stopped" and "listening". As depicted in Figure 4.6, when the Module is in the listening mode (because a rule caused the Scheduler to enabled it), the OBEX Receiver module requests a Bluetooth interface to the Interface Manager. Then it advertises the `OBEX Object Push` service (Bluetooth service ID `0x1105`) on the assigned Bluetooth interface. This enables the Hotspot to receive files from Bluetooth devices. When a new file is received, it is stored on a temporary directory and the event `obexin` is registered on the Scheduler, along with an argument with the path where the received file was stored and another argument containing the sender's device address. Thus, any Scheduler's action will be able to access the received file to perform any task, like, for example, sending it to an application.

Scanner Module. When enabled by the Scheduler (i.e. if the Hotspot is configured with a rule for scanning), it constantly scans for the present Bluetooth devices,

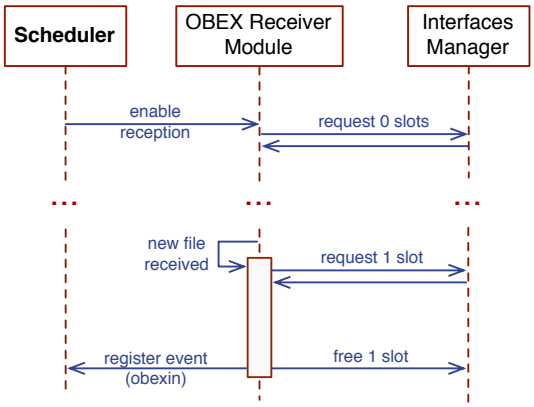

Figure 4.6. Sequence diagram illustrating a OBEX Receiver Module that is enabled and a file that is received by the Module (sent from a Bluetooth device).

as depicted in Figure 4.7. Each iteration results on a list of devices' hardware addresses. This list, associated to the date and time when the scan was performed is called a Sighting. Sightings are useful for any action, module or extension that want to deal with the list of present devices on a certain moment. Sightings can also be passed to the external applications. Every time a Sighting is generated (this data structure is detailed in Section 4.5) it can be locally stored on a file to be later retrieved. It may also be uploaded to the URL of an application via HTTP, if the Hotspot is configured with the rule with this objective (refer to the rule `postsightings` in Section 4.4).

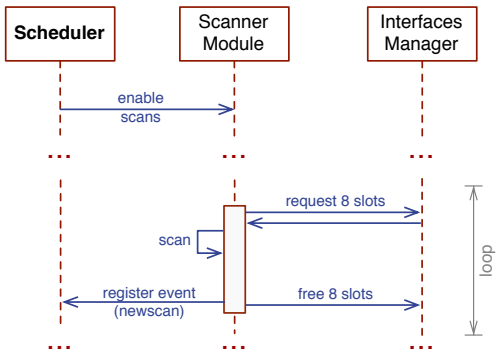

Figure 4.7. Sequence diagram illustrating a Scanner Module that is enabled and then perform multiple scan devices, in loop.

Optionally, the Scanner Module can be configured by the Scheduler to retrieve more information about each device. We identify this group of information as

the Extra Device Information, and it includes at least the following data: 1) the device class, representing the type of device 2) the human readable name and 3) the Bluetooth services available in the device. If configured to, after each scan, the Scanner will retrieve such information from each device, storing it on a local cache. The next time a new Sighting is generated this information about each device will be also attached to the Sighting.

The device scan process

During a scan, the Bluetooth interface that executes a scan is monopolized, entering on a state of high bandwidth consumption. This means that, during the scan no other Bluetooth tasks should be performed, or its performance will be drastically affected. For this reason, before a scan, the Scanner module requests 8 usage slots to the Interfaces Manager, freeing them at the end of the scan. Then, an event `newscan` is registered on the Scheduler, followed by an argument that contains the Sightings.

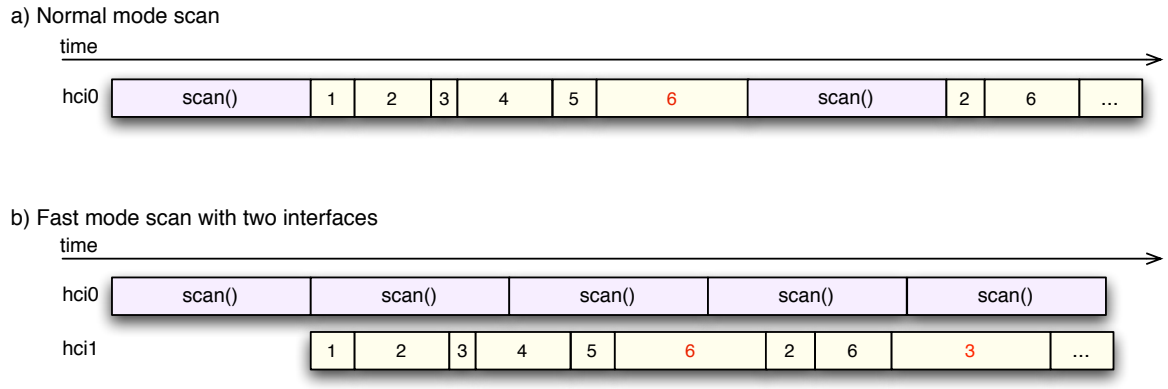

Figure 4.8. An example that compares the performance of a scanner with the fast scan mode disabled and another with the fast mode enabled.

Normal vs. fast mode scan

Some scenarios (detailed in the related work of Chapter 2 and in the analysis survey of Chapter 3) require a very fast scan for real time situations where a fast feedback is needed by the users. The Scanner Module supports a mode of scan that speeds up the process by scanning with one Bluetooth interface and

obtaining the Bluetooth device's names and services with another interface. This mode is called the "fast mode". It monopolizes one of the interfaces just for scanning, leaving the other(s) for all the other tasks. This way, the performance of the scan will never be affected. Figure 4.8 shows a comparison between these two modes, where the Extra Device Information is requested after or during the scans, respectively. In the example of the figure, the speed of this request is different for each device (identified by numbers) and one of the requests (6, marked as red) times out.

Extensions module. An extension is a software component that performs application-specific Bluetooth interactions. The Extensions Module integrates the available extensions with the remaining components of the system. To integrate with the Hotspot, extensions must be pre-installed on the "extensions" directory of the Hotspot and implement three methods: `start()`, `newScan()` and `stop()`. The Extensions Module is responsible for maintaining the instance of each running extension. All those three methods are called by the Extensions Module on the respective instance:

- The first method - `start()` - is called when the extension is started. An extension can be started in two situations: 1) when the Hotspot system starts up or 2) when a new Bluetooth device is detected by the Scanner Module. This method receives an argument with the Bluetooth interface assigned to the extension and an argument with the URL of the extension's configuration file. The content of this file must be handled by the extension. Section 3.2.3 details how extensions are handled by the Extensions Module.
- The second method - `newScan()` - is called every time a new scan is performed by the Scanner Module. This method receives an argument with the Sighting associated with the scan.
- The third method - `stop()` - is called when the extension is stopped by the Extensions Module. This situation occurs when a new list of rules is loaded on the Hotspot and the extension is not referred in the list.

Only an instance per extension is launched (a thread per instance). While running, an extension monopolizes a Bluetooth interface just for itself, avoiding collisions with other Bluetooth tasks inside the Hotspot. At the end, the extension will be responsible to stop itself when appropriate, using the method

`stop()` - and is strongly encouraged to do it as fast as possible in order to free the Bluetooth interface.

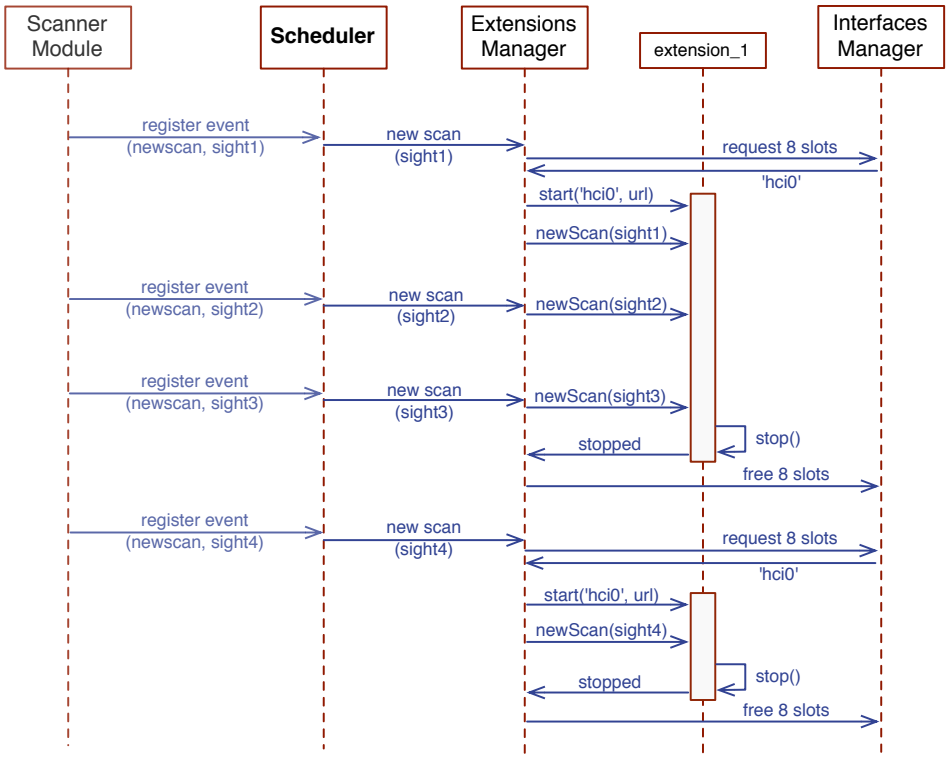

Figure 4.9. Sequence diagram illustrating an extension that is triggered by Scanner Module's scans.

Instantiation of extensions

When a new scan event is registered by the Scanner Module on the Scheduler, the Extensions Manager will be always notified. When this occurs, the available extensions are also notified: For each available extension, if it is already running, the extension is notified through the `newScan()` method. If is not running yet, the extension will first be started with the method `start()`, receiving two arguments: 1) the ID of the Bluetooth interface assigned to the extension and 2) an URL containing a file with eventual configurations of the extensions. The Extensions Manager will download this file and provide it to the extension. The semantic of this configurations will depend on the extension context, so only the extension will be responsible for interpreting this file.

Figure 4.9 shows an example where an extension `extension_1` is started when a

new scan occurs. The extension is running until it decides to stop (depending on its context). In this example, as it stops after the third scan, when a fourth scan occurs, the extension will be started again before receiving the Sighting resulted from the scan.

4.4 Rules's structure

The Hotspot Configuration Rules, introduced in Section 4.2.1, describe how a Hotspot should behave. The Hotspot's Rules List is a flat list which may contain any number of any type of rules. In this section we explain with detail all types of rules, along with an explanation about the arguments they support. Each rule is followed by an example of configuration.

4.4.1 Enabling device scans

This rule (identified by `scan`) enables the device Scanner Module. Almost all the other rules need the presence of this rule. If defined without any argument, only the hardware addresses of the present devices are periodically discovered. This rule accepts the following arguments. Time values are defined in seconds and boolean values are set to false by default. For the numeric arguments, the default values derive from the analysis of scenarios of the related work survey of Chapter 2:

- `scaninterval` - defines the time to wait between each scan[9] (default=30).
- `fastmode` - is a boolean value that enables or disables the fast mode scan.
- `getnames` - is a boolean value defining if the Scanner will get the name of each device after each scan.
- `getservices` - is boolean value defining if the Scanner will get the available services of each device on each scan.
- `namechache` - defines the amount of time that a name is kept in cache when obtained in a scan (default=60).
- `servicescache` - defines the amount of time in seconds that the list of services is kept in cache when obtained in a scan (default=3600).
- `scantimeout` - defines the duration in seconds of each scan until it times out (default=8).

[9]The scan time can vary depending on the number of present devices, specially if the fast mode isn't enabled and the names and services inquiries are enabled.

- `nametimeout` - defines the duration in seconds of the process of retrieving the name of a device until the process times out (default=8).
- `servicetimeout` - defines the duration in seconds of the processing of retrieving the available services of a device until the process times out (default=8).
- `scanmaxtime` - is the maximum time in seconds of each **scan** + **get names** + **get services**. This value forces the Scanner to stop after this time, even if one of the tasks is not completed. (default=30).

Fast mode device scans

The argument `fastmode` assigns the Scanner to one Bluetooth interface leaving the inquiries of names and services to a second interface, if available. Unless the argument `fastmode` is set to "true", the Scanner searches for the present devices, then inquiries each of them for its name. A device name is considered old after 60 seconds and the available services after 3600, but this values can be override by the arguments `namecache` and `servicescache`.

Maximum time of a full scan

Probably the most important argument for applications running on real-time scenarios is `scanmaxtime`, which specifies the maximum time between each scan, including the time for getting the names and the available services of each device. Note that this time will only affect the name and service getting phase and not the device scan phase itself. Section 3.3.1 explains why this argument is so important for the Hotspot performance.

Example: In the following example, assuming that two Bluetooth interfaces are installed on the Hotspot, the Scanner will search for present devices exactly each 30 seconds (on the first interface) and get the names and services of each device (on the second interface). If a specific device is constantly seen, it will be inquired for its name every two scans (30+30 seconds):

```
- scan:
    scaninterval: 30
    fastmode: true
    getnames: true
    getservices: true
    namechache: 60
    servicescache: 3600
```

```
    scantimeout: 8
    nametimeout: 8
    servicetimeout: 8
    scanmaxtime: 30
```

4.4.2 Sending Sightings to an application

This rule (identified by `postsightings`) tells the Hotspot to send the results of each device scan to a remote HTTP server (to be consumed by an application). When a rule of this type is present, the list of devices and the information collected by the Scanner - the Sighting - is sent on a HTTP POST request to an URL. If a scan results on an empty list of devices, the Sighting will not be sent to this URL, unless the argument `sendifempty` is set to "true". Thus, the rule accepts the following arguments:

- `url` - defines the location where to upload the Sightings (mandatory).
- `sendifempty` - is a boolean value defining if the Sighting is to be posted even if empty.

Example: With the rule of the following example, the Scanner module will send all the Sighting files to the HTTP server located at `http://app1.com/postsightings.php`, even if the Sighting includes an empty list of devices:

```
- postsightings:
    url: http://app1.com/postsightings.php
    sendifempty: true
```

4.4.3 Sending files to Bluetooth devices

This rule (identified by `deliverfile`) tells the Hotspot to deliver a file to the present Bluetooth devices that have the service `OBEX Object Push` available (Bluetooth service ID `0x1105`). When this rule is present, the Scanner will automatically inquiry the devices about their available services to understand if that service is available or not. When the Scanner detects the presence of devices, the file located at the URL provided is downloaded to the Hotspot and sent to all the devices seen by the Scanner. Each device that accepts the file will not receive the file anymore until the next day, except if the argument `rememberfor` is defined with the number of seconds that should pass until the that device should be forgot. However, if the file could not be delivered for

any reason (if a device does not accept the file or the connection fails) the Hotspot will try again for 3 times, or the number of times defined at the argument `retrytimes`.

The arguments accepted by the rule are:

- `url` - the location of the file to be delivered (mandatory).
- `devaddress` - a string to filter the recipient devices by their device address.
- `devname` - a string to filter the recipient devices by the human readable name.
- `devclass` - a string or hexadecimal value to filter the recipient devices by their device class (e.g: "Computer" or `0x100`).
- `rememberfor` - the amount of time in second until next delivery, after success (default=86400).
- `retrytimes` - the number of retries until timeout, for each device (default=3).

Device filtering

If the file is not intended to be delivered to all the devices, it is possible to define filters by device address, by device name or by device class (type of device). These filters can include exact strings to match the devices or can be defined with a wildcard (*). Thus, the file will only be delivered if the expression(s) defined on the rule match the device(s) addresses, names or services.

Example: In the following example, the file "boat.jpg" will be delivered to all the Nokia cell phones (device address starting with `00:17:E4`) that have the tag `iwantfiles` on their name. If the file could not be delivered, the Hotspot will try again for more 2 times. Then, if the delivery to a device is succeeded, the file will not be sent to that device again, during the next hour:

```
- deliverfile
    url: http://www.bavaria.com/images/boat.jpg
    devaddress: 00:17:E4:**:**:**
    devname: *iwantfiles*
    devclass: CellPhone
    rememberfor: 3600
    retrytimes: 3
```

4.4.4 Receiving files from Bluetooth devices

This rule (identified by `acceptfile`) tells the Hotspot to accept files from user devices. If one or more rules of this type are defined, the service `OBEX Object Push` (Bluetooth

service ID `0x1105`) will be automatically advertised on the Hotspot and its visibility will turn on, with the name "Bluetooth Hotspot" (this name can be overridden with the argument `advertisename`). From this moment on, the Hotspot accepts all the files received and sends them over an HTTP POST request to the defined URL. Device filtering for device address, device name or device class is identical to the previous rule, using the same three arguemnts. This rule adds an extra filter for the file name of the received file. Using a wildcard (*) it becomes possible to filter files by extension (e.g: `filename=*.jpg`), rejecting all the other possible extensions.

Thus, the arguments accepted by the rule are:

- `url` - the URL where to upload the file received from the user devices (mandatory).
- `advertisename` - the Bluetooth name of the Hotspot, to be advertised to user.
- `filename` - a string to filter the name of the file received from the device.
- `devaddress` - a string to filter the recipient devices by their device address.
- `devname` - a string to filter the recipient devices by the human readable name.
- `devclass` - a string or hexadecimal value to filter the recipient devices by their device class (e.g: "Computer" or `0x100`).

Filling the URL with relevant data

When a file is received by the Hotspot and is sent to the defined URL, the application that consumes the file will probably want to know 1) the name of the file and 2) whose device has sent it. In order to pass this information to the application, the tags `$filename` and `$devaddress` are available to be included in the appropriate place of the provided URL. For example, if the provided URL in the rule is

```
http://app1.com/post.php?addr=$devaddress&file=$filename
```

and a device with the address `00:11:22:33:44:55` sends the file `star.jpg` to the Hotspot, then the URL to where the file will be sent will be translated to

```
http://app1.com/post.php?addr=00:11:22:33:44:55&file=star.jpg
```

Advertisement in multiple interfaces

If multiple rules of this type are present, a file received by the Hotspot will be sent to all the URLs defined in all those rules. However, if different rules define different advertise names, two different interfaces (if available) will be used to receive files, advertising

the `OBEX Object Push` service on both interfaces. Suppose the next Rules List, as an example:

```
- acceptfile:
    url: http://app1.com/files.php
    advertisename: BlueBox
- acceptfile:
    url: http://app2.com/postfiles.asp
    advertisename: BlueBox
- acceptfile:
    url: http://app2.com/postfiles.asp
    advertisename: GreenBox
```

If the Hotspot is configured with the three rules of the above example, it will advertise the `OBEX Object Push` service in two Bluetooth interfaces with different names: 1) the name "BlueBox" is used in one Bluetooth interface and 2) the name "GreenBox" on the second interface. Thus, all the files sent from devices to the Hotspot using the first interface will be sent to both the URLs `http://app1.com/files.php` and `http://app2.com/postfiles.asp`. The files sent to the Hotspot using the second interface will only be sent to the second URL.

4.4.5 Starting an extension

This rule (identified by `runextension`) tells the Hotspot to start an extension, identified by a string with its name. If the argument `startup` is defined to "true", the extension will start when the Hotspot starts up (or anytime a new list of rules is installed). Otherwise, if this argument is not defined, the extension will only start when at least one device matches at least one of the filters (device address, device name or device class).

When the extension is requested to start, if is not running yet, the instance will be created. Then, when a Bluetooth interface is assigned, the extension will start running as a thread. The next time the rule is used (triggered by a device that was seen) the same instance will just be informed about the new device scan - using an appropriate method (refer to Section 4.3.3). If more than one application wants to use the same extension for different contexts, multiple instances will need to be created: one instance per application. To achieve this, each application specifies its own unique ID for their instance, using the `instanceid` argument. This ID can be a string or a number.

Along with the name of the rule, it is also required an URL where the configuration for the extension may be found. This file contains all the configurations the application want to pass to the extension.

This way, the arguments accepted by this rule are:

- `extname` - the name of the extension to be run (mandatory).
- `configurl` - the URL where the configuration of the extension is located (mandatory).
- `startup` - is a boolean value defining if the extension will start when the Hotspot starts up (or a new list of rules is loaded).
- `instanceid` - the identification of the application that uses this extension.
- `devaddress` - a string to filter the recipient devices by their device address.
- `devname` - a string to filter the recipient devices by the human readable name.
- `devclass` - a string or hexadecimal value to filter the recipient devices by their device class (e.g: "Computer" or `0x100`).

Example: With the rule of the example below, the Hotspot will start the extension "wiimote" when at least one device with the name starting with "Nintendo" is found:

```
- runextension:
    extname: wiimote
    configurl: http://app1.com/wiimoteconfigs.txt
    devname: Nintendo*
```

4.4.6 Feedback logs

The Hotspot holds a mechanism of feedback to the applications. As described in Section 4.2.4, feedback logs can 1) be delivered in a specific location for each defined rule or 2) be delivered in a unique location for the entire Hotspot. The first method is performed passing an extra argument (`logurl`) to the rule in question. All the rules accept this argument. It includes the URL where to deliver the feedback logs of that specific rule. The second method delivers all the feedback logs of the Hotspot in a unique URL, without the need to specify it on every single rule. With this objective, the Hotspot accepts a rule - identified by `postlogs` - that receives one argument:

- `url` - contains the location where to send the feedback logs (mandatory).

Example: In the example below, the Hotspot is configured with two rules. These rules tell the Hotspot to perform two types of tasks: to accept files from devices and to send the generated Sightings to an application:

```
- postlogs:
    url: http://192.168.1.50/logs/create
- acceptfile:
    url: http://app1.com/files.php
    logurl: http://app1.com/postlogs.php
- postsightings:
    url: http://app2.com/postsightings.php
    logurl: http://app2.com/postsightings.php
```

When one of these tasks is performed a feedback log is generated. After the execution of the first rule, their logs are sent to the address `http://app1.com/postlogs.php`. On the other hand, the second rule sends the logs to a different address: `http://app2.com/postsightings.php`. Independently of the rule that is performed, all the referred feedback logs are also sent to the URL of the Main Controller, located in `http://192.168.1.50/logs/create`.

4.5 Sighting's structure

The result of each scan performed by the Scanner Module is a list of devices - called a Sighting. It is usually intended to be uploaded to a remote application (refer to Section 4.2). Along with the list of devices, a Sighting also contains the identification of the Hotspot and the date and time when the scan was performed.

- `scannerName` - contains the name of the Hotspot that performed the scan
- `scanBeginTime` - defines the date and time the scan started at
- `scanEndTime` - defines the date and time the scan finished at
- `seqNumber` - defines the sequence number of the current sighting, incremented on each scan
- `seenDevices` - contains the list of seen devices. Each item of this list contains a device which must mandatorily include its 1) single hardware address. Besides, if it is known, it also contains: 2) the device's name, 3) its class hexadecimal value and 4) the list of its available services and associated channels. If the second and fourth values are present, also an extra field for each of them will also be

included, containing the last date and time each one was retrieved. When any of these fields is unknown, they are automatically not included.

Example: As an example, we will assume a Hotspot called "UbicompBluebox" that is periodically scanning for devices. The 17th scan started at 1h 32m 4s of April the 1st 2010, and took 48 seconds to retrieve all the data. Three devices were found, but only the name of the devices `22:22:22:22:22:22` and `33:33:33:33:33:33` were successfully retrieved. Moreover, only the services[10] of the last one could be collected. For this situation, the resulting Sighting would be the next one:

```
scannerName: UbicompBluebox
scanBeginTime: 2010-04-01 01:32:04
scanEndTime: 2010-04-01 01:32:52
seqNumber: 17
seenDevices:
    - device:
        address: 11:11:11:11:11:11
    - device:
        address: 22:22:22:22:22:22
        name: Mary Sea
        class: 0x102
        lastNameGet: 2010-04-01 01:31:15
    - device:
        address: 33:33:33:33:33:33
        name: Nokia Michael
        class: 0x102
        services:
            2: Dial-Up Networking
            9: OBEX Object Push
            10: SyncMLClient
            11: OBEX File Transfer
            12: Nokia OBEX PC Suite Services
            13: SyncML DM Client
            14: Nokia SyncML Server
            15: Imaging
            23: AVRCP Target
            28: Hands-Free Audio Gateway
```

[10]The channels and names of the services of the hypothetical device `33:33:33:33:33:33` were retrieved from a real device, a Nokia E65.

```
    29: Headset Audio Gateway
lastNameGet: 2010-04-01 01:31:53
lastServicesGet: 2010-04-01 01:18:34
```

Chapter 5

Evaluation

In the previous chapter we have proposed an architecture for a Bluetooth Hotspot that supports Bluetooth-based user-centered interactions. It is extremely important to evaluate how this architecture behaves on real scenarios during the design process because it allows to redesign some components based on the analysis of functional patterns.

The process of evaluation and validation is reported in this chapter, organized in two sections: first section (5.2) enumerates the goals for evaluation; second section (5.3) describes how prototypes were deployed, along with the results of each deployment.

5.1 Evaluation environment

To evaluate both the Hotspot's architecture and the integration model, we have developed a few prototypes with the objective of being deployed in real scenarios. These prototypes were designed to run on Linux-enabled Ethernet routers, like the one shown in Figure 5.1. In these routers we installed OpenWrt[1] Linux operating system - a Linux distribution optimized to run on embedded devices with low resources of memory, CPU and disk space. This operating system allows to run all the essential features of our prototypes, like e.g. running multiple multi-threaded processes, accessing a TCP/IP network and access the Bluetooth stack (BlueZ[2]).

We have chosen to work with this kind of devices due to many factors, referred below. Values shown are estimated values, obtained from the units we have used as prototypes and also obtained from a sample of the OpenWrt's table of supported

[1] OpenWrt Linux distribution - `http://www.openwrt.org/`

[2] BlueZ is a set of libs and system tools that implements the Bluetooth stack for Linux - `http://www.bluez.org/`

hardware[3]):

- Size (around 20x10cm) eases the installation of a Hotspot in hidden places for environments like bars, halls or streets.
- With a low power consumption (around 5 to 10W), it becomes possible to deploy a Hotspot (or a group of them) without high energy consumption issues.
- CPU clock (around 250MHz) and the available memory (around 16MB) proved to be more than enough to run the entire OpenWrt system and our prototype Hotspot's software without loss of performance, comparatively to the performance of the development prototype environment on a computer.
- Equipped with two USB ports it is possible to plug Bluetooth USB interfaces (an USB hub can be attached to install a large number of Bluetooth interfaces).
- The estimate price of these kind of routers floats between 70€ and 100€, depending on factors like available memory or Central Processing Unit (CPU) clock speed.

The Hotspot prototype software developed to run in these routers was developed in Python[4] with PyBlueZ[5].

5.2 Evaluation objectives

We have created and deployed multiple instances of the Hotspot prototype. Prototypes run on multiple Linux Ethernet routers, and were used by two projects currently under development at our department. This study mainly aimed at evaluating and validating three essential key issues of the system:

Integration model. We want to evaluate the integration model between Bluetooth Hotspots and applications. In particular, we want to evaluate the Hotspot Configuration Rules and the configuration process, the device Sightings structures, and the interaction model between the Hotspot and applications.

[3]OpenWrt's table of hardware - `http://wiki.openwrt.org/toh/start`

[4]Python is an high-level scripting programming language - `http://www.python.org/`

[5]PyBlueZ is a wrapper that gives access to the BlueZ Bluetooth stack for Linux - `http://code.google.com/p/pybluez/`

Figure 5.1. An Ethernet router (equipped with an USB Bluetooth dongle) running a prototype of the Bluetooth Hotspot software on OpenWrt Linux.

Stability and performance. One of our goals is to design a system capable of running on an embedded device (a Linux-enabled Ethernet router). The developed prototypes also have the objective of evaluating the viability of using OpenWrt Linux running Python and PyBlueZ libraries to support all the Bluetooth operations (reported in Section 4.3.3).

Viability of a reusable and shared component. We want to understand which operations a Bluetooth interface can perform simultaneously without loss of performance. Different operations take different times to perform. For example: 1) scan, 2) scan + get device name, 3) scan + get device services, 4) scan + get device names + get device services + delivering/receiving files or 5) other combinations of Bluetooth operations and interactions. With these deployments, we wanted to measure performance times in real scenarios. In some contexts, some operations may take too much time if they run sequentially that it strongly discourages the usage of a single interface. In that situations, the usage of multiple interfaces should be required.

5.3 Real scenarios deployments

The validation of the prototypes was done in two different phases:

- in a first phase, we have tested the Hotspot's Modules independently, without being integrated with the whole Hotspot system and
- in a second, we have tested the Hotspot's Modules, all together on the same Hotspot, integrated with all the other components.

In both phases the validation was performed using prototypes of the designed system on two projects running at our department[6] - InstantPlaces [27] and AnyWherePlaces[7] - both oriented to the area of Pervasive Computing and Smart Spaces. Both phases were performed with our prototypes running on ASUS WL500gP v2[8] routers equipped with the Linux distribution OpenWrt 8.09.2 and a minimal version of Python 2.5. The choice of older versions of the operating system and Python interpreter were due to stability issues: During the development phase of the prototype, recent versions of OpenWrt (10.03) and Python (2.6) were also tested, but the system became really unstable, with unwanted "segmentation faults" and spontaneous system reboots.

5.3.1 First phase

In the first validation phase, only the Scanner Module was deployed. This module was collecting device Sightings and sending them to the HTTP servers of InstantPlaces and AnyWherePlaces, simultaneously. The prototypes developed to evaluate the Hotspot on this phase are characterized for having: 1) no support for multithreading; 2) hard-coded Hotspot Configuration Rules and 3) only one Bluetooth interface used for all the Bluetooth tasks: to perform device scans, to get the names and to get the services of the devices. The objectives of this phase were:

- the validation of the proposed Sighting's structure,
- the evaluation of the scan duration times and
- the evaluation of the robustness of OpenWrt running Python to perform the desired Bluetooth tasks.

[6]Departamento de Sistemas de Informação - `http://www.dsi.uminho.pt/`

[7]AnyWherePlaces - A project in the scope of the a research program on Systems Software for Ambience Intelligence - `http://ubicomp.algoritmi.uminho.pt/system-software/`.

[8]ASUS - `http://www.asus.com/`

Despite the system had been designed to use YAML for data representation (refer to Section 4.5), in this prototype we used a XML representation of Sightings with a structure suggested by the project InstantPlaces, in order to ease the development and avoid the implementation of proxies to convert the data.

Silent scan and proximity sensing

The InstantPlaces project has deployed two Hotspots at both entrances of our department and one at a high school (Escola Secundária das Taipas). All of them were constantly and repeatedly collecting the addresses and names of Bluetooth devices.

During three months, InstantPlaces applications were just silently collecting the Sightings from our Hotspots (by "silent" we mean collecting data without being noticed by who is passing by) and not showing any information to the users. On a fourth month, a public display was installed at the school, displaying the name of Bluetooth devices, linking this information with a Facebook[9] application that shows the photo of device's users. Additionally, another Hotspot was deployed at Vila Verde's Library (in Braga, Portugal), silently collecting device Sightings to InstantPlaces.

Results: This phase allowed us to conclude that for silent scanning scenarios the Scanner period for each full-cycle scan[10] may be increased without too much impact: if any device was not seen it would be like it was not present. However, if the user was expecting some feedback based on his presence, the period of each scan could not be set in a magnitude of minutes, but it should be set in a magnitude of seconds (around 30 seconds). A greater value would result in a poor usability for the user.

We could also conclude that the inquiry phase of the name of each device may take too much time when a large number of devices is present. This situation was predicted during the design phase section - issue referred in Section 3.3.1, where a user waits for a feedback twice the expected time. This issue proves that the support for multiple Bluetooth interfaces is a strong requirement in order to achieve concurrent tasks (e.g scanning with a device and getting device names with another). This leaded us to the implementation of the "fast mode scanning", described in Section 4.3.3.

The collected information about present devices (hardware address, name, class and services) was useful, not only for our research, but also for the InstantPlaces investigation because is allows to validate 1) the Sighting's structure and 2) the differences

[9]Facebook - `http://www.facebook.com/`

[10]An entire full-cycle scan includes: 1) searching for the present devices, 2) retrieving its names and 3) retrieving the available services on each one.

between silent scanning and situations where users known that their devices were being searched (or where they could suggest content to show on the public displays, using a predefined tag on their device's names).

For the studied cases of deployments, the time between the moment a device enters in the range of a Hotspot and the moment a Sighting (containing all the information collected from a full-cycle scan) is generated to be send to remote applications, was between 30 and 90 seconds, with most of the cases taking less than 60 seconds.

5.3.2 Second phase

The second phase of validation was essentially intended to validate all the system components, multithreaded and coordinated between them. As PyBlueZ libraries does not support multiple Bluetooth interfaces, we have developed a patched version of this library to support this feature, in order to use the Interfaces Module as it was designed (refer to Section 4.3.2).

Many Hotspot's prototypes were deployed in different contexts to evaluate its performance and integration in scenarios of 1) proximity sensing and 2) content delivery, as detailed below.

Device Name interaction for proximity sensing

This stage of evaluation was an extension of the first one, to systematize its results. We installed the same two Hotspots at 1) our department entrance and another four in four different high schools: 2) Escola Secundária de Camilo Castelo Branco, Vila Real 3) Escola Secundária de Mirandela 4) Escola Secundária de Valpaços and 5) Escola Profissional de Felgueiras.

During three weeks, each of these Hotspots was collecting device Sightings to the AnyWherePlaces application (running at our department). With this data, the application was presenting the name of present Bluetooth devices in a public display installed near to each Hotspot. Running this new version of the prototype all the Hotspots were installed with Internet connectivity to be able to reach the AnyWherePlaces application. The main objective of this deployment was the validation of:

- the performance of "normal mode scanning" - where device scanning and name retrieval are performed consecutively (refer to Section 4.3.3),
- a mechanism to define a time out for device scans in situations for a large number of devices and

- system stability.

Each of the six deployed Hotspots was installed with just one Bluetooth interface, running in "normal mode scanning", with scan periods of 30 seconds. Two Hotspots were configured with the maximum time for each full-cycle scan time (which includes scan + names of devices + services available) defined to 30 seconds[11] and the three others were configured without this limit.

Results: In the first phase of validation, when the number of device was above 5, the time of a full-cycle scan exceeded more than a minute in most of the cases. This could be avoided with the definition of a maximum time for a full-cycle scan. However, this could lead to a situation where some device names and services were not obtained.

For scenarios of silent scanning this problem is not so relevant if the time for keeping names and services in cache is high (i.e. if a name or service could not be retrieved in a scan it should be retrieved in next scans, and then kept in cache). However, highly interactive scenarios where the devices names need to be constantly refreshed, this cache time cannot be so high.

During all this deployment phase, when we forced a value for a maximum scan time, the mean number of scans needed to successfully get the name of a device was two (2). Without this maximum time defined the mean time was one (1) scan. This situation turned mandatory the implementation of the "fast mode" scan (described in Section 4.3.3 and whose evaluation deployments are described in next section).

Some stability issues were detected on BlueZ libraries while running on OpenWrt, which do not occur on the development environment: when a large number of devices are present, the BlueZ Bluetooth stack randomly lose connection with the Bluetooth interface, forcing to perform a soft reset to the system. We are expecting to use future validation phases to better study and to solve this issues, as well as waiting for future updates to the implementation of these drivers and libraries for OpenWrt.

Content delivery

During two weeks, two Hotspots were deployed at our department in two different entrances. For a week, the AnywherePlaces project installed public displays for the School of Engineering's week from our university. The objective was 1) to display the names of present Bluetooth devices and 2) to deliver a file to each of them. With

[11] For these two Hotspots the configuration rules installed where exactly as the example shown in the example of the rule that "enable device scans" (`scan`) in Section 4.4, except the argument `getnames` which was set to `false`.

around a thousand students visiting our department, this became a good opportunity to validate almost all the designed components of the Hotspot. We Installed two routers running this new version of the prototype, one week for testing and one week for validation on the real scenario, aiming at:

- to validate the management of multiple Bluetooth interfaces by the Interfaces manager,
- to compare the results of the Scanner in "fast mode" scan with the "normal mode" scan (used on previous phases),
- to validate the OBEX file delivery,
- to validate the Configuration Rules structure with multiple types of rules defined and
- to validate the patched PyBluez libraries and also the LightBlue[12] libraries that ease the development of OBEX file transfers over Bluetooth.

Both the Hotspots were configured to 1) scan for the present devices and to 2) send the generated Sightings to the AnywherePlaces application, in order to show the names of present Bluetooth devices in the public displays and to 3) send a welcome text file to each device (this file was available at a URL on the AnyWherePlaces server).

The most important difference between these two Hotspots was the mode of scan: while the first Hotspot had just one Bluetooth interface installed, the second one had two. With the Scanner module working in "fast mode", one interface was dedicated just for device scans, leaving the second one for getting names, getting services and sending files.

Results: Only 10% of the attempts to deliver a file to the devices were successful. All the other 90% fail due to rejection of the file by the user or due to time out. This means that a problem of usability may exist, which is not the focus of our work but may be an important result for the applications. We aimed to analyze how much time a user had to wait for a file after entering on a space: from the point of view of the Hotspot, this corresponds to the amount of time between the first time a device is seen and the moment when a notice of a new file appears on the user's device screen.

[12]LightBlue is a Python library that eases the access to Bluetooth operations of BlueZ Bluetooth stack, in particular those related with OBEX file transfers and advertisement of Bluetooth services - `http://lightblue.sourceforge.net/`.

The average time of delivery was about 50 seconds, even with some moments when 15 devices were present, which is completely acceptable for all the studied scenarios of content delivery (refer to Section 2).

Also, some users reported that their phones asked for a code to pair the phone. After analyzing those devices, we concluded that this process of pre-pairing is mandatory for some models, meaning that automated file delivery as desired was not possible for those kind of devices. This lifted another usability problem, which may be studied in a future phase.

There was a big difference of performance between a Hotspot installed with one or two Bluetooth interfaces:

Single Bluetooth interface. Even defined with a period of 30 seconds between each scan, the router with only one interface installed - and thus using "normal mode" scans - performed many full-cycle scans (scan + get names + get services) that took more than 60 seconds. Furthermore, when a max full-cycle scan time of 30 seconds was forced, half of the names could not be retrieved, which was unacceptable for the current scenarios.

Multiple Bluetooth interfaces. The other router, with two Bluetooth interfaces installed, configured with the "fast mode" scan enabled - which means that one of the interfaces becomes dedicated to devices scans - could obtain the names and services of almost every device, even when delivering files in parallel. Moreover, also the scan duration time was respected most of the times, with an effective scan mean duration of 15 seconds when a small number of devices is present and a mean duration of 30 seconds when a large number of devices is present.

This means that scenarios where a large number of devices are present (both content delivery and scan scenarios) the use of two Bluetooth interfaces becomes mandatory, if the performance of the Hotspot's interactions is crucial.

5.3.3 Third phase

At the moment of writing of this dissertation some key validation issues were still being evaluated. This means that we could not acquire well-grounded results yet but we still have chosen to describe the objectives of this phase, even without results and even under development. Thus, with this validation phase, we want:

- to get better results of comparison between using the Hotspot with the Scanner in "fast mode" scan and "normal mode" scan,

- to evaluate how the Hotspot behaves when using OBEX file reception along with other Bluetooth tasks, on real scenarios and
- to deploy an extension in real scenarios, installed on the Hotspot as a third-party developed extension.

During this evaluation phase, the Hotspot prototype includes a fully functional OBEX Receiver module and a Hotspot's extension developed with the objective of evaluating not only the extension mechanism but also the extension itself. The extension that was developed intends to be adapted to Wiimote-based interaction scenarios, as surveyed in Section 2.5. It eases connections with Nintendo's Wii console remotes[13] and its functionality is described in the next section.

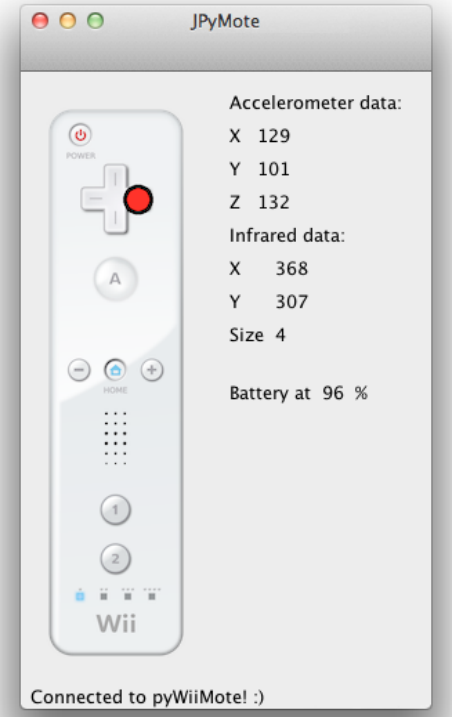

Figure 5.2. A screenshot of our prototype of an application that integrates with the Hotspot and identifies the data inputs of Wiimotes connected to the Bluetooth Hotspot.

[13]Nintendo Company, Limited. - `http://www.nintendo.com/`

Wiimote Extension

This extension turns the Hotspot on a bridge between the Bluetooth Nintendo's Wiimotes and a remote application that recognizes its gestures [40]. For each discovered Wiimote, the extension establishes two connections, in both directions: 1) a Bluetooth serial connection with the Wiimote and 2) a socket connection with the application that analyzes gestures and pressed buttons. After having established these two connections, the extension redirects to the application all the data received from the Wiimote stream. This stream includes the `(x,y,z)` values of the Wiimote accelerometer, the `(x,y,z)` coordinates of the location of the Wiimote and the list of buttons that are being pressed.

In our prototype, the application is a Java program that receives the data from that connection and illustrates the Wiimote status at the screen, just for validation. A screenshot of this application is seen in Figure 5.2. Our aim is to interpret the gestures and provide the results to an API that can be used by other applications, with the objective of deploying a real scenario to validate this model. A possible scenario is an interactive presentation at a museum or an artistic installation were users can interact with the environment using a Wiimote.

As explained in Section 4.3.3, as an extension starts, it receives an URL where to get the file with its configurations (refer to Section 4.4 for the rule that starts an extension). For this prototype of the Wiimote Extension, this file is an XML file with five elements:

- `rumbleTime` - defines the amount of seconds to enable the vibrator of the Wiimote when a successful connection is established,
- `sendTo` - defines the host name of the server (application) where to send the output from the Wiimotes,
- `tcpPort` - defines the Transmission Control Protocol (TCP) port of the above server.

In order to clarify this issue, please consider the following example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rules>
    <rumbletime>1</rumbletime>
    <sendTo>192.168.1.5</sendTo>
    <tcpPort>1234</port>
</rules>
```

This configuration file will make the Wiimote Extension to establish a TCP socket connection with a remote application that is running at the port `1234` of the server with the IP `192.168.1.5`. When a connection between the Hotspot and a Wiimote is successfully established the Hotspot establishes the TCP connection with the application server, and the Wiimote vibrates for 1 second. Through this connection, all the output from the Wiimote is redirected to the application for gesture interpretation.

Chapter 6

Conclusions

The main objective of this dissertation was the proposal of a new system component, centered on Bluetooth-based user interactions. The work was essentially focused on these interactions. We have surveyed related work with the objective of proposing an architecture of this system component - the Bluetooth Hotspot. We proposed and made its functionality available. Such component may now be used in actual systems and applications as a Bluetooth interface that interacts with users' mobile devices.

To evaluate the proposed architecture and integration model, prototypes were developed and deployed in real scenarios, gathering multiple and important results. These results had extreme value during the design of our component, but are also important as a case study for the involved projects.

Results

The designed integration model - that includes configuration rules, configuration processes, feedback logs and file resources on a RESTful philosophy - proved to work flawlessly with existing systems and applications. During the evaluation phase, these systems easily adapted to our Bluetooth Hotspot without any integration issue. Moreover, the objective of freeing the owner of the applications from Bluetooth-related issues could also be fulfilled, e.g: 1) application owners could retrieve the scan results in useful time, just by defining a limit for a full-cycle scan; 2) the delivery or reception of a file to/from users' devices could be easily performed without understanding how OBEX transfers works. In the future, with support from the extension mechanism, many other scenarios and situations can also be evaluated.

It was possible to conclude that, in most of the cases, the utilization of more than one Bluetooth interface installed on the Hotspot is mandatory. The performance of

Bluetooth tasks becomes really affected while a device scan is performed, which leads us to dedicate an interface just for this task, leaving the other(s) performing other device's interactions.

Design and implementation obstacles

Some implementation problems became an obstacle to the study and development of the Bluetooth Hotspot. OpenWrt Linux distribution proved to be extremely stable but the implementation of the Bluetooth stack (provided by BlueZ libraries) did not. Randomly, but sporadically, the Bluetooth stack loses the connection with the Bluetooth interfaces without any chance of recovery, forcing the system to be rebooted in order to access the device again. Despite the quickly reboot of OpenWrt on those kind of devices (around 30 seconds), this became a problem on real-time scenarios where users' could not wait so much time for an interaction. Nevertheless, these losses of connection also happened when using just the BlueZ tools without our software. On each evaluation phase, tests were performed on a development environment before running the prototypes inside the deployed Ethernet Linux routers. This allowed us to identify that these unexplained issues came from the OpenWrt environment and not from our implementation. We are currently in contact with the open source community of BlueZ and Python on OpenWrt, trying to fix this problem and to obtain a stable solution.

Future work

During this research, some usability issues were detected, that are out of the scope of our work, but were predicted by other related works: the increasing number of devices that are originally configurations as hidden devices or with short periods in the discovery state. Another typical problem was the obligation of some devices to pair with a code before receiving or sending a file from/to the Bluetooth Hotspot.

The duration of this work has not allowed to design and implement other important features related with scalability and security. Our integration model does foresee any security issue, but in a future phase we intend to design mechanisms of authentication, authorization and accounting. During our deployments we used TCP/IP firewalls and Virtual Private Networks (VPNs) to implement at least some of the security issue. In what concerns the system scalability, we foresee two design improvements: 1) the Hotspot Configuration Rules should be installed directly from the applications and 2) the Hotspots should interconnect and synchronize the rules between them, in multiple Hotspot scenarios. This would free the utilization of the Main Controller, simplifying

the architecture.

The relationship between a Hotspot and specific applications can be simple and easy to be implemented if both of them are deployed together, for the same end. However, if a Hotspot is already deployed, new applications do not recognize the presence and the availability of Hotspot(s) automatically. A mechanism to handle the registration of Hotspots is important in the future. This way, further applications may consult this service to discover the available Hotspots on the specific space they want to operate. This is the typical service that can be integrated in the Main Controller.

The source code of the Hotspot prototype developed, during this dissertation was published as an open-source project at Source Forge repository: `https://sourceforge.net/p/bluebox-hotspot/`. Providing the source code, it may empower not only this project, but many other projects in the area of Pervasive Computing. Along with the source code, we provide a package to be included on OpenWrt's software repository[1], to ease installations on new systems, using its package management system (`opkg`).

Conclusion

Overall, we showed that there is a space for a Bluetooth system component that can be easily shared and reused. We have described the main key design issues for such component and its system integration. We have deployed our component within different systems. It proved to have an acceptable performance in most of the predicted scenarios. Additional work has to be done to improve the integration process, such as the study of Hotspot registration and discovery process, but in our opinion, this is a preliminary result and contribution for the sustainability of Pervasive Computing industry.

[1] OpenWrt's software repository - `http://wiki.openwrt.org/doc/packages`

Bibliography

[1] Bluetooth core specification v2.0. technical report. Tech. rep., Bluetooth Special Interest Group, November 2004.

[2] Aalto, L., Göthlin, N., Korhonen, J., and Ojala, T. Bluetooth and WAP push based location-aware mobile advertising system. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services* (New York, NY, USA, 2004), MobiSys '04, ACM, pp. 49–58.

[3] Abowd, G. D. Classroom 2000: An experiment with the instrumentation of a living educational environment. *IBM Systems Journal 38*, 4 (2000), 508–530.

[4] Abowd, G. D., Ebling, M., Gellersen, H.-W., Hunt, G., and Lei, H. Guest editors' introduction: Context-aware computing. *IEEE Pervasive Computing 1*, 3 (2002), 22–23.

[5] Ahmed, H., EL-Darieby, M., Abdulhai, B., and Morgan, Y. Bluetooth and wi-fi-based mesh network platform for traffic monitoring. In *Transportation Research Board 87th Annual Meeting* (2008).

[6] Bär, H., Häussge, G., and Rössling, G. An integrated system for interaction support in lectures. *SIGCSE Bull. 39* (June 2007), 281–285.

[7] Bardram, J. E., Hansen, T. R., and Soegaard, M. Awaremedia: a shared interactive display supporting social, temporal, and spatial awareness in surgery. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work* (New York, NY, USA, 2006), CSCW '06, ACM, pp. 109–118.

[8] Baumann, S., Jung, B., Bassoli, A., and Wisniowski, M. BluetunA: let your neighbour know what music you like. In *CHI '07 extended abstracts on Human factors in computing systems* (New York, NY, USA, 2007), CHI EA '07, ACM, pp. 1941–1946.

[9] Camacho, T. A. D. Proximity sensing and context-aware content dissemination. Master's thesis, Department of Mathematics and Engineering of the University of Madeira, 2009.

[10] Cano, J.-C., Manzoni, P., and Toh, C.-K. Ubiqmuseum: A bluetooth and java based context-aware system for ubiquitous computing. *Wireless Personal Communications 38* (2006), 187–202.

[11] Chatschik, B. An overview of the bluetooth wireless technology. *Communications Magazine, IEEE 39*, 12 (December 2001), 86–94.

[12] Cheverst, K., Dix, A., Fitton, D., Kray, C., Rouncefield, M., Sas, C., Saslis-Lagoudakis, G., and Sheridan, J. G. Exploring bluetooth based mobile phone interaction with the hermes photo display. In *MobileHCI '05: Proceedings of the 7th international conference on Human computer interaction with mobile devices & services* (New York, NY, USA, 2005), ACM, pp. 47–54.

[13] da Cruz Bernardo, J. F. Bluetooth naming for situated interaction in ubiquitous environments. Master's thesis, Departamento de Sistemas de Informação, Universidade do Minho, 2009.

[14] Davies, N., Friday, A., Newman, P., Rutlidge, S., and Storz, O. Using bluetooth device names to support interaction in smart environments. In *MobiSys'09* (2009), pp. 151–164.

[15] Eagle, N., and Pentland, A. Social serendipity: Mobilizing social software. *IEEE Pervasive Computing 4* (April 2005), 28–34.

[16] Eagle, N., and (Sandy) Pentland, A. Reality mining: sensing complex social systems. *Personal Ubiquitous Comput. 10* (March 2006), 255–268.

[17] Fielding, R. T., and Taylor, R. N. Principled design of the modern web architecture. *ACM Trans. Internet Technol. 2* (May 2002), 115–150.

[18] Fitz-Walter, Z., Jones, S., and Tjondronegoro, D. Detecting gesture force peaks for intuitive interaction. In *IE '08: Proceedings of the 5th Australasian Conference on Interactive Entertainment* (New York, NY, USA, 2008), ACM, pp. 1–8.

[19] Fonseca, R., and Simões, A. Alternativas ao XML: YAML e JSON. In *XATA 2007 — 5ª Conferência Nacional em XML, Aplicações e Tecnologias Aplicadas* (February 2007), J. C. Ramalho, J. C. Lopes, and L. Carríço, Eds., pp. 33–46.

[20] Fresnedo, O., Iglesia, D. I., and Escudero, C. J. Bluetooth inquiry procedure: optimization and influence of the number of devices. In *CSN '07: Proceedings of the Sixth IASTED International Conference on Communication Systems and Networks* (Anaheim, CA, USA, 2007), ACTA Press, pp. 205–209.

[21] Harter, A., Hopper, A., Steggles, P., Ward, A., and Webster, P. The anatomy of a context-aware application. *Wireless Networks 8* (March 2002), 187–197.

[22] Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., and Jansen, E. The gator tech smart house: A programmable pervasive space. *Computer 38* (March 2005), 50–60.

[23] Jayaraman, P. P., Zaslavsky, A., and Delsing, J. On-the-fly situation composition within smart spaces. In *Proceedings of the 9th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking and Second Conference on Smart Spaces* (Berlin, Heidelberg, 2009), NEW2AN '09 and ruSMART '09, Springer-Verlag, pp. 52–65.

[24] Johanson, B., Fox, A., and Winograd, T. The interactive workspaces project: Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing 1* (April 2002), 67–74.

[25] Johansson, P., Kazantzidis, M., Kapoor, R., and Gerla, M. Bluetooth: an Enabler for Personal Area Networking. In *IEEE Network Magazine, Wireless Personal Area Network* (2001), vol. 15, pp. 28–37.

[26] José, R., and Bernardo, F. Extended bluetooth naming for empowered presence and situated interaction with public displays. In *3rd Symposium of Ubiquitous Computing and Ambient Intelligence 2008*, vol. 51 of *Advances in Soft Computing*. Springer Berlin / Heidelberg, 2009, pp. 57–65.

[27] José, R., Otero, N., Izadi, S., and Harper, R. Instant Places: Using Bluetooth for Situated Interaction in Public Displays. *IEEE Pervasive Computing 7*, 4 (2008), 52–57.

[28] Kindberg, T., and Fox, A. System software for ubiquitous computing. *IEEE Pervasive Computing 1* (January 2002), 70–81.

[29] LeBrun, J., and Chuah, C.-N. Bluetooth content distribution stations on public transit. In *MobiShare '06: Proceedings of the 1st international workshop on Decentralized resource sharing in mobile computing and networking* (New York, NY, USA, 2006), ACM, pp. 63–65.

[30] Mahato, H., Kern, D., Holleis, P., and Schmidt, A. Implicit personalization of public environments using bluetooth. In *CHI '08 extended abstracts on Human factors in computing systems* (New York, NY, USA, 2008), CHI '08, ACM, pp. 3093–3098.

[31] McCarthy, J. F., Congleton, B., and Harper, F. M. The context, content & community collage: sharing personal digital media in the physical workplace. In *Proceedings of the 2008 ACM conference on Computer supported cooperative work* (New York, NY, USA, 2008), CSCW '08, ACM, pp. 97–106.

[32] Nicolai, T., Yoneki, E., Behrens, N., and Kenn, H. Exploring social context with the wireless rope. In *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, R. Meersman, Z. Tari, and P. Herrero, Eds., vol. 4277 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 874–883.

[33] ONeill, E., Kostakos, V., Kindberg, T., Schiek, A., Penn, A., Fraser, D., and Jones, T. Instrumenting the city: Developing methods for observing and understanding the digital cityscape. In *UbiComp 2006: Ubiquitous Computing*, P. Dourish and A. Friday, Eds., vol. 4206 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 315–332.

[34] Peterson, B. S., Baldwin, R. O., and Kharoufeh, J. P. Bluetooth inquiry time characterization and selection. *IEEE Transactions on Mobile Computing 5*, 9 (2006), 1173–1187.

[35] Pietiläinen, A.-K., Oliver, E., LeBrun, J., Varghese, G., and Diot, C. Cityware: Urban computing to bridge online and real-world social networks. In *Applications, Technologies, Architectures, and Protocols for Computer Communication* (2009).

[36] Podnar, I., Hauswirt, M., and Jazayeri, M. Mobile push: Delivering content to mobile users. In *in Proceedings of the International Workshop on Distributed Event-Based Systems (ICDCS/DEBS'02* (2002), IEEE Computer Society, pp. 563–570.

[37] Rashid, O., Coulton, P., and Edwards, R. Providing location based information/advertising for existing mobile phone users. *Personal Ubiquitous Computing 12*, 1 (2008), 3–10.

[38] Sanchez, J.-M., Cano, J.-C., Calafate, C. T., and Manzoni, P. Bluemall: a bluetooth-based advertisement system for commercial areas. In *PM2HW2N '08: Proceedings of the 3nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks* (New York, NY, USA, 2008), ACM, pp. 17–22.

[39] Satyanarayanan, M. Pervasive computing: vision and challenges. *IEEE Personal Communications 8*, 4 (August 2001), 10–17.

[40] Schlömer, T., Poppinga, B., Henze, N., and Boll, S. Gesture recognition with a wii controller. In *TEI '08: Proceedings of the 2nd international conference on Tangible and embedded interaction* (New York, NY, USA, 2008), ACM, pp. 11–14.

[41] Seixas, H., Salgado, N., and José, R. Wiinteraction: A study on smart spaces interaction using the wiimote. In *INFORUM* (2009), pp. 395–406.

[42] Sheridan, J. G., Price, S., and Pontual-Falcao, T. Wii remotes as tangible exertion interfaces for exploring action-representation relationships. *Workshop on Whole Body Interaction, CHI* (April 2009).

[43] Young, S. Bluetooth traffic monitoring technology. *Center for Advanced Transportation Technology, University of Maryland* `http://www.catt.umd.edu/documents/UMD-BT-Brochure_REV3.pdf` (September 2008).

Acronyms list

UML Unified Modeling Language

HTTP Hypertext Transfer Protocol

FTP File Transfer Protocol

URL Uniform Resource Locator

USB Universal Serial Bus

XML Extensible Markup Language

YAML YAML Ain't Markup Language

REST Representational State Transfer

SOAP Simple Object Access Protocol

IR Infrared

STL Standard Template Library

CF Compact Flash

WAP Wireless Application Protocol

API Application Programming Interface

SMS Short Message Service

RFID Radio-Frequency IDentification

MAC Media Access Control

PDA Personal Digital Assistant

IP Internet Protocol

LED light-emitting Diode

ID Identification Number

NAT Network Address Translation

RAM Random Access Memory

TCP Transmission Control Protocol

RPC Remote Procedure Call

POS Point of Sale

GPRS General Packet Radio Service

SDK Software Development Kit

SSH Secure Shell

GUI Graphical User Interface

CPU Central Processing Unit

VPN Virtual Private Network

AP Access-Point

BEN Bluetooth Extended Naming

UCI Unified Configuration Interface

OBEX OBject EXchange

PAN Personal Area Network

BNEP Bluetooth Network Encapsulation Protocol

HID Human Interface Device

RFCOMM RFCOMM, Serial Cable Emulation Protocol

L2CAP Logical Link Control and Adaptation Protocol

SDP Service Discovery Protocol

HCI Host Controller Interface

SPP Serial Port Profile

SPP-over-IP Serial Port Profile over IP