Proximity Sensing and Context-Aware Content Dissemination

in Partial Fulfilment of the Requirements for the Degree of
Master of Informatics Engineering

Presented to the Department of Mathematics and Engineering of the
University of Madeira, Portugal by

Tiago Alexandre D. Camacho

in September 2009

Supervisor: Prof. Vassilis Kostakos, PhD

Declaration

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Up to now, this thesis was not published or presented to another examinations office in the same or similar shape.

Tiago Alexandre Dias Camacho

Abstract

With the incorporation of wireless modules into mobile equipment, new opportunities for sensing people and offer them innovative services arise. In this thesis we present a prototype system that uses Bluetooth to perform proximity sensing and context-aware content dissemination. After analysing the sensed data, our system disseminates content by means of a Service Specification Language (SSL) which can describe a set of rich context-aware services. Our results show the usefulness of our system, as we demonstrate that it is a low-cost and flexible alternative to more expensive methods of proximity sensing and content dissemination.

Acknowledgments

I would like to start by thanking my supervisor, Professor Vassilis Kostakos, for his support and inspiration throughout this thesis. He always provided me with good advises, and was always available when needed. I learned a great deal from him, and for that I will always be thankful.

I would also like to thank Claudio Mantero for his helpful comments and advices, and for making infrastructures available for testing.

Finally, I would like to thank my family and close friends, with special emphasis to my parents and brother. Without their help and support, both psychological and financial, this thesis would never be completed.

Table of contents

1 Introduction

With the advance of wireless technologies and the maturation of ubiquitous and context-aware computing, researchers are becoming more interested on the way these fields influence our daily lives. Although we are witnessing the widespread deployment of wireless technologies, still the full exploitation of this reality requires for the design and development of new infrastructures that exploit these technologies in favour of users.

One of the potential areas on which wireless technologies are useful is in the process of presence determination. Some of these technologies, such as Bluetooth, are termed ideal for this purpose, as they require no permission by users to establish low-level connections between equipment, therefore making the determination of people presence a non-intrusive process.

Effectively, with the addition of wireless components into mobile equipment, people carry with them components which can be sensed. As we lengthen the areas on which we deploy sensor nodes, we create the notion of a omnipresent sensing infrastructure capable of determining people's presence, using this information for, among many other possibilities, providing users with a set of innovative context-aware services.

In this work we discuss a prototype system used for proximity sensing and context-aware content dissemination. The system is composed by several distributed components, on which the mobile ones are termed as stations. These may be installed at various locations, and are constituted mainly by two components: a device scanner and a content dissemination infrastructure. The system works by continuously scanning for nearby Bluetooth enabled devices, consequently using this information to feed the content dissemination infrastructure. The determination of whether content is to be disseminated is done through the analysis of a service queue. Also, the system makes use of *Service Specification Language* (SSL) to create logical representations (e.g. services) with distinct types of content and restrictions associated with them. Services will therefore hold the information necessary for the dissemination system to determine to which destinations, and under what conditions, is content to be delivered to.

1 Introduction

The structure in which we present our work is as follows. In **Section 2** we discuss the factors which motivated our work, and exemplify potential benefits of the system. **Section 3** presents the opinions collected from public transportation users, and the results and analysis which influenced some design aspects of our system. **Section 4** shows the fields which are inter related with our work, as we first present a more broad perspective of related work, before moving into a more in-depth description of specific areas. **Section 5** is where we describe our system. We use a top-down approach in this section, starting by a more abstract description, before moving into more detailed component definition. In **Section 6** we have the results and analysis of our pilots, and also present test subjects opinions of a sample service. Finally, in **Section 7** we present our conclusions and discuss possible future directions of our work.

2 Motivation

In this section we present the main factors of motivation that led to the development of our work. We start by discussing the importance of proximity sensing, before moving to content dissemination. We thereafter present an overview of our system, and finalize the section with the demonstration of potential system benefits.

2.1 Proximity Sensing

Depending on the nature of the information, some types of data are inherently difficult to be automatically determined. An example is seen in public transit companies, that to this day still rely on manual observation and incomplete data derivation for the determination of passengers' entry and exit points. These methods, used to determine important transportation analysis structures, lead to poor results, as both freshness and quality of data are undermined [Kostakos 08].

With the advent of technologies such as the *Global Positioning System* (GPS), the determination of positioning information has been facilitated. Nowadays, many vehicles, and even mobile equipment, have an incorporated GPS module that provides reliable localization determination. Still, long range satellite and terrestrial positioning systems are not suitable for all environments. Technologies such as GPS, Loran, and Omega are inadequate for both indoor and pin-point positioning, and therefore the diversity of fields on which they may be used is limited [TC 03].

Furthermore, there are times that the localization information that we wish to capture is in relation to another object. In this type of situation, we are not interested in the global position, but only in the relative one. This type of sensing that determines if two objects are close to each other is referred to as *proximity sensing*, and technologies such as infra reds or radio-frequency may be used to accomplish it.

As radio-frequency based wireless technologies continue to spread throughout mobile equipment, new fields of opportunity for proximity sensing

arise. Bluetooth, for example, is seen as a good low-cost and reliable choice to determine if two objects are in the vicinity of each other. The widespread adoption of the technology has potentiate the sensing of Bluetooth discoverable equipment. If we deploy scanners throughout a target area, the inquiry mechanism of the technology allows us to determine the existence of nearby Bluetooth enabled devices. Furthermore, additional positioning determination is possible, as the technology allows for approximation of relative distance by means of signal strength measurement. All of this is done non-intrusively, as these operations dispense human intervention.

The array of opportunities that advent from the determination of relative localization of mobile equipment using such a low-cost technology are manifold, as it can be used for simulation purposes, environment affluence classification, passenger counting, and context-aware content dissemination.

2.2 Content Dissemination

Usually, content dissemination happens in a non-personalized manner. For example, the information that is passed to people by means of an electronic board is not personalized. These dissemination components, present in many types of settings, broadcast information in a very restricted manner. Not only the content that is disseminated is limited, but also the way content is presented is restricted by the hardware limitations. This fact is not related solely to electronic boards, as many other dissemination systems lack flexibility. The PORTAL system [BCG 05], a city-wide network which provides public transit bus information access, is another example of a content dissemination system which uses specialized components. The system works by installing several dedicated machines throughout the city, so users may query for public transit related information. The need to have a high-cost, static components installed at a city-wide levels, solely for presentation purposes, challenges the real advantage of such system.

The use of wireless technologies for disseminating content is seen as advantageous. People already carry with them mobile equipment that hold the capability for presenting various kinds of information. As users are the ones that own the components on which content is stored and accessed upon, we delegate the presentation and personalization aspects to users, removing the

necessity of specialized dissemination components. We exploit the capabilities of users' devices, and establish the infrastructure on which a set of innovative and personalized services may be created and offered to users.

Given the ideas of proximity sensing and content dissemination, and how they can be used to better serve users, we proceed with an overview of our system, and how it relates to these two subjects.

2.3 System Overview

In this work we describe a system that performs proximity sensing for enabling context-aware content dissemination. The system, composed by a set of elements, works by installing several stations in areas of interest. These stations are usually small computers (but not restricted to) and are equipped with two Bluetooth dongles. One dongle, the *scanner*, is responsible for the capture of surrounding Bluetooth enabled devices, as the other has the responsibility to deliver content to end-users. A configuration with multiple delivery dongles is also supported by the system, as it is a configuration with solely one dongle.

The scanner feeds the information to the *service scheduler*, the component which has the responsibility to determine if a service is to be executed. The scheduler holds a queue of service objects and, by using contextual data received from the scanner, analyses services to determine if the constrains on which they were built upon are met. When such happens the scheduler will order for service execution, and consequent dissemination of content to the destination.

Service objects are composed by a set of *flags*, which determine the restrains and content associated with the object. Flags include date and time of execution, location, content location, class of device, time in range, and destination. All the indicated flags upon creation must be in conformity so that the scheduler triggers a service for execution.

Stations are installed at arbitrary locations, and use a wireless connection to access the Internet. Upon initialization the stations contact the *broker* component, so they register themselves with the system. This is done so we refer to stations by name, removing the burden of numerical IP

addressing. This is also done because the broker is present as an intermediary element on all connections to and from stations.

Also, a *central infrastructure* exists. Composed by several components, it holds a database containing information which is partially replicated locally at the stations. The central infrastructure also provides for station synchronization and overall remote management.

Given the overall description of our system, it is now pertinent to discuss the potential gained benefits of deploying it.

2.4 Potential System Benefits

Basically, our system has two objectives: 1) sense nearby Bluetooth enabled devices, and 2) disseminate content according to contextual data. The process of automatically sensing devices has uses in many situations. Depending on the installation environment, stations may use the proximity sensing information to achieve objectives other than feeding the dissemination infrastructure.

For example, Bluetooth technology is useful for counting public transit passengers. With the installation of our system on public transit buses, we may exploit passengers' Bluetooth enabled devices to determine entry and exit points, as defined in [Kostakos 08]. Furthermore, the collected data would be useful for other related purposes, such as bus travel time estimation [SF 03].

Another possibility is to implement stations at environments such as commercial areas or public transit bus stops. The collected information could then be used to perform estimations on the number of people in the area. This information would also be useful for area assignment according to its affluence, or for determining how much time people spend in the area.

Furthermore, the installation of the system enables for a rich infrastructure that enables the creation of context-aware services. Several kinds of settings are allowed, as it is possible to construct services which can be personalized according to contextual information, individual preferences, and even equipment details.

2 Motivation

As data is fed by the scanner, the system has the ability to determine if action is to be taken. For example, if one device is seen continuously for a long time in a commercial area, maybe that person is doubtful on what to buy. The system would “sense” this indecision and send a recommendation to the user. Another possible application would be to offer a service that disseminates news headlines to users while they wait for a bus to arrive, or are sitting in a café.

In sum, the advantages of our system are considerable, as we enable for a low-cost and reliable method of proximity sensing and context-aware service offering. The settings on which the system may be installed are unlimited, as its flexibility frees the system from environmental restrains.

One of the many areas of interest in which our system is seen as potentially beneficial is public transit. In order to gain insight on people's interests of content dissemination at public transit infrastructures, we devised a questionnaire and distributed it to public transit users. The results and conclusions draw from this questionnaire is what we discuss in the ensuing section.

3 Public Transit Passengers' Opinions

As people daily commute from work and school to their homes, many use public transportation. Accounting for the time people both wait for and use this kind of transport, passengers can actually spend hours each day at public transit infrastructures. It is this premise that makes the deployment of a heterogeneous information dissemination system at these infrastructures interesting.

From the beginning of our work we recognized public transit as an area of great interest and potential, on which our system could be helpful. As such, and before we delved into the design and development of our system, we devised a questionnaire and distributed along public transit users. The objective was to understand their habits, the kind of equipment they use, and their personal preferences.

In this section we present the questionnaire, its results, and the inferred conclusions obtained through our analysis. Several interesting facts were brought to our attention, some of them already corroborated by previous research. More importantly, the questionnaire helped us validate important aspects of our system.

3.1 Data Summary

We begin this section by presenting the collected data. Starting 15/10/2008, data was collected until 21/10/2008. Questionnaires were distributed at several locations and times, as it is seen in Table 1. Our sample size is 105 respondets, 51 are female (48.5%), and 54 are male (51.5%). The age distribution of the individuals is depicted in Figure 2, where the 20-30 age group is the most common.

The questionnaire collected demographic data, public transit usage habits, mobile equipment usage and Bluetooth related information, services of possible interest to users, and the preferred way users would like to access content.

3 Public Transit Passengers' Opinions

The majority of the respondents were regular users of public transportation (59%), as is seen in Figure 1. Most respondents (73%) reported they wait between 5 and 15 minutes for the bus arrive (Figure 3). Therefore, we can state, with 95% of certainty, that between 64.5% and 81.5% of public transportation passengers must wait 5 to 15 minutes for the bus to arrive.

More than 80% of respondents reported that they use their portable communication equipment while waiting for the bus. Thus, we can state that between 73.5% and 88.5% of public transportation users occupy some of their waiting time by operating their devices. Also, it seems that often respondents use their devices for messaging purposes (60%). On the other hand, accessing on-line content doesn't seem popular among the respondents (4.8%).

About 73% of respondents reported that they have a mobile device which supports Bluetooth (Figure 4). Based on this data, we can state (with 95% of certainty) that 64.8% to 81.8% of public transportation users have a Bluetooth capable mobile device.

Nevertheless, only about 11.5% of users stated that they have their Bluetooth mode enabled (Figure 5). This suggests that 5.3% to 17.5% of public transportation users have their devices' Bluetooth enabled. This data is coherent with previously obtained percentages [OKKSPFJ 06]. Security (35%) and power consumption (27%) are the main reasons for disabling Bluetooth discoverable mode, as is seen in Table 3.

Data about the preferred way of accessing Bluetooth service was inconclusive, as depicted in Figure 6. Respondents made clear that they don't want to be pushed uninteresting information unconditionally (3.8%), but are receptive to receive pushed content if it is or their interest (33.6%). They were also split between being pushed content with previous registration (31.7%), and retrieving the content for themselves (i.e. pulling content) (30.7%).

Given the presentation of the collected raw data, we now proceed to the discussion of the results.

		Day (October 2008)							
		15	16	17	18	19	20	21	Totals
Time of Day	10:00	0	4	0	0	0	5	0	9
	11:00	3	0	0	0	5	0	0	8
	12:00	0	0	0	0	0	1	0	1
	13:00	8	0	5	0	0	0	0	13
	14:00	0	0	0	6	0	0	0	6
	15:00	10	0	4	0	0	0	0	14
	16:00	14	7	5	0	0	0	0	26
	17:00	1	0	3	4	0	0	6	14
	19:00	0	3	6	0	0	0	2	11
	20:00	0	0	0	0	0	0	3	3
Totals		36	14	23	10	5	6	11	105

Table 1: Answered questionnaires distribution

	Phone Calls	Messages	Entertainment	Internet	Other
Replied	43 (41%)	63 (60%)	37 (35%)	5 (4,8%)	0 (0%)

Table 2: Respondents' device usage at bus stops

	Security	Power Consumption	No reason	Other
Replied	37 (35%)	28 (27%)	10 (9,5%)	0 (0%)

Table 3: Reasons for disabling Bluetooth

	Bus Schedules	Buses Arrival Times	Entertainment	Nearby bus stops locations	News	Other
Replied	68 (64,7%)	75 (71,4%)	37 (35,2%)	35 (33,3%)	32 (30,4%)	2 (0,02%)

Table 4: Respondents services preferences at bus stops

	Bus Schedules	Buses Arrival Times	Entertainment	Nearby bus stops locations	News	Other
Replied	20 (19%)	61 (58%)	39 (37%)	33 (31%)	45 (43%)	1 (0,009%)

Table 5: Respondents services preferences while traveling

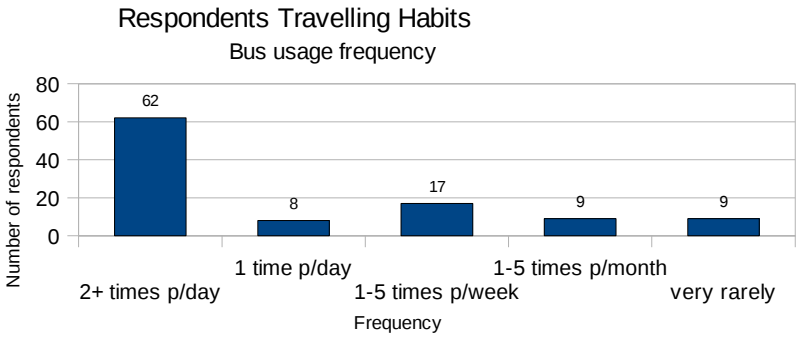

Figure 1: Respondents bus usage frequency

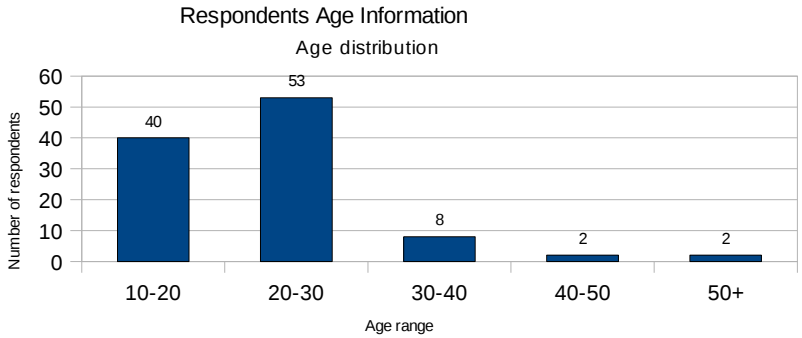

Figure 2: *Respondents age distribution*

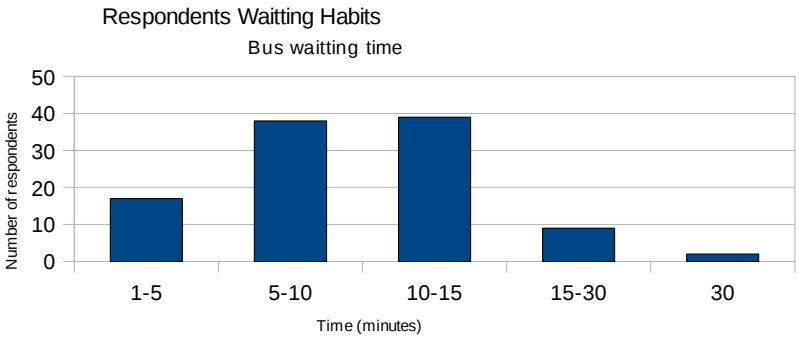

Figure 3: Respondents waiting time at bus stops

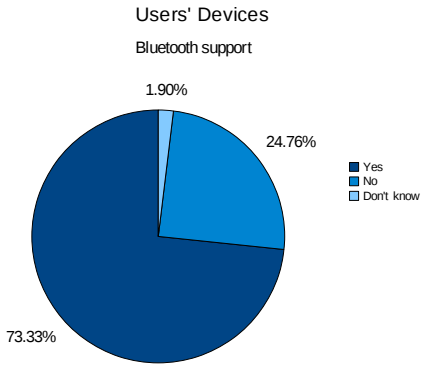

Figure 4: Devices' Bluetooth support

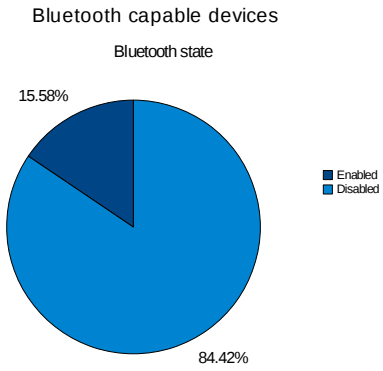

Figure 5: *Enabled Bluetooh devices*

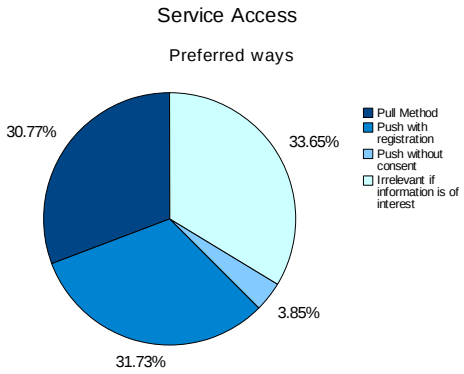

Figure 6: Respondents preferred way of service access

3.2 Discussion

In this section we present the discussion of the previous results. We use as basis for this discussion the variables associations presented at **Appendix A2**. By describing associations and stating their major influence factors, we dervied useful information that aided us in the design and development processes of our system. Images and tables reffered along this sections are located at **Appendix A3**.

3.2.1 Location

- A relationship between the location where the questionnaire was delivered and the choice of a service that indicates nearby bus stops exists. Our analysis shows people that answered the questionnaire at UMa location are more inclined to want a service that indicates nearby bus stops localizations. By looking at Figure 71, we see a standardized value of 2.23 and a percentage of nearly 48%, clearly indicating that such an association exists. This could be due the fact that more that 50% of the respondents at UMa said that they wait between 10 to 30 minutes for the bus the arrive, possibly leaving them more interested in catching an alternative bus at a nearby bus stop.

3.2.2 Demographics

- Sex and device usage for messaging purposes are related. Figure 72 indicates that women seem more inclined to send/receive messages. Almost 73% of the female respondents claimed that they use their device for this specific purpose. An hypothesis that explains this is that women are more prone to use this kind of service than men.
- An association exists between sex and wanting a service that indicates nearby bus stops. Almost 33% of the males stated that a service of this kind interests them (Figure 73). Therefore, it seems that men are more prone to know nearby bus stops localizations. This again could be related to the time that male respondents wait for the bus. As 50% of males wait between 10 and more than 30 minutes, this could lead them to look for alternative bus stops.

- Age and device usage at bus stops are related. Table 8 shows us that respondents of 10-20 and 20-30 age groups make constant use of their mobile equipment while they wait for the bus. More than 50% of respondents of the 10-20 age group wait between 10 and more than 30 minutes. Also, around 39% of respondents of 20-30 age group wait between 10 and 30 minutes. An hypothesis that time is an influencing factor on device usage can be raised to explain this.
- Additionally, age and the use of mobile equipment to send messages are related. By observing Table 9, we verify that standardized residuals of -2.88 and 2.35 indicate that respondents that belong to the 10-20 age group are more inclined to use their equipment for messaging purposes. We could state that younger people tend to have less disposable income, therefore using messaging as a more economic way of communication.
- Our analysis showed us that age and wanting to access a news service while at the bus stop are related. By looking at Table 10 we observe that younger respondents are disinterested in a service as this. On the contrary, older respondents saw this as of interest to them. We could argue that, in general, younger users are less interested in certain kind of services, such as news.

3.2.3 Bus Usage and Waiting Time

- An association exists between usage frequency of public transportation and to disable Bluetooth for no concrete reason. Looking at Table 13 we observe a standardized residual of 3.05, which indicates that respondents that use the bus 1 to 5 times a week, are more prone to disable their equipment' Bluetooth for no specific reason. Maybe these respondents simply disable Bluetooth as they see no purpose in it.
- Our analysis showed that waiting time and device usage are related. Our statistical data show us that people who wait between 5 and 10 minutes are less prone to use their mobile equipment. Also, there is a tendency for device usage as waiting time increases (Table 12). This data comes to substantiate previous results which indicated that device usage is influenced by waiting time.

- Waiting time is also related to device usage for messaging purposes. Our data indicates that people who wait between 5 to 10 minutes are less prone to use their device to send/receive messages (Table 11). This could be age related, since only 25% of the respondents of the 10-20 age group wait 5 to 10 minutes for the bus.

3.2.4 Mobile Devices Practices

- Device usage and wanting to access a news service at bus stops are related. Concretely, people who use their equipment to make/receive phone calls are more prone to want a news service (Figure 74). This could be related to age, because about 70% of respondents who use their equipment to phone have ages raging between 20 and 50+ years.
- Also, device usage is related to wanting a news service while travelling. Again, we see that users who use the device for phoning purposes are more prone to select the news service (Figure 75), which indicates a age related factor.
- People who use their device for messaging are also interested in using it for entertainment purposes. Figure 76 show us that indeed there is a tendency for those who don't use the device for messaging purposes to not use it for entertainment purposes. As seen previously, younger respondents tend to send more messages (Table 9), therefore we can assume that these same respondents are more prone to use their device for entertainment purposes.
- Device usage and disabling Bluetooth for security are related. This is seen in Figure 77, where people who send more messages are more prone to disable their devices' Bluetooth discoverable mode for security reasons. Again recalling Table 9, we can assume that this could be related to age, as younger people are more security-aware in regard to Bluetooth.
- Sending messages while at the bus stop relates to wanting a news service in the same situation. Figure 78 data indicates that those who use their device for messaging are less interested in a news service at the bus stops. Again, and recalling Table 8 and Table 9, we can state

that this is related to age, where younger users are less interested in this kind of service.

- Disabling Bluetooth for power consumption reasons is related to device usage. More concretely, those respondents who used their device for entertainment purposes are more inclined to disable Bluetooth for power consumption reasons (Figure 79). We can assume that entertainment software tends to make devices' battery consume faster, therefore alerting users of the implications of Bluetooth on power consumption.

3.2.5 Types of Services

- Wanting to access detailed bus schedules while at bus stops is related to wanting to know the arrival time at the destination stop. By looking at Figure 80, we observe that people who don't chose to know detailed bus schedules also seem less interested in knowing the arrival time at the destination stop. Furthermore Figure 81 shows that people who don't want to access detailed bus schedules at bus stops, also don't want to access while travelling. We can only assume that these people aren't interested at all with time issues.
- Wanting to know nearby bus stop locations is related to wanting the same service while travelling (Figure 82). We could argue that these people are cautious and would like to know where they could alternatively take a bus in case they need it.
- Wanting to know bus stops localizations while at bus stops is related to wanting a news service while travelling. Figure 83 show us that an association between these two variables exists, as people tend to choose both of these services. An hypothesis is that these are people who like to be informed.
- Wanting a news service while waiting for the bus is related to wanting the same service while travelling. Data indicates that those who chose one are likely to chose the other (Figure 84). Looking back at Figure 82, we could argue that people like to have access to similar services both in bus stops and while they travel.

- Those who want to know detailed bus schedules while travelling also seem interested in knowing their destination arrival time (Figure 85). We could argue that these people are interested in time and schedule issues (contrary to what we saw in Figure 80 and Figure 81).

3.3 Implications

Having presented the data and its analysis, we now present the design implications that were inferred from the questionnaire.

As verified in **Section 3.1**, respondents demonstrated their interest in knowing buses time related information. Specifically, 71.4% of the respondents viewed this as an important factor. Also, 64.7% stated that having access to detailed bus schedules is a useful feature. Therefore, it seems that such services would be of interest to users.

Additionally, respondents also seemed interested in specific services as they travel. The data demonstrates that there is a preference for accessing services that help time pass as passengers travel. Such tendency is justified by the percentage of respondents who stated their interest in having access to news (43%) and entertainment (37%) services. There was also a preference by the majority of the respondents (58%) for knowing the time remaining until they reached their destination. It seems clear that the system should be flexible enough to support various kinds of services.

Interesting, and with direct relation to the previous inference, is the fact that in **Section 3.2.5** we demonstrated that there is an association between wanting the same types of services both while waiting for the vehicle as while inside it. Such particularity leads us to take special attention in the flexibility of our system, as we believe that the implementation and specification of services should be sheltered from contextual factors.

Section 3.1 also demonstrates that respondents aren't interested on being pushed information unconditionally, a fact already confirmed by previous research [AGKO 04, PHJ 02]. On the other hand, respondents seem indecisive as to the actual preferred way of information access. The data indicates there is a split preference (of about 30% each) between user initiation (i.e. a pull method), service registration (i.e. pull method with previous registration), and simply not caring (as long content is of

interest).This indicates that various methods of content dissemination should be supported.

Given the implications that the questionnaire had in our system, we proceed to the review of the state of the art and related work.

4 State of The Art & Related Work

Throughout this section we present and review a number of fields that relate with our work. The purpose is to review concrete subjects, identify their foundations, and specify why they are important to us.

We begin with a horizontal review directed at more encompassing areas. Information Capture & Dissemination, Intelligent Transportation Systems, and Distributed Systems are example themes we present. The purpose is to first contextualize our work at a higher abstraction level, before drilling into more specific subjects.

Thereafter, our discussion will be directed at the areas of Bluetooth and Context-Aware Computing. Aspects will be reviewed more thoroughly, as our work is more closely related to these areas.

4.1 Broad View

By first providing a broad view we hope to better contextualize our work. As several fields contribute to our system specification, we fell that multiple aspects of dissimilar nature need to be discussed.

4.1.1 Information Capture & Dissemination

The capture or collection of information has always been viewed as of extreme importance. Possessing data about a subject is what enables us to study and better understand it. Without it, our view of specific subjects would be limited and incomplete, as the ability to perceive possible implications, and determine the influence of external factors would be diminished. The field of statistics is the most well know example of this. Information collection is the pillar that supports statistical analysis and inferences. For obtaining conclusions about a survey, for example, data must be collected and analysed.

Historically, information collection has been done by means of human intervention. The most common situation is when specialized individuals are responsible for collecting other individuals' information. Optionally, it can be

the own individual on which information needs to collected that reports back to specialized personnel, as is common in several types of studies. Or, taking another example, there may be a manual observation of some phenomenon, as there is when manual methods are used for counting passengers that alight a bus.

Problems associated with manual methods of information capture and collection are mainly associated to their cost, and to some point to their limited nature. Hiring individuals to collect information of thousands of persons is an expensive process. First, it is expensive at an operational level, as individuals are usually hired externally to the company. Secondly, information is easily outdated, as manual collection is done sporadically and with pre-determined goals in mind, and therefore fails to provide both an up-to-date view on the subject on which information was collected, as well as using it on other subjects. Thirdly, it lacks accuracy, as the sample size obtained with manual data collection tends to be significantly smaller than with automatic methods. Finally, manual information capture is limited in its scope, as it unsuitable for several types of activities, such as localization.

The term automatic information capture is defined as the action of identifying objects and collecting data about them in an automated way [VK 07]. Naturally, and contrary to manual methods, automatic techniques utilize computers to accomplish this task. The information is collected and posteriorly manipulated using electronic means, therefore dismissing human intervention altogether. The objective is to improve the efficiency of data collection as a process, which translates to reduced costs, more accurate results and overall is seen as beneficial to organizations. Automation of data capture is already seen as paramount in civilized countries, as several fields such as health informatics depend heavily on it [Norris 02].

Another common function for which information systems are used is for content dissemination. Although some authors define information dissemination solely as the act of *pushing* data to users, in this document we define information dissemination as the act of delivering content, independently of the way this is accomplished. Tan & Ooi [TO 00] share this view, as they define information dissemination as a process achieved either by the user's request (e.g. *pull*), or by the source's own initiative (e.g. *push*). Both

these methods have their peculiarities, as each one of them is more suitable for delivering content under different contexts.

Reportedly, the push model is more efficient when a large number of clients is present, and the pull model is more indicated for a small number of clients [TO 00, AFZ 97]. The justification is that the pull model depends heavily on the client-server model, and therefore is, at an architectural level, more prone to efficiency problems due to the fact that it must potentially handle a large number of requests simultaneously. The push model is sheltered from this limitation, as it is independent of the number of clients listening for content. The content is disseminated using broadcasting protocols and algorithms, and therefore the efficiency of the process is shielded from eventual misses that may occur. Of course this last assumption is made by with the presupposition that we are using technologies that support broadcasting, which is not always the case. Also, it is not always suitable to use unreliable methods for content delivery, as important information such as delivery success rates are very difficult to collect.

It is currently known that it is beneficial to conjugate the two described models of information dissemination. This is easily understandable, as each one works better when certain conditions are met. Therefore, it seems only logical to assume that merging the two of them would be advantageous. A generic architecture of this kind is present by Tan & Ooi [TO 00], as they define it as an *integrated dissemination model*. Acharya *et al.* also proposes an integrated model based on the *Broad-Cast Disk* approach [AFZ 95]. It uses both push channels and a point-to-point pull channel that they define as a *pull-based backchannel* [AFZ 97]. This pull channel would then be used to send requests to the content distribution source.

Our work relates to the notions here presented. The system functions by installing stations at target areas, automatically capturing Bluetooth enabled devices. Furthermore, the system uses this information to feed the content dissemination infrastructure that uses a push-only method to deliver content to users.

One of the many areas where our system can be useful is in the field of public transportation. In fact, information capture and dissemination using technological innovations in transports is a field that gathers much interest,

and may be seen as being part of the broader field of *Intelligent Transportation Systems* (ITS). For this reason, we discuss ITS in the ensuing section.

4.1.2 Intelligent Transportation Systems

ITS is a set of interrelated fields whose purposes are, among others, to achieve better transportation efficiency and diminish the environmental impact that self-propelling vehicles have [FJMFC 01]. ITS is of particular sensitivity to scientific and technological innovations, which can be exploited advantageously in transportation related areas. It is usually said that the global purpose of ITS is to use appropriate technology to add “intelligence” to infrastructures, vehicles and their users [FJMFC 01]. ITS does gather enormous worldwide interest both from transportation professionals, the automotive industry and from governmental entities alike.

Usually we can divide the number of fields that constitute ITS in six distinct categories, which may be further sub-divided into other categories [Shibata 99]. A field of special interest to us within ITS is that of *Advanced Public Transportation Systems* (APTS) and its sub-fields *Automatic Vehicle Location* (AVL) and *Automatic Passenger Counting* (APC). Many public transit companies already employ the referred APTS techniques, being very common to achieve AVL using *Differential Global Positioning Systems* (DGPS) [Zhao 00, LHHR 00]. The vehicles are usually equipped with GPS modules that estimate positioning. Posterior differential GPS [MOJ 95] correction is applied, and this positioning data is then uploaded to a central infrastructure by means of a wireless communication channel, such as *General Packet Radio Service* (GPRS) link [Zhao 00]. Subsequent data manipulation usually happens, as the received information serves as input for systems whose purpose may be to track down vehicles and provide times of arrival estimations. An example is the BUSVIEW graphical system [MD 01]. Also, and with the propagation of electronic travelling titles, most public transit companies employ APC by simply reading passengers' tickets or travelling cards. Such method of APC is considered infeasible in public transit vehicles such as buses and trains, simply because exit points are not usually captured. As a solution for this problem, other techniques such as pressure sensors and light beams [HQR 94] or passenger count through image interpretation [Wilson 06] may be

implemented. Another option is to enforce passengers to present their travelling titles at exit points. These solutions are far from optimal, as forcing title presentation is considered to be non-practical, and use of sensors and cameras is bound to have a large operational cost. A valid low-cost and reliable solution is that presented by Kostakos [Kostakos 08]. A Bluetooth scanner is implemented aboard a bus, as it perpetually scans for nearby devices. Posteriorly, the gathered data is interpolated with the public transit AVL data, and both passengers' entry and exit points are successfully inferred. Results are representative, and intrusiveness is non-existent.

Our system is somewhat related to the concepts previously described. We could possibly install the system both inside public transports, as in public transit infrastructure, such as bus stops. The installation in public bus stops would be interesting, as it would give us the ability of studying passengers' waiting habits, as well enable for the dissemination of a rich set of context-aware services, such as estimated transport arrival time, real-time vehicle position determination, among many others. Additionally, installing the system inside public transportations, such as buses would provide for the derivation of O/D matrices, as described in [Kostakos 08].

With this in mind, it becomes interesting to present and describe O/D matrices and their importance.

4.1.3 Origin-Destination Matrices

Origin-Destination (O/D) Matrices are an essential tool in transportation analysis. This structure has strategical importance in transit agencies due its relevance in both analysis and planning activities. O/D Matrices used in public transportations are simply structures that contain information about the passengers' flow along the various nodes that constitute the network.

The difficulty of estimating O/D Matrices for public transportation is particularly related to the kind of vehicle. Some types of transports such as subways make the derivation of O/D Matrices easier, as we can capture entry and exit points automatically through *Automated Fare Collection* (AFC) mechanisms. On other kinds of transports such as buses and trains, this derivation is not as straightforward and reliable as we would like it to be.

What is usually observed in such type of public transit vehicles is the use of manual methodologies to perform O/D Matrix estimation. Data is collected through the execution of a survey, and by applying one of several possible statistical techniques the O/D Matrix is inferred. An overview on the kind of statistic methods normally used is seen in [Ben-Akiva 87]. The main issues with this kind of estimation are related to the high operational costs of executing the surveys. For this reason, and for the fact that surveys are not actually executed with O/D Matrix derivation in mind, these tend to happen very sporadically, and fail to reflect fluctuations that possibly happen in a timely manner [Cui 06].

Additionally, and when entry points are automatically collected, there is the possibility of O/D Matrix estimation using only passengers' entry points information. This kind of situation (e.g. exclusive automatic entry point collection) is widely seen both in trains and buses, as usually these transportations enforce title presentation at or before vehicle boarding. Systems that perform O/D Matrix estimation using solely origin-only data have already been proposed for both rail systems [ZRW 07] as well as public transit buses [Cui 06]. Independently on the the quality of the obtained results with these mechanisms, it is well known that the determination of both entry and exit points leads to more accurate and representative O/D Matrices [Gordillo 06]. This affirmation is understandable because if we automate passengers' entry and exit points capture, we remove the need for human interference, and turn the derivation process solely a matter of electronic manipulation. Costs are reduced, results more accurate, and the frequency on which derivation is obtained attenuated [Cui 06].

O/D matrices estimation is one of the possible uses of our system. By deploying Bluetooth sensing nodes within buses, we lay the foundations on which up-to-date and low-cost O/D matrices may be inferred. This notion of sensing nodes leads us to another interesting subject to our work – sensor networks. The relation that this subject has with our work is what we discuss next.

4.1.4 Sensor Networks

As ubiquitous computing [Weiser 93] sees widespread use, everyday objects gain the potential in becoming artefacts with computational and sensing capabilities. Hence, the notion of a set of interconnected sensors gains wider acceptance. A sensor network may be seen as a set of small nodes which have limited computational power and memory, and a sensing module. Several particularities must be met in order for a set of sensors to be classified as a network. One of the most important is the small computations that each sensor usually performs, so that centralized computation efforts may be relieved [ASSC 02]. Also, it is very common for these nodes to perform cooperative work, therefore augmenting the accuracy and overall sensing capacity achievable by means of individual sensors [HKB 99].

In order for the individual sensors to form a network they must be interconnected. Wireless technologies are seen as ideal for this purpose as they require no direct physical connection, therefore retaining the unobtrusive nature of ubiquitous computing [HKB 99]. Naturally, several technologies can be used to achieve this, as some of them are inappropriate due to their inherent complex stack protocol [LDB 03]. Each node in a sensor network maintains its functionality as if they were independent of all other nodes. This allows, among other things to increase resistance to failure, as the network functionality is unbounded to any specific component. Each node has a respective mission and collects its own contextual data. Naturally many types of data may be gathered by sensors, being one of them individuals/equipment sensing.

Our system cannot be seen as a real sensor network, as for example is defined in [ASSC 02]. Our nodes are in fact more than simple sensing nodes, as they hold the capability of performing high demanding computations. Furthermore, our nodes do not have the need to communicate between them directly. In reality, if we were to enable such facility, our overall system performance would probably be undermined. Also, our nodes energy consumptions are high compared to what is usually seen with small sensing nodes, and therefore don't have the capability of working autonomously for high periods of time. Finally, the environments on which our nodes can be

deployed are restricted to non-extreme settings, as they don't hold the capability of enduring such hazardous environments.

With the previous assertions in mind, there are some characteristics of our system common to sensor networks:

- The system performs sensing through the use of Bluetooth. All stations have this capability, as the information is posteriorly stored centrally.
- The system is fault tolerant. It has the capacity of enduring individual node failure, ensuring overall functionality even when individual nodes fail [SSJ 01, HSA 00].
- The nodes exchange information with each other and execute cooperative labour, although indirectly. This is done using the central infrastructure, which queries individual nodes, and posteriorly delivers relevant information to the remaining nodes, ensuring synchronization and that no redundant information dissemination is done. Curiously, we use this synchronization approach to resolve a common problem of sensor networks – sensor overlap [HKB 99], in which is very common that two nodes overlap their sensing activities, therefore sending replicated data to another node.

As we presented the similarities that bound sensor networks with our work, it is now opportune to refer to another relevant field on our work – *Distributed Systems* (DS).

4.1.5 Distributed Systems

A DS can be thought of as a logical grouping of a set of functionalities when the components that constitute the system are physically separated. In other words, a DS is a set of inter-connected components – by means of a network – that work together in order to achieve specific common goals [CDK 05]. Various subjects such as CORBA [Bolton 01] may be discussed within the field of DS, but to our work we are mainly interested in three specific topics:

(a) The characteristics that define a DS

(b) The notion of a distributed database

(c) The relationship with mobile and ubiquitous computing

For a system to be though of as a DS it must conform to a set of characteristics like autonomous component functionality. The components that constitute the system have only a limited and restricted view of the system as a whole, as they have no specific domain knowledge of other components, and therefore perceive the system in a very restricted way [CDK 05]. Also, the ability to endure component failure is a common particularity of DS. Recalling the definition of sensor networks (which is a particular kind of DS), it was seen that functionality should be unarmed even when individual components, for some reason, cease to function to their full potential.

Also, and relation to point (b), the notion of DS is also extended to data repositories. As the volume of information exponentially grows, it is very common to see repositories that spawn over several physical locations. The simplest architecture used in distributed databases is that of multiple-client/single-server [OV 99]. The server holds the actual *Database Management System* (DBMS) which maintains system wide information. When needed, the clients proceed to remote repository access (which can be direct or indirect) storing and/or retrieving information accordingly. Naturally, this way of data distribution has associated particularities. Namely, replication is achieved [OV 99, CDK 05]. We have the system wide data in the server, but a percentage of that data is replicated locally at the clients. This allows, among other things, to gain additional access speed, since the need to constantly contact the server for database related actions is removed. Further, using distributed databases schemes such as this allows for easy information recovery. In case of local data corruption we may proceed to recovery by querying the database. Finally, concurrent access to the database is fairly straightforward, since only the server entity is preoccupied with such issues.

We also highlight the continuously common introduction of mobile devices in distributed systems [Kleinrock 95]. As these gain the ability to communicate using different wireless means, they are in fact turning themselves part of larger and more complex systems. As always with distributed systems, this equipment has no knowledge of the complexity of the system as it functions autonomously from the remaining components. It is now common to verify small mobile equipment having access to certain services when they are in specific locations. This particular method of information

access is known as location-aware computing and it plays a major role in the specification of our system. We discuss this subject latter in our work.

Our system is perceived as a distributed system. It is composed of several physically dispersed components that work with each other to achieve global goals. Also, the system is not strictly dependent on any of these individual components to maintain functionality, as it can handle failure. In the individual nodes we make use of a DBM [BN 92] engine in order to achieve local replication and persistence. A central component with a RDBMS exists, as it holds information related to all individual components. Finally, and as we disseminate information to mobile equipment, these may be seen as nodes that are part of a more complex and distributed system.

After this overview of distributed systems we proceed to the description of the mechanism that actually makes distributed systems a reality. For this reason we present a discussion about *Inter Process Communication* (IPC) methods in the following section.

4.1.6 Inter Process Communication

A process is the execution of a program and consists of a set of bytes which a CPU interprets [Bach 86]. In typical UNIX and UNIX-like systems, processes are loaded in memory and have several types of information related to them, including the text segment (e.g. machine instructions), initialized data, uninitialized data, the stack and the heap [Bach 86, SR 05]. A process is then an instance of a program which has associated to it a set of resources. As processes are forbidden of accessing each own resources directly, a mechanism for allowing inter-communication is required.

IPC is a mechanism which allows for unrelated processes to inter-exchange information. Many kinds of IPC mechanisms exist, as some of them are indicated for local IPC and others for remote IPC. Some examples of IPC include signals, named and unnamed pipes, shared memory, UNIX domain and Internet sockets, and message passing. IPC mechanisms are not restricted to be used exclusively between them, as they can be combined in order to obtain a higher degree of expressiveness. A typical example is to use both Internet sockets and message passing for achieving remote communication between processes. Also, some types of IPC are inherently designed to handle

asynchronous events (signals), as others are designed to handle synchronous events (named pipes). Still, others have the ability to handle both types of events. The most known example is that of sockets, where modern operating systems and programming languages define an *Application Programming Interface* (API) that supports both blocking (synchronous) and non-blocking (asynchronous) versions of these.

Sockets – more concretely Internet sockets – are one of the building foundations of inter-network communications. An Internet socket is in fact a pair formed by two entities: the IP address of the target host and the port on which a process is listening to. Virtually all modern operating system provide a TCP/IP implementation in their kernel which is composed by three distinct layers [WS 95]: the socket layer, the protocol layer and the interface layer. As the socket layer – and consequently sockets – is situated the nearest of the application layer, it provides for an abstraction between applications and underlying transportation layers such as TCP and UDP. Processes therefore use specific function low-level invocations (e.g. system calls) so that IPC using sockets can happen in a transparent and compatible way between processes, that can be separated by different networks, different underlying protocols, different operating systems and even different applications.

In sum, IPC mechanisms are essential to our system because of the following reasons:

(a) Our system is a distributed system composed by three separate entities: nodes situated at public transit infrastructures; a broker responsible for name lookup and connection establishment; a centralized infrastructure

(b) All entities described above need, at some point, to execute several operations simultaneous. By using a multi-process paradigm we can fulfil the system's needs.

As demonstrated, IPC represents an important aspect of our system. We use several of the techniques described in this chapter, as IPC is executed both locally and remotely. A working “protocol” for communication is defined, so that different components can request for concrete actions from other elements.

The discussion of IPC mechanisms marks the end of our broad view of related work. Throughout this section several issues were discussed, as they where used to lay the foundations on which the ensuing chapter and included subjects are built upon.

4.2 In-Depth View

In this section a more meticulous review and discussion is performed. The subjects discussed within are less far-reaching than those seen in the previous section, but the contribution they have in defining and contextualizing our work is greater. For this reason, a more through scrutinization is made. In concrete two subjects are discussed: Bluetooth wireless technology and Context-Aware computing.

4.2.1 Bluetooth Wireless Technology

Bluetooth is a radio wireless technology whose development started in the mid 1990s. It has over the years gained increasing acceptance, as its presence is noticed in virtually all recent electronic equipment, such as PDAs, mobile phones and personal computers. The world-wide adhesiveness of Bluetooth technology is demonstrated by the estimated 520 million devices that supported it in 2006[1].

Several particular characteristics differentiate Bluetooth from other wireless technologies. Its low cost and low power consumption justify why there is such a widespread integration on electronic equipment. Furthermore, and comparing for example with *Infra Reds* (IR), which are also commonly present in mobile equipment, Bluetooth works by transmitting and receiving the radio signal in a omnidirectional manner, therefore dismissing the need for having line of sight or antenna directionality.

More technically, Bluetooth is composed by a set of interconnected layers. This layer oriented specification is much common in networking communication stacks, as the most known examples are the TCP/IP and OSI models [SFR 03]. The purpose for dividing communication stacks into several layers is to allow for heterogeneous host systems to communicate using solely

[1]http://www.economist.com/sciencetechnology/tq/displayStory.cfm?story_id=7001843

common logical agreements – the protocols [Stevens 94]. Figure 8 shows the disposition of the layers and the belonging protocols [Bisdikan 01].

4.2.1.1 Bluetooth Lower-Levels

The Bluetooth radio operates at the 2.4GHz unlicensed Industry Scientific and Medical (ISM) band. As this is a free band, many components potentially operate at the same frequencies. In fact, the IEEE 802.11x family of wireless technologies uses this specific band. This leads to situations of interference when both technologies are in the same operating range. Manufacturers are aware of this, as the devices which operate at this band must be able to share frequencies and tolerate interference. By using a *Frequency Hopping Spread Spectrum* (FHSS) technique, Bluetooth manages to reduce interference, as well improve security measures of the technology [Wang 01]. The radio therefore employs a pseudo-random algorithm, executing 1600 hops/sec over a bandwidth of 79 one-megahertz channels. The frequencies on which Bluetooth operates are therefore specified by the following expression:

$$f = (2{,}402 + k)\,MHz, 0 \leq k \leq 78$$

Even though FHSS and error coding techniques are used for minimizing interference, Bluetooth inter-operability with WLAN (e.g. IEEE 802.11x) leads to higher packet lost probability, and consequently to undermined performance for both of these technologies [GCR 03].

Bluetooth has three distinct classes: class 1, 2, and 3. Class 1 devices are the most powerful, and consequently transceivers of this type can communicate within 100 meters of each other. Class 2 transceivers are the most commonly used, due to their balanced power consumption and range. These devices are present in virtually all commercial mobile equipment, and can establish connections up to 10 meters from each other. The less powerful devices are those which belong to class 3. Their power consumption is minimum, and for that reason wireless communication can only be established if devices are up to 1 meter from each other. Figure 7 demonstrates the differences that exist between range radius of Bluetooth classes.

A word on the fact that if we use two dongles, one class 1 and other class 2, the maximum range will not be of 100 meters. This is due the fact that

range is limited by the class 2 transceiver lower power output. On the other hand, using such configuration allow us to extend the class 2 transceiver beyond the 10 meters limit, but nowhere near 100 meters. Nevertheless, it is possible to use specialized hardware such as high gain antennas, so that long range Bluetooth communication is possible.

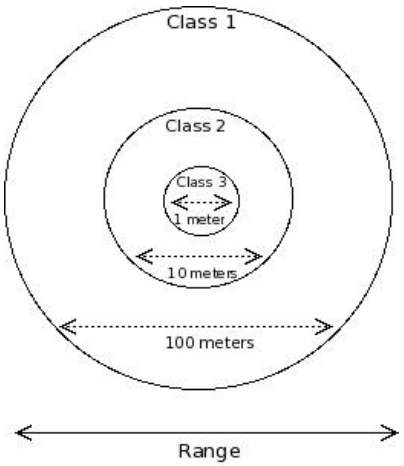

Figure 7: *Bluetooth classes range*

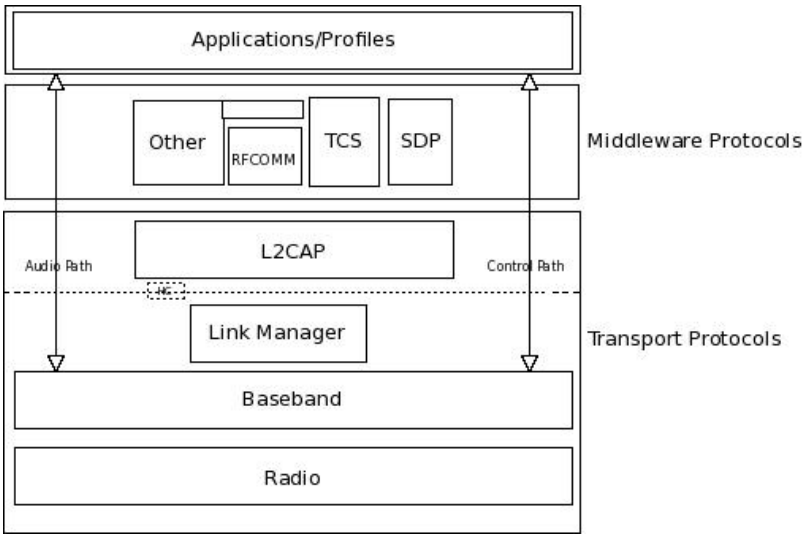

Figure 8: *Bluetooth Protocol Specification*

Moving to the baseband protocol, it is situated just above the actual physical radio transmission medium. Baseband defines how Bluetooth links are established, how Bluetooth networks are created, how transmission medium is

shared, and also define the structure of the low-level packets [Bisdikan 01]. Each Bluetooth device has associated with it a unique 48-bit address (BD_ADDR), which can be though off as the equivalent of a Ethernet MAC address. Individual identification becomes possible due to this feature, as no two addresses are equal. Associated to each Bluetooth device is also a 28-bit clock value. For communication to be initiated between two Bluetooth devices, interventionists must exchange both the BD_ADDR and clock values. A network of Bluetooth devices is formally defined as a *piconet*, and in each piconet there must be only one *master* and up to 7 *slaves*. Optionally, there is the ability to inter-connect piconets, therefore forming a *scatternet*. No direct communication between slaves within a piconet can exist, as the master unit acts as a routing element. Also, for communication to be effective and full-duplex supported (e.g. transmission and receiving happening simultaneously), a Time-Division Duplex (TDD) technique is used for sharing the bandwidth available. In practice this means that each device is allowed only to transmit for 625 μs at a time, in each channel (or frequency), although a packet can be separated along 1, 3 or 5 distinct channels [Wang 01]. It is the master's unit clock which identifies the frequency or channel on which transmission is going to happen. For this reason, a Bluetooth device can only be master on one piconet. Also, master units will transmit only on even-numbered slots (e.g. 625 μs time units), and slaves will transmit only on odd-numbered slots.

For a piconet to be created some steps must be performed. If the master already knows the address of the slave, then the *inquiry* phase may be skipped. This phase consists of locating devices and obtaining their address and clock values. It works only if the inquired device is in *discoverable* (or *inquiry scan*) mode, and the inquiring device transmits inquiry messages. The discoverable device will then respond by sending inquiry reply messages, which, among other things, contain the address and clock values. Naturally, the devices which are set to discoverable mode need to continuously check for inquiry requests, which leads to increased power consumption, even when Bluetooth isn't actually being used by the user.

This is due the fact that Bluetooth discoverable devices remain in stand-by mode, during which they listen to inquiry messages every 1.28 seconds, using one of the 32 pre-established inquiry frequencies. As such, inquiring devices use periods multiples of 1.28 seconds to find Bluetooth devices in the

vicinity. The Bluetooth specification states that in an error-free environment, an inquiring device must spend 10.24 seconds (i.e. 8 * 1.28 seconds) in order to discoverable all surrounding Bluetooth enabled devices.

After the details regarding specific devices have been determined, the master can then proceed to invite elements to join the piconet. This process is known as the *paging* phase, and works similarly to the inquiry phase. First, a device must be set to *connectable* (or *page scan*) mode. Then, the device that wants to initiate the communication – the paging device – will transmit a paging message to the connectable device. This later device will then reply with a *page response* message, sending its details over the communication channel. After this, the piconet can be created. A device that is set to connectable mode also needs to perform periodical scans, and therefore also contributes to increased power consumption, although less than those devices in discoverable mode.

Devices actually communicate through the exchange of baseband packets. Each packet has a 72-bit *Access Code* (AC) field, a 54-bits header and a 2746-bit payload. Although this is the base structure of baseband packets, several packet configurations can be made. The only field that is required to be present at all times is the AC, which is used exclusively when we have a baseband packet of type ID [Bisdikan 01]. In order for packets to be exchanged a link must be established. Two kinds of links can be established when using Bluetooth. *Asynchronous Connectionless* (ACL) links are the most used type, and provide no Quality of Service (QoS) mechanisms. On the other hand, this Best-Effort link does provide for integrity using retransmissions, and uses error correcting techniques. Also, there is the possibility of establishing *Synchronous Connection-Oriented* (SCO) links. Although SCO does not allow for retransmission, its link symmetry and constant rates, make it ideal for voice communication between Bluetooth devices.

Moving up the protocol hierarchy, the *Link Manager Protocol* (LMP) is the component whose actually responsible for controlling the Bluetooth link. LMP takes care of security related issues, which includes authentication and encryption. There is a large specification of LMP *protocol data units* (PDU), as some of them are related to security issues, and others to information exchange, crucial for connection establishment. Due to the nature of Bluetooth, asymmetric authentication methods do not apply, and therefore a

simple challenge/response mechanisms is used for authentication [Bisdikan 01, GPS 04]. The first time two devices met, it is necessary to use a *Personal Identification Number* (PIN) for initializing the authentication process. After PIN data has been correctly inserted, the authentication process will from then forward dispense the use of a PIN, as it will utilize a 128-bit authentication key. The authentication process can in reality be dismissed all together, as some application profiles don't need to perform authentication for exchanging information. Furthermore, upon link establishment encryption can be set using the previously known link key used for authentication. The LMP also allows for usage of low power modes. These are used to reduce ower consumption, but also to allow for multiple operations to happen simultaneously. This is why although the limit for a piconet is of 7 slaves + 1 master, by using power modes we can get up to 256 slaves (in *parked* mode) in a piconet. Also, it is in this layer that *friendly name* requests are made, as each Bluetooth device may have a name associated with it. It is very common for mobile equipment to also discover friendly names upon inquiry requests, but this delays the procedure, as it is necessary to establish a connection between devices [AK 05].

The *Logical Link Control and Adaptation Protocol* (L2CAP) is the layer most closely situated to the host system (an can in fact be implemented at the host). L2CAP takes care of care of datagram segmentation and reassembly, multiplexing of service streams, and QoS issues [GPS 04]. The main purpose of the L2CAP is to work as a translating unit between the higher-level Bluetooth protocols that run in the host, and the lower-level protocols that run in the Bluetooth device.

4.2.1.2 Bluetooth Middle-Levels

The *Host Controller Interface* (HCI) is defined as a mere interface so that hosts can access the lower-level protocols of Bluetooth modules. Since there is the possibility that lower and higher-level Bluetooth protocols are separated, there is a need for establishing an interface to control Bluetooth modules. This is usually the case when an external Bluetooth module is connected to a PC through an USB port. On the other hand, if the Bluetooth module is integrated into a larger device and the same microprocessor controls them both, then the

HCI firmware implementation can be neglected [GPS 04]. A common example of this is a Bluetooth headset.

The protocols included in the middleware layer are usually implemented at the host level. They work to alleviate the higher level protocols from Bluetooth concrete specifications, abstracting the communication process. An important protocol that lies within this layer is the *Service Discovery Protocol* (SDP). Its function is to determine the hosts' available services, and to collect information on how to use them. A Bluetooth service is defined by *Universally Unique Identifiers* (UUIDs), which are used to describe services' names and attributes. The RFCOMM protocol is also widely used in Bluetooth communications. It is an emulation protocol that creates a virtual RS-232 (e.g. serial) link between devices. RFCOMM enables for several types of applications to work using Bluetooth. Examples include object exchange (OBEX) between devices, point-to-point connection establishment (PPP), and telephony control signalling (AT) commands.

4.2.1.3 Bluetooth Profiles

At the top of the Bluetooth stack reside the specifications that allow for construction of compatible applications. These specifications are called profiles, and they define the conditions which must be followed for inter-communication between devices to happen. All profiles are extensions of the *Generic Access Profile* (GAP), as new profiles can be devised upon existing ones. This allows for creation of profile hierarchies, as this the case of the Object Push Profile, which is based upon the Generic Object Exchange Profile, which in turn is based upon the Serial Port Profile [Bisdikan 01, GPS 04].

4.2.1.4 WAP over Bluetooth

Bluetooth versatility is further demonstrated by its capability of supporting additional networking communication stacks on top of the Bluetooth stack. It is possible, for example, to use the *Wireless Application Protocol* (WAP) suite over Bluetooth. WAP is an industry standard technology used for presenting content in hand-held mobile equipment. WAP actually consists of several protocols, and therefore forms a networking communication stack. Given that many hand-held devices posses a WAP browser software

component, users can make use of it to access a set of services by means of a Bluetooth link. Figure 9 [AL 00] demonstrates the involved layers in order for this to be accomplished. Additional information on WAP over Bluetooth can be seen in [AL 00].

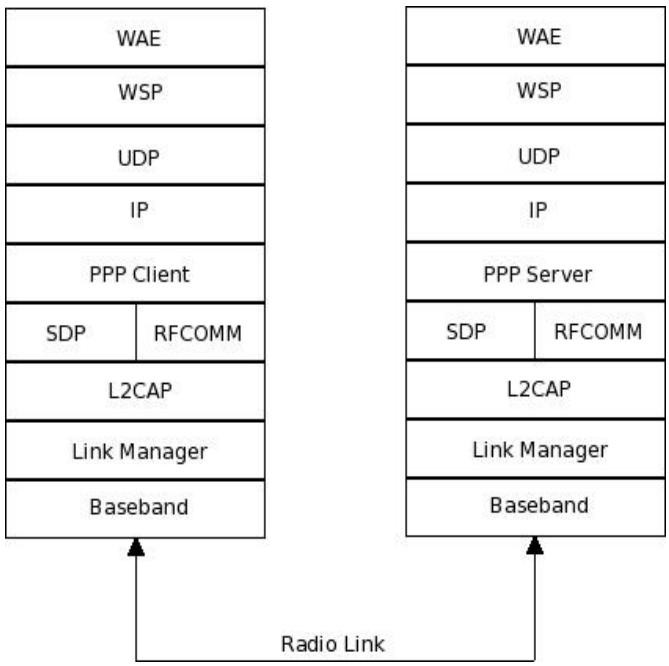

Figure 9: *Client/Server communication using WAP over Bluetooth*

With this section we gave a detailed overview of the Bluetooth Wireless Technology. We described the stack that enables for Bluetooth inter-communication, and verified that it is common that higher-level layers to be implemented outside the actual Bluetooth device. Bluetooth profiles were also discussed, and the versatility of the technology was mentioned, as we saw that is possible to combine external communication stacks with Bluetooth.

Bluetooth has great relevance on our work. First, our system uses the Bluetooth inquiry capabilities to continuously scan for nearby discoverable devices. Secondly, we utilize Bluetooth SDP and profiles to perform content delivery. As devices are discovered, the system determines if a pre-defined service exists which has that address as destination. If it does, the system tries and push – using OBEXPUSH – the content to the user. As usually we are interest in nearby devices, we limit our scanning capabilities with the use of a

class 2 dongle. On the other hand, as we are interested in maximizing delivery capability, we use a class 1 dongle for content delivery.

This notion of using a wireless technology (Bluetooth) to delivery content at specific locations and under specific conditions, leads us to another crucial notion in our work – Context-Aware Computing. What Context-Aware Computing is and how does it influence our work is what we describe in the ensuing section.

4.2.2 Context-Aware Computing

Multiple aspects must be taken into consideration when defining an element's context. Information like location, time, and weather are all valid factors that contribute to the delineation of the context on which an element is inserted into. Context-Aware computing is a field on which contextual information is the central element on defining the behaviour of hardware and software systems.

4.2.2.1 Location-Aware Computing

Historically, Context-Aware computing is an extension of the notion of Location-Aware computing. Aalto *et al.* [AGKO 04] define a Location-Aware service, as a service whose behaviour is mostly determined by location information. In their work, they developed a push-only dissemination system, entitled *B-MAD*, which uses Bluetooth and WAP to perform content delivery. Users must pre-register their Bluetooth address and phone number for receiving content. Thereafter when a user is in the vicinity of a Bluetooth station, this element will flag a centralized ad server using a WAP connection. Consequently, the ad server will send the Bluetooth station information so that content is delivered to the end user using a Short Message Service (SMS) message. The main problems with this approach are the need for disclosing phone numbers, and the constant communication with the centralized server leading to additional latency.

The Place Lab initiative [SLBGMLBHI 03] uses a component installed in end-user devices, known as *Place Bar*, to direct users to relevant web content, taking as a parameter the Wi-Fi Access Points location information. The goal is to create a positioning system, termed as a *Global Wi-Fi Positioning System*,

analogous to GPS. To achieve this, the unique *Basic Service Set Identifier* (BSSID) – which corresponds to an Ethernet MAC address – is used to create a database that associates Wi-Fi access points to a certain location. The Place Bar would then query the database for determining the current location, and thereafter direct the user accordingly. Although this is an interesting concept, the current proliferation of Wi-Fi spots makes the maintenance of this database a very difficult job. Also, users would need to download specific software for this functionality to be enabled in their devices. Other interesting developments in Location-Aware computing include the creation of the *Vehicular Information Transfer Protocol* (VITP), which is used for creating services over V*ehicular Ad-hoc Networks* (VANETs) [DFNI 07]. The purpose would be to facilitate decision, by providing traffic related information such as traffic congestion points, therefore working as an enhancer of already common employed technologies such as on-board GPS. Harter *et al.* [HHSWW 99] developed a system which provided 3-D modelling and localization of users inside a building. The usage of ultrasonic transceivers, referred to as *Bats*, provided the information necessary to successfully locate users inside a closed space, a task usually made difficult due high interference levels which make radio-based and electromagnetic techniques non-optimal. The interesting point here is to note that these previously described applications are mostly driven solely by one contextual aspect – location.

4.2.2.2 Context-Aware Multidimensionality

Context-Aware computing introduced new dimensions on which applications behaviour could be specified. Although location still plays an important role in defining context, other important factors such as temporal and physical aspects play an active role in specifying context. The own notion of context in computing is far from being commonly agreed upon. In our opinion, the best notions which define Context-Aware are those mentioned by Schilit *et al.* [SAW 94] and Chen & Kotz [CK 00], where the authors define context in four distinct categories:

- Computing context – such as networking and connectivity
- User context – such as location and profile specification
- Physical context – such as weather and pollution levels

- Time context – such as time, day of week and day of year

This multidimensional aspect of context is again mentioned by Schmidt *et al.* [SBG 99], where additional dimensions are considered to influence contextual specification. Further, Dey & Abowd [DA 99] categorize contextual factors as being primary and secondary. They argue that factors as location, time, activity, and identity are to be considered primary aspects, as they convey the power for inferring additional types of information. The authors also provide a taxonomy of Context-Aware features, in which they define as being part of one of three categories:

- Presentation of information and services to a user
- Automatic execution of a service
- Tagging of context to information for later retrieval

In sum, it seems clear that defining context is not straightforward, as many aspects come into play due the dynamic nature of the environment.

4.2.2.3 Context-Aware Applications

Throughout the last decade and a half many research efforts have directed their efforts towards Context-Aware computing. The field of Context-Aware per see spawned mainly due to the work developed at Olivetti Research. The *Active Badge* prototype system was developed in 1992, and it consisted of using a IR emitting badge for locating staff within a building [WHFG 92].The novel contribution of this project was not due the localization technique used, but to the fact that the badges would respond to environmental stimulus. Concretely, a light sensor was included in the badges. The absence of light would make the badge redefine its behaviour, reducing IR signal emission frequency, and consequently saving power. Also, the active badge location information was used so that a receptionist would perform call forwarding to the staff's nearest available phone. Another work developed by Olivetti Research, and with direct relation with the Active Badge, was the *Teleporting* system [BRH 94]. This applications worked by allowing users to access X-server sessions by using computational resources most nearly available to them. The Active Badge system would be used to perform user localization, and allow the application to “follow” the user around. Brown *et al.* also defined a system which used the Active Badge notion. The purpose would

be to route a message to a visitor which had an Active Badge, even if the user didn't posses a pager. In that case, the message would be routed to the closest person of the required destination.

The *Cricket* system is defined as *location-support* infrastructure [PCB 00]. It works in a decentralized manner, using *beacons* to transmit signals to *listeners*, that the latter can use to determine position within a building. It works by using a technological combination of Radio-Frequency (RF) and ultrasonic signals. The beacons disseminate these signals periodically, and the listeners use this information along with an inference engine to determine current position. Contextual location related information (e.g. position and distance) are used to determine the possible position of users.

Rukzio *et al.* [RSH 04] devised a system that uses physical posters for content dissemination purposes. Encoded images are embedded into the posters, as users utilize their mobile devices cameras to capture images, and consequently send them to a server located in the vicinity of the posters. This is accomplished through a Bluetooth link. The server will, upon image reception, use image recognition software to determine the correct content to send to the user. For the system to work, the user most download a Java based application, termed as a *Simplicity Personal Assistant* (SPA).

LeBrun & Chuah [LC 06] deployed what they termed as *Content Distribution Stations* at public transit vehicles. These stations are content dissemination units that deliver information to public transit passengers. As users transit between several points of the transit network, much idle time exists. The goal would be to take advantage of that idle time, and utilize an offer an array of services, raging from news to multimedia download. The content is local to the stations, as no direct networking connection exists with the outside world. The stations would therefore need to synchronize periodically with servers located at, for example, major stations. This kind of concept introduces the notion of *Delay Tolerant Networking* (DTN), allowing for information routing and dissemination even when a direct connection to a network does not exist. Although the authors state that both Wi-Fi and Bluetooth technologies can be used to access the content, the way this is actually done is unclear.

A content dissemination architecture that uses mobile devices as relay points is the issue of foci in [LLSFC 06]. The system is defined by three distinct components: fixed source nodes, mobile relay nodes, and data sinks. The goal is to use the mobile relay nodes (and also data sinks) to enhance the information diffusion rate. Data is transmitted using Bluetooth from source nodes to mobile and data sinks nodes, as these hold the capability of acting as a forwarding element, disseminating the content to other potentially interested nodes. User interests and location define the context in which content is to be delivered. This is also a form of DTN, as information is routed to destination nodes without previous pre-established structure. Interesting results were obtained with almost 90% of delivery success. On the other hand, there is the need for mobile nodes users to agree to act as forwarding elements.

Ravi *et al*. [RSDI 05] use Bluetooth SDP facilities to create a protocol entitled SDIPP. The goal is to add personalization capabilities so that innovative services are implemented. Users would discover services located at specific location using Bluetooth discovery capabilities. Thereafter, there would be a need to initiate a GPRS connection with a centralized infrastructure, retrieving data for enabling usage of the service. Consequently the user would interact with the application using a Bluetooth connection. The authors demonstrate an interesting application of this system, where doors within a building are opened according to users' devices. Still the grand limitation is related with the necessity of using a GPRS connection by the user, which has inherent costs associated to it.

Opportunity Knocks is a system aimed at providing cognitive assistance to mental impaired individuals [PLGCLOWFK 04]. The system is directed at those individuals that use public transportations, but due their declined cognitive abilities experience problems when travelling autonomously. Opportunity Knocks works by using a GPS module, a GPRS-enabled mobile phone, and a central inference software. The GPS and mobile phones are naturally carried by the user, and communicate through Bluetooth. The central inference engine uses location information and past history to determine, for example, the probability that individuals have board the wrong bus or missed their exit stop. By using this conjunction of location and temporal contextual

information, along with a sophisticated inference engine, Opportunity Knocks demonstrates the usefulness of context-aware computing.

By using location information and users' profiles specifications, the *Social Serendipity* system exploits mobile devices capabilities for facilitating social interactions [EP 04]. The authors have developed an application, named *BlueAware*, which uses Bluetooth inquiry capabilities to scan for devices in the vicinity. The purpose is to augment the notion of social software, which ironically usually require users to be in direct contact with computers for interacting with other people. The Serendipity system consists of two distinct applications. The first is a Bluetooth scanner termed as *BlueAware*. This scanner, that runs in the user's mobile device, is continuously scanning for nearby Bluetooth devices, and also maintains a *proximity log*. BlueAware collects the data that will be used by other applications, such as the Serendipity application. It also runs in the end-user mobile device, but must connect to a central service by using a GPRS connection. The objective is to search for matching profiles that may be in the vicinity, notifying accordingly the interested parts.

Very closely related to our work is the *BlueMall* application [SCCM 08]. This Java-based system is defined as a context-aware ubiquitous Bluetooth advertising system, developed to be used in commercial areas. It works by placing several *Access Points* (AP) throughout the advertising area. When a device comes into range of an AP – which is constantly running a scanning program – the AP element will contact a central server to determine if any content is to be pushed to the user. The system does allow for time-related specification of content delivery, and also allows defining a list of addresses to ignore. BlueMall has the ability to recall which devices were already served, and therefore does not continuously tries to contact users. Nevertheless, the constant need for contacting the central server, and the use of Ethernet to inter-connect components seems to limit the scope of BlueMall to indoor environments.

Ubiqmuseum is another example a context-aware advertisement system [CMT 06]. It was developed to augment museum visitors experiences. The system consists of three different components: *Museum Information Points* (MIPs), clients, and a central server. Museum visitors will need to have a Bluetooth enabled equipment along with a pre-downloaded Java application.

This Java application (which is the client) will give the opportunity for users to define their personal preferences, letting them choose options such as language, type of device, and level of information detail. When the user wants to retrieve information from the MIPs, it will have to execute an inquiry and consequently connect to the MIP component. After this is accomplished, the MIP will contact the central server, sending it a *code operation* (codop) which the server will interpret and take the appropriate action. Finally, the MIP will receive the information from the server, and push it to the client application. Likewise the BlueMall application, Ubiqmuseum restricts itself to a very concrete field, which in this case is museums settings.

OpenProximity[2] is an example of a *proximity marketing* system. It is composed by several components and allows for remote web management. The system supports the use of a dongle for scanning and another for uploading content, which can maintain up to 7 simultaneous connections. Administrators of the system will have to create *campaigns* in order to disseminate content to users. Associated with these campaigns is a set of rules that can be configured to indicate the method of delivery (OBEXPUSH, OBEXFTP, etc.), time settings, and address filtering. Although it is stated that OpenProximity can be configured to have a central component controlling individual stations, it is not clear how this works. Other systems similar to OpenProximity exist, such as *BlueMagnet*[3] and *Fexmax*[4]*.* The method of functionality is similar to OpenProximity, where campaigns with associated rules are created in order to schedule content delivery to users.

A combination of hardware and software components, the *Bluegiga Access Servers* are also commonly used for proximity marketing, but their flexibility allows them to be used with broader objectives in mind, like context-aware applications. Depending on the version, these systems can be equipped with 3 Bluetooth radios, supporting up to 18 simultaneous connections. Although Bluegiga access servers work mainly by pushing content to users, they can be configured to receive input requests through a pull channel, therefore implementing a integrated dissemination model (recall **Section 4.1.1**). Additionally, many filtering options can be configured for content

[2]http://sites.google.com/a/aircable.net/aircable/Home/openproximity-2-0

[3]http://www.bluemagnet.com/

[4]http://www.fexmax.com/

dissemination, including time, type of equipment, brand of equipment, and distance. Our work relates very closely to the Bluegiga Acccess Servers, in which it allows for creation of a set of services to disseminate content to users according to pre-established rules. Nevertheless, our work deals solely with the software part, and for that reason is not restricted to any particular type of equipment.

Throughout this section we described in detail the notion of context-aware computing. We verified that context-aware is indeed much more than just location, as it can include many other aspects, such as personal preferences, time restrictions, and even physical particularities. Moreover, we viewed many examples of context-aware applications, and how these work in real-world environments. We saw that these types of applications can be used in many distinct areas, including assistance of impaired individuals, social networking, proximity marketing, and overall content distribution. With the presentation of both Bluetooth technology and context-aware computing we contextualized our work and now proceed to the technical description of the system.

5 System Description

The purpose of this section is to technically describe our system using a top-down approach. First, we present some example scenarios to help visualize system's potential. Secondly, we discuss system's main requirements so that a notion of functionality is obtained. Thirdly, we demonstrate a highly abstract view of the system, as we present the overall system architecture. Thereafter, we focus our discussion on the presentation of individual system components, and finalize this section with the demonstration of concrete particularities of the system, including configuration aspects, technical details, and algorithms.

5.1 Example Scenarios

The system may create a set of context-aware services to disseminate content to users. Several types of contextual impositions are possible, including time and date of execution, location of device, and present nearby devices.

For example, it is possible to create a service that disseminates *static* (i.e. that doesn't change over time) content to John. Imagining that John's birthday is on the 1st of July, a service can be created that indicates the system to try and deliver content at “Bus stop 1” when he sees John. Optionally, the service may be set to run all day, before expiring after midnight.

Another example is to create a simple textual content which the system broadcasts constantly to all Bluetooth devices in the vicinity. We could set this service to run for ever, but remembering the devices to whom content had been successfully delivered, so no duplicated dissemination happens.

Still, a more interesting example is to create a service which delivers a personalized image content to a set of of users. The system is set so that delivery is triggered only when John, Mary, and Marc appear, together, at bus stop “Bus stop 10”. Furthermore, only when these three devices are seen continuously for more than 5 minutes, will the system disseminate the content to all of them.

It is also possible to fetch *dynamic* content, i.e. content that changes over time. For example, if we have a web page that periodically refreshes its content output, we can create a service that points to that web page, and define it to delivery content to Suzie every two days, between 12:00 and 14:00 at either “Bus stop1” or “Bus stop 2”.

In sum, our infrastructure allows us to specify a rich set of features that define the restrictions on which a service is to be executed. In order to better understand how this works, we proceed to the definition of the overall system requirements of the system.

5.2 Overall System Requirements

Since the beginning of our work we knew that the system's requirements would be ill-defined, and consequently subject to modifications along the software development cycle. As such, it was clear that the best alternative would be to adopt an agile software development methodology if we were to cope efficiently with the more than probable changes. The chosen method was the *Agile Unified Process*[5] (AUP), a simplified version of the popular Rational Unified Process (RUP). AUP defines four phases and seven disciplines. Phases work in a serial manner, in which disciplines have different weights. Disciplines on the other hand work in a iterative manner and define the concrete activities that team members are suppose to perform. AUP also uses several common used agile techniques such as test-driven development, just-barely-enough modelling, and stakeholders communication encouragement.

Initially, the system's functional and non-functional requirements were unclear. To overcome this situation, research was done and the questionnaire present at **Section 3** was issued. The result was the following initial functional requirements:

1. The system shall, at least, be composed by two distinct separated elements: *Blue Stations*, which may be deployed at several types of environments, and a *Central Infrastructure*, which is deployed at a well known location.

[5]http://www.ambysoft.com/unifiedprocess/agileUP.html

2. The Central Infrastructure shall be responsible for storing stations' contents, be responsible for synchronization, maintain a central database, and allow for centralized control.
3. The Blue Stations must be able to perceive Bluetooth discoverable devices in the vicinity and collect their information. This component will be termed a *scanner*.
4. The scanner shall be configurable, as we may choose to change the time that a scanner searches for nearby devices (e.g. inquiry time), as well the time that a scanner sleeps after each scan (e.g. scanning frequency). Also, we may choose to output devices' friendly names, *Class of Device* (CoD), and the time stamp of the discovery.
5. The Blue Stations shall offer the possibility to deliver both static and dynamic content to users.
6. Content will be offered in the form of services, in which each service is fully configurable. Options like date of delivery, time of delivery, and destination are to be specified upon service creation.
7. Content will be delivered using a push model of information dissemination.
8. Several destinations can be specified in a service, being also possible to specify a broadcast address.
9. Services creation will be done through some kind of interface, that can be either a graphical interface, or a console application.
10. Blue Stations shall support the use of multiple Bluetooth dongles. If just one dongle is used, then both scanning and deliver procedures are executed by the same dongle. If more than one dongle is used, then the scanning procedure will be made by one dongle, and the delivery executed by the remaining ones.

Additionally, the following non-functional requirements were specified:

1. The system shall function uninterruptedly. It must support errors, individual component failure, and implement recovery mechanisms.

2. The system must be designed with flexibility in mind. Future functionality addition shall not influence overall architectural structure.

With this basic requirements in mind, our next step was to designed an overall architecture, which abstractly described our system.

5.3 Overal System Architecture

As we move the description of our architecture components, it is pertinent to define a list of keywords so understanding of the system functionality is made simpler:

- **Blue Station** – The components that are installed at various locations, and are controlled remotely. The stations hold the capability for performing scanning and content dissemination.
- **Central Infrastructure** – The set of components present at a central location. Its purpose is to provide for overall management of stations.
- **Broker** – An intermediary component that provides naming lookup capabilities, and is involved in all connections to and from stations.
- **Service** – The logical component which holds information about the content to deliver, to whom content is to be delivered, and under which contextual setting.
- **Command** – These components are created from service objects, and can be seen as a stripped down version of services. Commands hold only content, method of execution, and destination related information. They are posteriorly interpreted by handlers, that do the actual delivery to users.
- **Flags** – A set of configurable restrictions that can be set upon service creation. Its purpose is to define the context under which a service is to be executed.
- **Service Scheduler** – The component that holds a queue of services, and determines which of the individual components may be executed, and under which order. When a service is triggered for execution, the scheduler translates that object into a command, and passes it along the delivery system.

- **Delivery System** – The component responsible for the execution of commands. This component is the actual responsible for content delivery to users, as it holds the classes that implement the logic for interpreting commands – the command handlers.

As keywords are defined, we proceed to the actual description of the high-level architecture. Looking at Figure 10 we have the main components of the system. We observe the existence of four distinct components: Blue Stations, the Broker, the Central Infrastructure, and User Devices.

We verify that the broker connects both stations and the central infrastructure, by setting between them. Although not restricted too, connection between stations and the broker are usually done with the use of a GPRS/3G connection. Connection type between central infrastructure and broker usually is done through a wired connection. Also, end-users communicate with our system using Bluetooth.

Given the demonstration of the high-level system architecture, we now proceed to the discussion of individual system components.

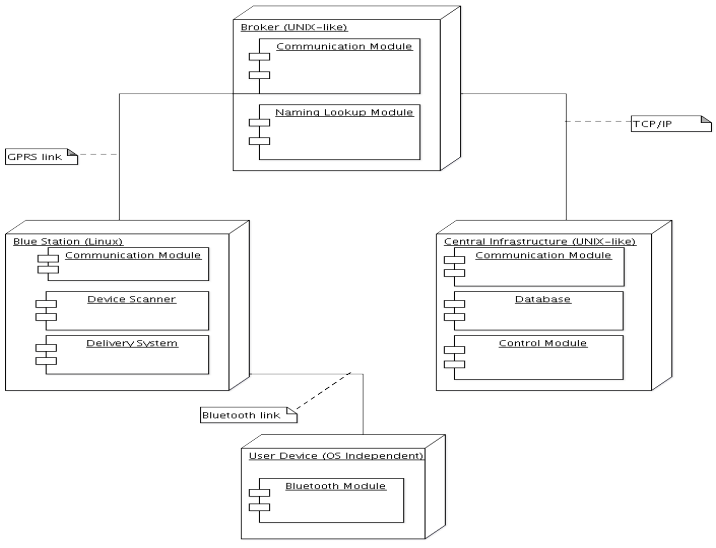

Figure 10: *Updated high-level architecture*

5.4 Component Description

Besides the stations, the central infrastructure, and the broker, the system is composed by an additional component – the administration console. Being part of a distributed system, these components have the ability to work separately from each other, as long as a network connection exists that inter-links them. The most important elements are the stations, which actually scan the vicinity for Bluetooth enabled devices, and further determine if content is to be disseminated. Let's then proceed to an individual description of these components.

5.4.1 Blue Stations

Blue stations are the mobile elements of the system, as they may be deployed at various environments. They are usually small computers with two Bluetooth dongles and a 3G modem attached, as seen in Figure 11. Nevertheless, stations may use several Bluetooth dongle configurations, as well several types of Internet connections (Figure 12).

The basic element of the station is the scanner, a Bluetooth dongle that permanently scans for nearby devices. The scanner normally is implemented using a class 2 Bluetooth dongle. The reason for using a class 2 dongle relates to scan range limiting, as we are interest in devices which are closer to the stations. Nevertheless, if interference is high, the use of a class 1 dongle may be justifiable.

On the other hand, we want to maximize our delivery range. For this reason we use class 1 devices as delivery dongles. Theoretically, many delivery dongles may exist, but in reality constraints such as USB power consumption play an important role in determining the maximum allowed devices running in simultaneous. Further, the more devices exist near each other, the greater the interference will be, and consequently signal quality and delivery success rates may be influenced. We raise attention to the fact that delivery dongles are optional, as blue stations have the capability to work with a single dongle. This situation is undesirable, since using a dongle for both scan and content delivery leads to poor results.

The WAN modem is used so that a connection with the Internet is established. Actually, any technology is possible but due the mobile nature of the stations wireless technologies are more suitable.

The Blue Station in itself is a computer which runs a GNU/Linux operating system. Our current configuration uses a Slackware[6] distribution along with the BlueZ[7] library and tools. Additionally we also use the OpenOBEX[8] package for sending content to end-users using the OBEX protocol. The language on which the major components are implemented is Perl, but several external utilities are used, and some components are written in Bash.

Regarding the software elements of the Blue Stations (Figure 13), six main components exist: the device reader, the publishing module, the service scheduler, the delivery system, the logging system, and the communication system. Some elements communicate bidirectionally and others unidirectionally.

Figure 11: *Asus EEE PC with two Bluetooth dongles and a 3G modem*

[6]http://www.slackware.org

[7]http://www.bluez.org

[8]http://dev.zuckschwerdt.org/openobex

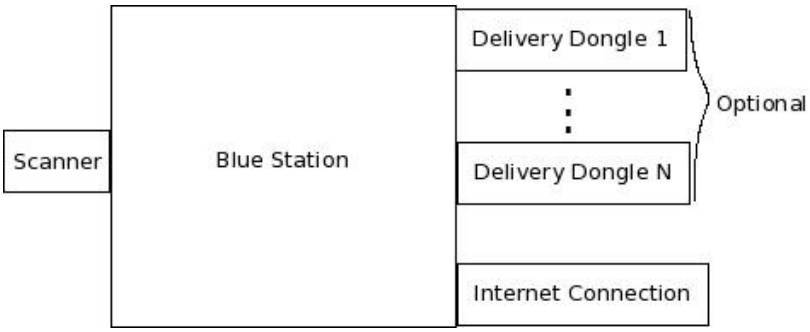

Figure 12: *Blue Station physical elements*

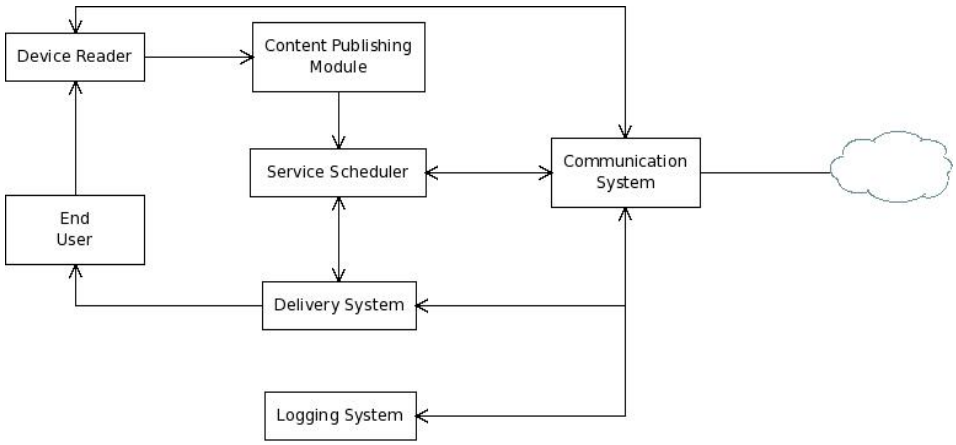

Figure 13: *Blue Station software elements*

5.4.1.1 Device Reader

The device reader, written in both Perl and Bash, is the element responsible for scanning the vicinity, and is composed by several different components (Figure 14). The scanner is encapsulated in a separate application named *btreader.sh*. This tool is entirely written in Bash, and is simply a configurable interface to an already existing BlueZ component named *hcitool.* By default the scanner outputs devices' Bluetooth addresses, the time stamp, and the clock and class hexadecimal values.

The scanner supports modification of various settings, including inquiry duration, scanning frequency, device friendly name retrieval, and the interface to use in the inquiry process. If only one Bluetooth dongle is available, then the scanner can be set to use a lock option, which translates into having exclusive access to the Bluetooth dongle. This mechanism of

locking works together with the delivery system, so that content delivery and inquiry are exclusive.

The scanner works by executing hcitool to generate output, and posteriorly performs a set of modifications using UNIX standard tools. In particular, *sed* and *AWK* are used to perform these transformations, and consequently direct the generated content to a *named pipe* (FIFO) structure.

After the transformation, the content is forwarded through the FIFO to another Perl component named *btdevread.*pl. This component receives raw textual content and uses it to create device objects and maintain a list of these elements. Upon list update, this component will send the device list to the publishing system using another form of IPC – UNIX domain sockets. The sequence of events described is depicted in Figure 15.

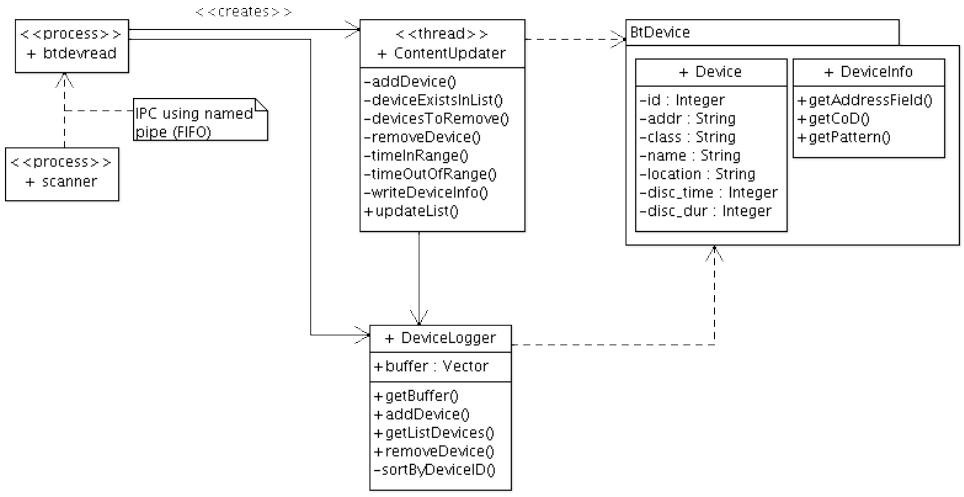

Figure 14: *Device Reader static view*

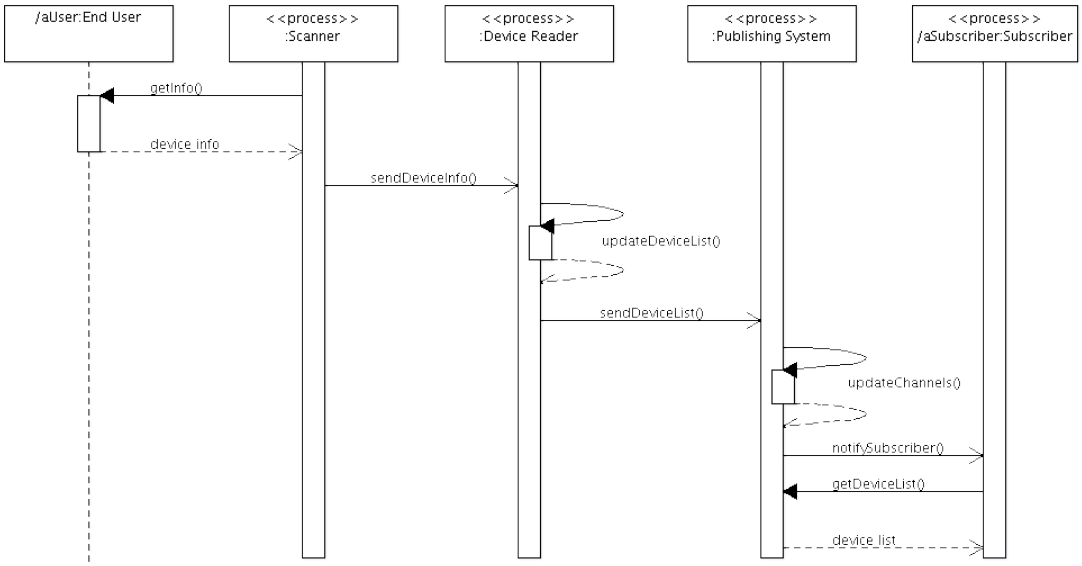

Figure 15: *Device Reader sequence diagram*

5.4.1.2 Publishing System

Before content from the scanning process is sent to interested parties, it is received by the publishing system. We use this *publisher/subscriber* architectural pattern to decouple the device reader from those elements interested in the device related information. Five distinct channels are made available by the publishing system for subscription, and are described as follows:

- *DEV_ENTRY* refers only to those devices that enter the scan range, and have not been seen in the recent past by the scanner.
- *DEV_EXIT* refers to those devices that were in range of the scanner, but haven't been seen again for a specific amount of time, and therefore have been excluded of the device list.
- *DEV_UPDATE* refers to those devices that already existed in range scan.
- *DEV_ENTRY_UPDATE* conjugates both DEV_ENTRY and DEV_UPDATE.
- *DEV_ALL* refers to all previous mentioned types of events (e.g. entry, exit and update).

The publishing system is also written in Perl, and works by using a UNIX domain socket server. It will wait for the device reader to send content along this path, and as an after-effect will proceed to channel content update and subscribed elements notification. The static structure of the publishing system is observed in Figure 16, where we see that many subscribers may be connected to the publisher.

Using this structure for passing information along the interested elements adds flexibility to the system, as we easily add or remove subscribing elements at run-time, and maintain the system functionality intact. For example, we could add a component responsible for estimating the number of people in the vicinity without any complications to the remaining components. It is Important to note that although the publishing system receives content through a UNIX domain socket, it passes the content along to subscribers using the most efficient method of IPC – shared memory segments [SR 05].

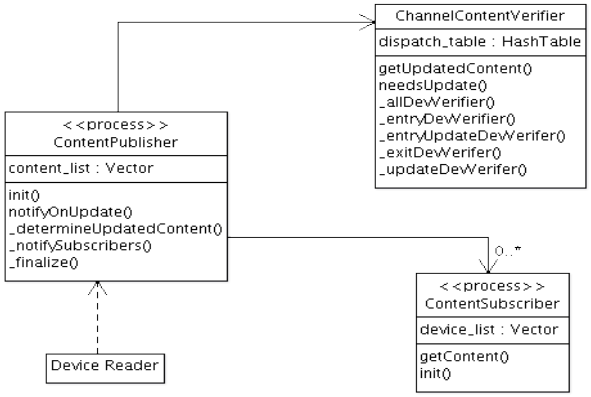

Figure 16: *Publishing System static structure*

5.4.1.3 Service Scheduler

After published, the list of devices is used by the subscribers. Currently, we have only one component which subscribes to content, using the DEV_ENTRY_UPDATE channel. That component is the service scheduler.

Responsibilities associated with it include service queue management, translation of service objects into command objects, and maintenance of blacklisted addresses. The class diagram of the service scheduler component is seen in Figure 17.

Service objects (Figure 18) are composed of a set of attributes, and have another class associated with them – the *ServiceFlag* class. These latter objects are used so we can define the particularities of services using what we term as *flags*. In reality these provide a rather flexible way to define service behaviours, as it is possible to specify and combine an array of these flags.

The service scheduler is a multi-process program which uses shared memory segments for inter-communication among local component processes. As it uses a DEV_ENTRY_UPDATE channel, the service scheduler will receive an updated list of devices every time a new device is seen, or when information related to existing devices is updated. The verification of this device list happens synchronously, as the main process is accountable for analysing both service queue and device list, and determining if any service needs to be executed. If multiple dongles are used, the system supports multiple simultaneous service execution. Further, if only one multicast service is being executed, and multiple dongles are being used, the system has the capability to use distinct dongles for delivering content to different destinations.

For determining which services to execute the scheduler performs a series of tests on the service objects present in the queue. The component responsible for this is an instantiation of the *ServiceReader* class, and it works by verifying if a service conforms to the flags specified upon creation. The method used for flag verification is seen in Figure 19. Only when a service passes all the tests is it marked by the *ServiceScheduler* for execution.

Although the ServiceScheduler class determines which services are to be executed, it is not its responsibility to do the actual execution. Instead, a new process is spawned and a class of type *ServiceHandler* is created. The first step taken by this component is to verify if there is the need to fetch content for the service. This is only the case when the *TARGET* flag is set, indicating the URI where content is to be fetched from. On the other hand, the ServiceHandler will not worry to fetch content if content is static, which is the

case when content is directly embedded into the service object upon creation (an example is seen in Code 1). The ensuing step made by the ServiceHandler is to proceed to the translation of the service object into a command object. After this is done, the ServiceHandler opens a connection, using a UNIX domain socket, with the delivery system, and instruct this component to add and execute the command. The ServiceHandler will then wait until completion, and update the Service object accordingly.

The actual update of the service is done by the ServiceScheduler class. This class implements the interface for actual service manipulation, and therefore all service related operations are made by it. Updating a service object will usually consist of updating the shared memory segments, but also the DBM file so that data persistence is maintained. Some updates, such as marking a service as non-active, are not (and should not be) reflected to the DBM file, as this will not only undermine performance, but as well make the service scheduler initialization process further complicated.

Further important is the fact that the service scheduler component listens for outside connections, enabling for remote control over the component. This is implemented by the *ServiceSchedulerProxy* class, which will create a UNIX domain socket and wait for connections to come in. When a remote request is made, the ServiceSchedulerProxy component will delegate execution to the *ControllerRequestHandler*, and thereafter will send the reply back to the client how issued the request. This strategy of delegating a request to other component was used throughout our system, and it is based on the object-oriented *Proxy* pattern [GHJV 95, Martin 03].

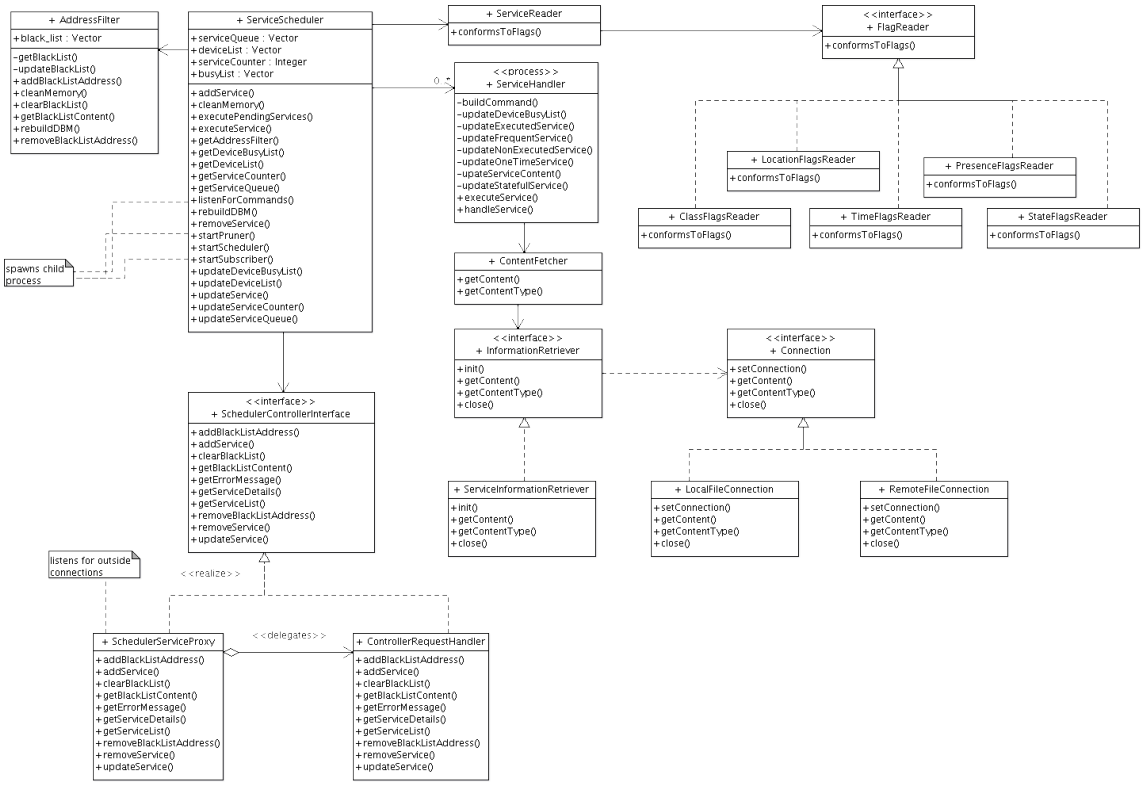

Figure 17: *Service Scheduler class diagram*

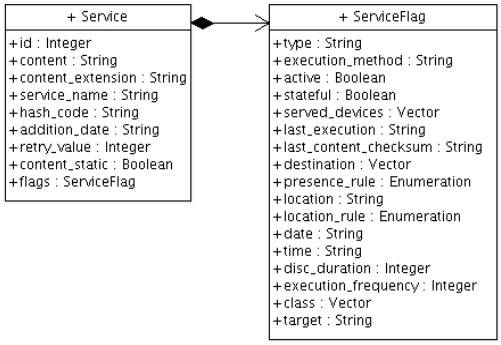

Figure 18: *Service class*

```
my $service = BtService::Service->new(
      CONTENT => 'Happy new year',
      CONTENT_EXTENSION => 'txt',
      FLAGS => {
            SERVICE => {
                  EXECUTION_METHOD => 'OBEXPUSH',
                  STATEFULL => 1,
            },
            TRIGGER => {
                  DESTINATION => ['FF:FF:FF:FF:FF:FF'],
                  LOCATION => 'UMa',
                  DATE => '01/01/10',
                  TIME => '*;*',
            }
      }
);
```

Code 1: *Creation of a service that broadcasts a "Happy new year" message on the 1st day of the year 2010, all day long at location "Uma". The service is statefull and therefore remembers already served devices*

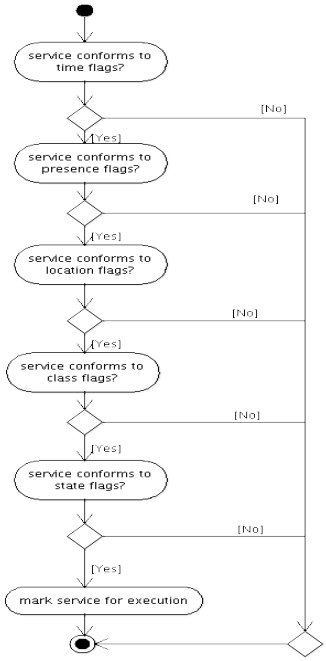

Figure 19: *Service flags verification*

5.4.1.4 Delivery System

Delivering content to the end-user is the responsibility of the delivery system. After the service scheduler creates services and translates them to command objects, the delivery system tries to execute them, and consequently disseminate content to the user.

The delivery system also works in a multi-process environment. Upon initialization two processes exist, and when a command has to be executed a new process is created. The inter-communication done inside the delivery system is, analogously to the service scheduler, done using shared memory segments.

When a command arrives to the delivery system, the first component to receive it is an object of type *CommandListener*. This component uses a UNIX domain socket to listen for command objects sent by the service scheduler. Upon reception, the CommandListener determines the local operation to execute, which is one of two: add the command to the queue, or mark it for execution.

For a command to be executed, it has to be passed to an object of type *CommandDispatcher*. This module implements all the logic necessary for determining which *CommandHandler* object needs to be instantiated, so that the actual execution of the command can proceed. The CommandDispatcher maintains a list of Bluetooth interfaces, marking them accordingly when a command execution is in place. Also, this component has the ability to work together with the device reader, by using a locking mechanism that ensures exclusive access to the Bluetooth dongle, if a single dongle is present.

The referred method of command execution is based in the *Command Dispatcher* pattern [DF 01]. By using this pattern we achieve greater flexibility for the delivery system. First, we enable for handler registration, removal and replacement at run-time, without the need for component reinitialization. Secondly, by putting the command execution logic into the CommandHandler objects, we gain the ability of creating *lightweight* command objects, and therefore transmit these much more efficiently across processes.

In Figure 21 we have depicted the sequence of events that basically happen in a delivery process. The content is received by the CommandListener, and by consequence the method *handleCommand()* is

issued to the CommandDispatcher. This later component then proceeds to the instantiation of the correct CommandHandler object and interface reservation (*buildCommand()* method), and finally demands for the delivery of the content to the end-user. The CommandDispatcher gets the operation status from the CommandHandler, and prints the information related to it. It also replies to the service scheduler component, so that the scheduler can update its service queue accordingly.

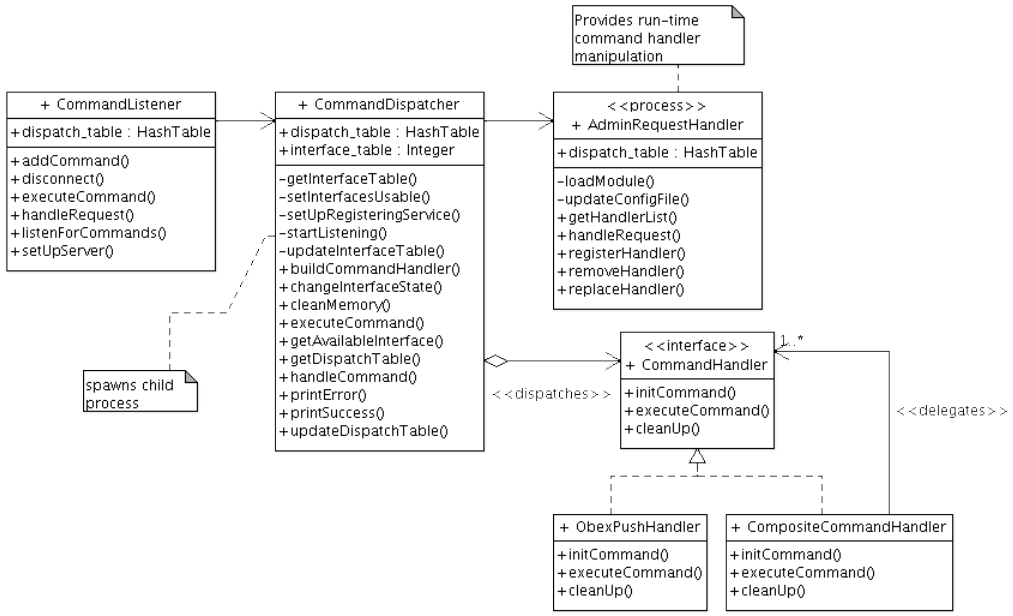

Figure 20: *Delivery System structure*

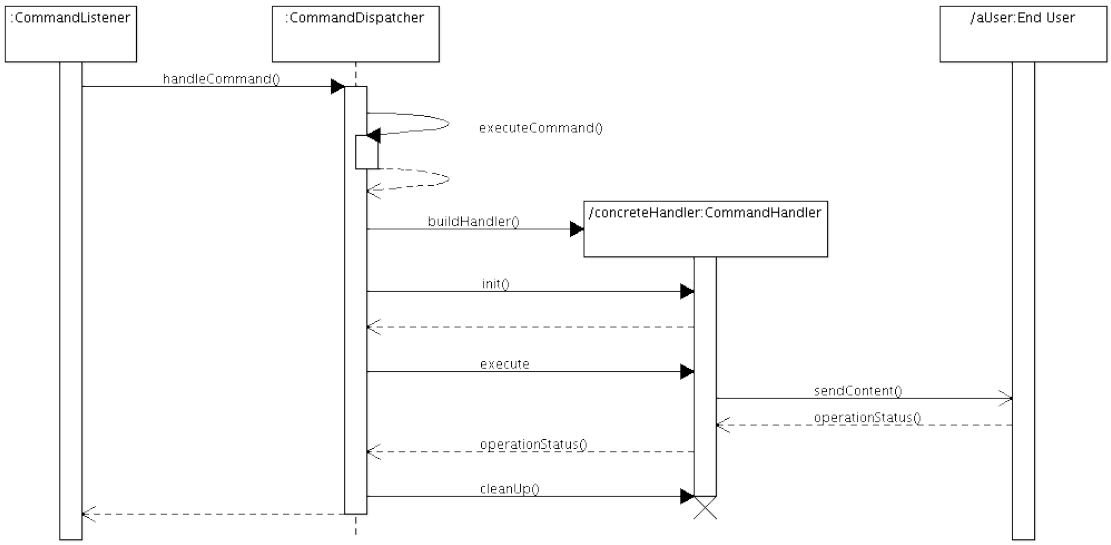

Figure 21: *Sequence diagram for delivery process*

5.4.1.5 Communication System

The component responsible for communicating with the outside world is the communication system. It lies in the boundary of the Blue Station, and works very similarly to a proxy. The device reader, service scheduler and delivery system all implement mechanisms for remote control, but they listen only for connections using UNIX domain sockets.

By creating a component whose solely responsible for inter-network communication, we never expose internal elements directly to the outside world. Instead, all communications must pass through the communication system, as it decides to which local component a request is to be forwarded. This is better understood by looking at Figure 22, where we see that the only component that uses Internet sockets is the communication system. All other components are restricted to local communication.

Another aspect of the communication system is that it accepts both local and remote requests. In other words, local components can ask for the communication system to forward specific requests to the broker (and consequently to the central), but it also accepts remote requests from the outside world. Forwarding local requests is straightforward, as the link with the outside world is 1-to-1. On the other hand, forwarding remote requests

involves determining which component is the content intended too. For achieving this, we associate each operation with a specific component by using a *dispatch table*, a technique utilized in many system's components. The components that constitute the communication system are seen in Figure 23.

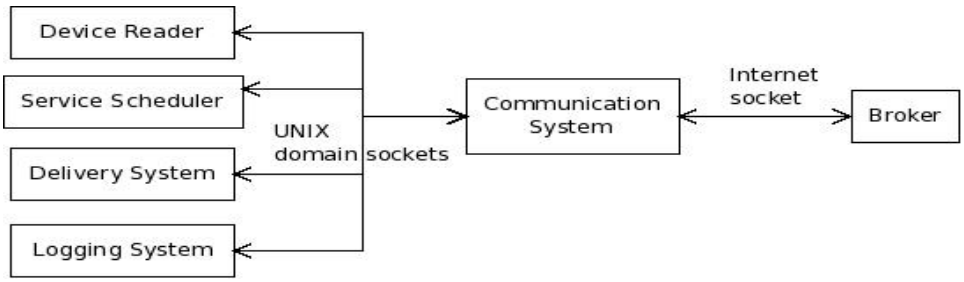

Figure 22: *Communication System methods of communication*

Another aspect to mention is that local components can send buffered requests to the communication system. This way, requests are serialized and stored in a file, as the communication system periodically checks to see if content needs to be sent. Buffer is cleared when content is successfully sent to the destination.

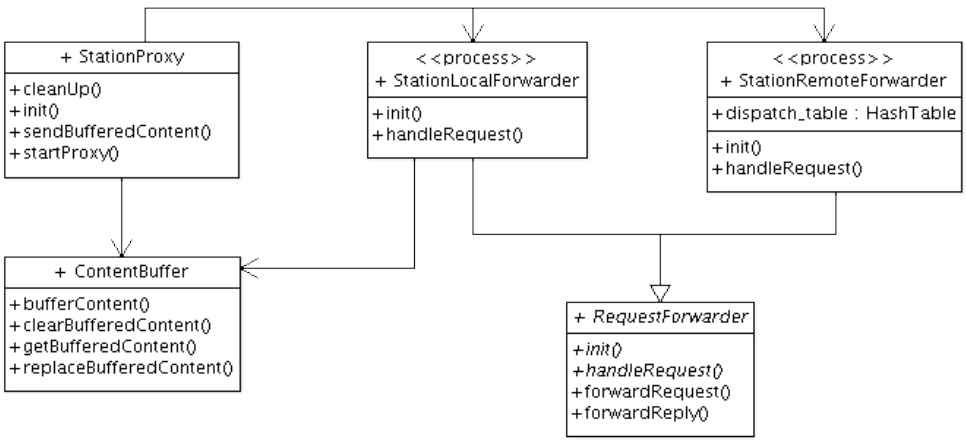

Figure 23: *Communication System class diagram*

5.4.1.6 Logging System

The logging system has a simple structure (Figure 24), and its only responsibility is to collect the log files created by the remaining components (e.g. device reader, service scheduler, delivery system, and communication system).

When the Blue Station is started, a script is responsible for scheduling execution of the logging system. It uses the user's *crontab* entry for instructing the operating system to run the logging system everyday at 23:59. Thereafter, the logging system fetches the content present in the log directory, creates a compressed file, names it accordingly, and finally sends a buffered request to the communication system, so that the log content is transmitted to the central infrastructure. Code 2 shows the steps taken for scheduling the logging system.

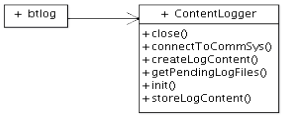

Figure 24: *Logging System class diagram*

```
#Schedule for logging content delivery to central
if [[ -z "$BLUE_HOME" ]]; then
  echo "BLUE_HOME variable is not set. Cannot set up crontab entry for
content logging delivery. Quiting..."
  stopStation
fi
if [[ ! -e cron_command ]]; then
  echo "59 23 * * * $BLUE_HOME/LoggingSystem/exec_log.sh $BLUE_HOME"
>cron_command
  #Error occured
  if [[ $? != 0 ]]; then
    echo "An error occured while schedulling for content sent to central
server. This will need to be done manually"
  #Append created cron content to crontab
  else
    #Put existing cron tab entrys into temporary file
    crontab -l >orig_cron_tab
    #Append to cron_command file and add it to cron
    cat orig_cron_tab >>cron_command
    crontab cron_command
    if [[ $? != 0 ]]; then
      echo "Couldn't set up cron command set up. Logging System is not
scheduled to send log content"
    fi
    #Clean up the recently created cron_command file
    rm cron_command
  fi
fi
```

Code 2: *Code for schedulling logging system execution*

With the discussion of the logging system we conclude our overview of the Blue Station component. Let's now proceed to the intermediary element – the Broker.

5.4.2 Broker

As an intermediary component, the broker is involved in all communications between stations and the central infrastructure. As such, the broker acts as a naming server, but also as a request/reply forwarder. The behaviour of the broker is similar to the communication system of the stations. In reality, the *BrokerConnectionHandler* implements the same abstract class that some of the components of the communication system – the *RequestForwarder* (Figure 25).

The addition of the broker allows us to gain run-time name lookup, an important aspect if we consider that stations may have dynamic addresses, and that referring to them by a numerical IP would be problematic. Further,

the broker is present in all connections to and from the stations. This is advantageous because if we assume that stations may not be able to listen directly for outside connections, the use of tunnelling techniques is necessary. With the existence of an intermediary component, we remove the burden of the stations and central infrastructure to implement concrete communication logic, as we need to worry only in implementing the code necessary for them to connect to the broker.

The broker maintains a data structure which allows it to associate a station's name with a concrete address. This structure is an hash table, and is termed as a *server table*. A station must register with the Broker before the remaining components are able to contact it. This is done when the station component is initialized, where there is a direct request to the Broker to perform registration. If the station is said to be already registered, the station will then request for replacement of the address, as there is a chance that the address has been modified.

Although security is not one of our main concerns at this point, we implemented a simple mechanism for making sure that a station cannot camouflage as another. Upon the first registration, a station is given a random *hash code* that it must store in order to identify itself in the future. By observing Figure 26 we see the sequence of events that must take place in order for a remote operation to be executed.

In order to support multiple connection forwarding, the Broker component uses a multi-process strategy. When a new request comes in, a new child process will be spawned. If the request is to be forwarded to a station or to the central infrastructure, the Broker will locate the component and establish a connection with it. The process is independent from the registration type, as the Broker maintains only a simple string composed by address and *port* of the component.

As the overview of the Broker is made, we can proceed to the discussion of the remaining component of the system – the central infrastructure.

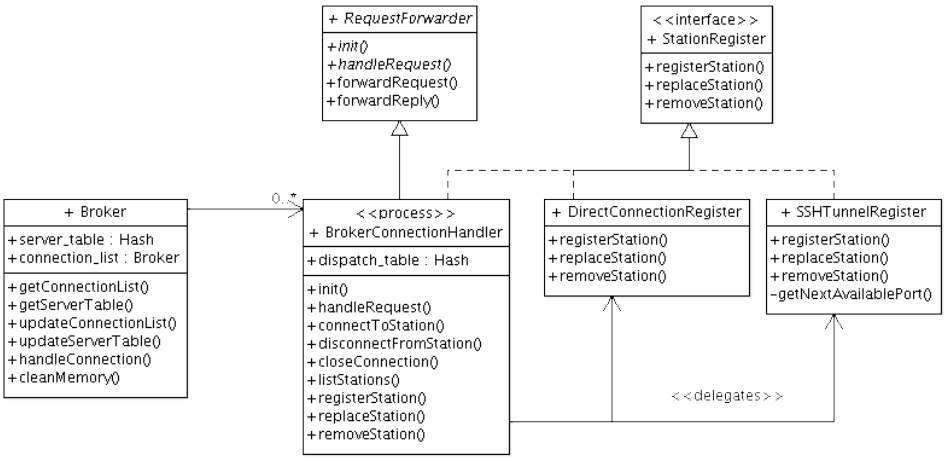

Figure 25: *Broker class diagram*

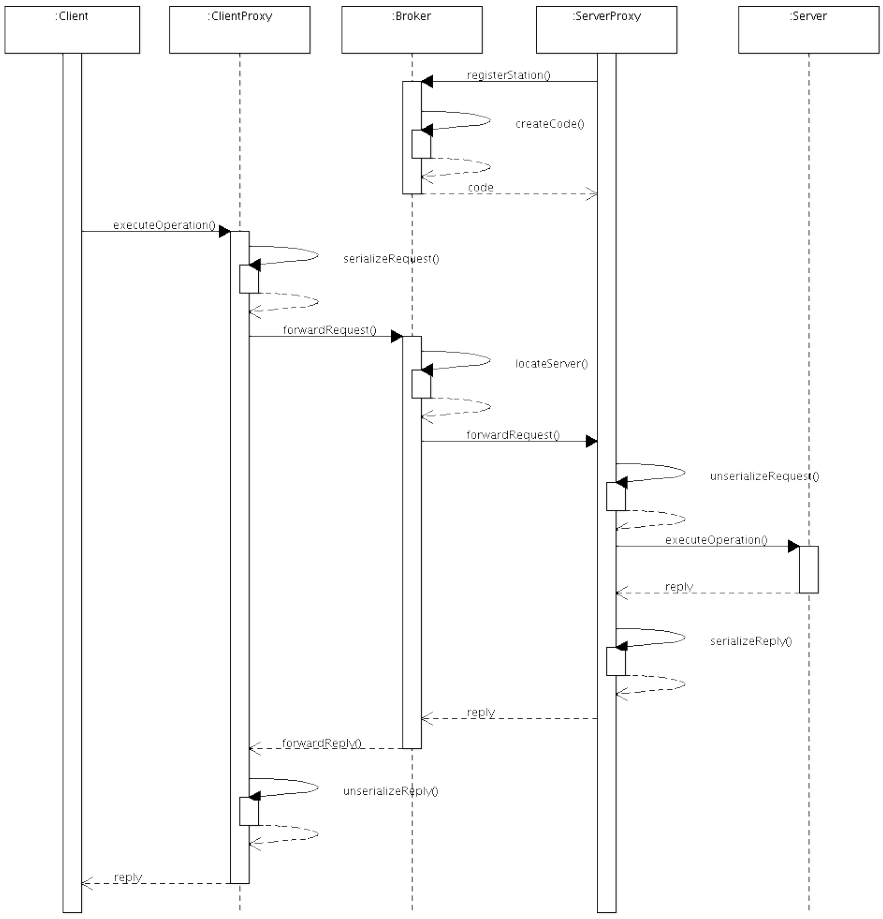

Figure 26: *Broker component behaviour*

5.4.3 Central Infrastructure

Composed by several different components, the central infrastructure has the responsibility to log content, manage the database, and provide the mean to control individual stations. Element disposition is seen in Figure 27.

The most important component is the s*tation manager*. Its objective is to provide the means for remote station management. It is a multi-process program, and it works by listening for requests using a UNIX domain socket. The classes that constitute this component are shown in Figure 28. As is observed, this component has the responsibility of database management. When a client issues a request to the station manager, the *StationManager* class creates, after spawning a new child process, a new

StationControllerProxy object to handle the request. This element has the responsibility of performing database management – using the *DatabaseManager* class – and delegate execution to the *StationController* class, which implements all control station related methods.

Receiving the content sent by the stations' logging systems is the responsibility of the *log manager.* It listens for incoming local connections, and upon concrete request it creates the log directories and files, so that each station has a dedicated directory for storing log content.

Working similarly to the station communication system, the central infrastructure communication system will provide inter-network connectivity. Like the blue stations, the central infrastructure must execute a registration process with the Broker. The communication system has the responsibility of registring with the Broker, and act as a request forwardered. The static structure of this element is seen in Figure 29.

Given the description of the central infrastructure we move to the discussion of the administration console.

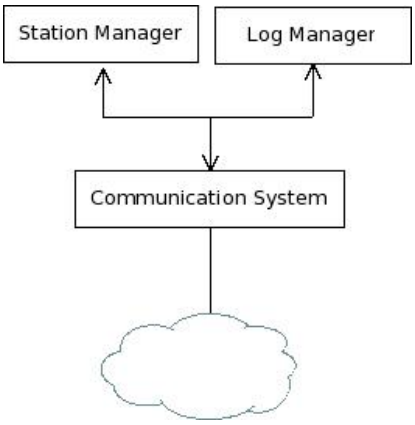

Figure 27: *Central Infrastructure elements*

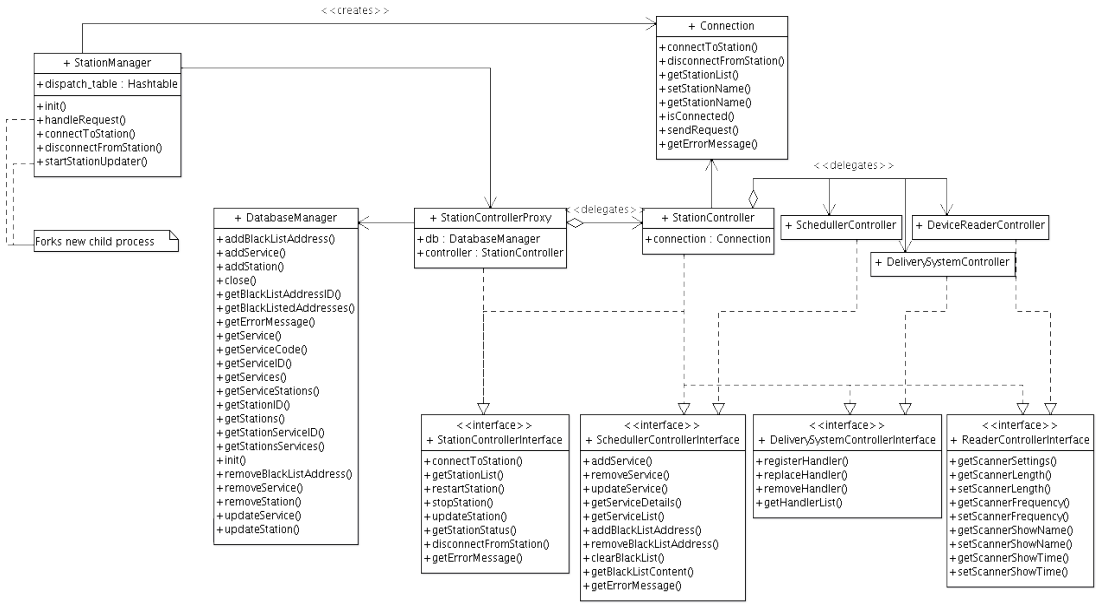

Figure 28: *Station Manager class diagram*

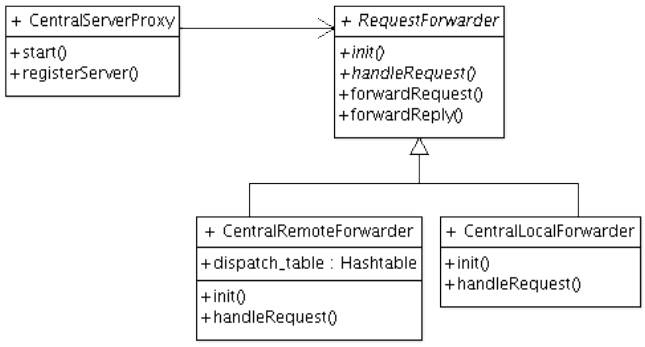

Figure 29: *Central infrastructure communication system*

5.4.4 Administration Console

A component not discussed until now is the administration console. Its objective is to provide an interface so that station and service management is possible. The administration console also connects to the broker component, as it needs the broker naming capabilities to consequently connect to individual stations, and also to the central infrastructure.

The administration console is built upon a *Model-View-Controller* (MVC) architectural pattern. This is done so that we may easily migrate from a console based interface to a graphical based one. Using MVC we separate concerns and as such future modification of interaction method is made simpler, as we have only to modify the View classes. Figure 30 shows the administration console composing elements.

With the use of the administration console we may connect to any station registered with the broker. A list of commands is made available upon initialization of the console, as we indicate the name of server to which we wish to connect. After successfully establishing a connection to a station, the list of possible commands expands, as we are able to control all the elements that constitute the station (Figure 31). As an example, in Figure 32 we see the output produced by the *ds* command, that shows the details related to a specific service.

Also, it is possible to connect to the central infrastructure using the administration console. Upon successful connection, a different set of commands is made available to the administration, as he gains the possibility of controlling individual stations, as well remove stations from the broker and database.

Overall, the administration console allows us to remotely manage both stations and the central infrastructure by using the broker's naming lookup capabilities. It is possible to run the console in any location, as long as the minimum necessary modules are available.

With the discussion of the administration console we come to the end of our system overview. In this section we presented a more detailed specification of the system than in the two previous sections. For further system understandability, and its functionality logic, we present in the ensuing section a more detailed scrutinization of several important aspects of the system.

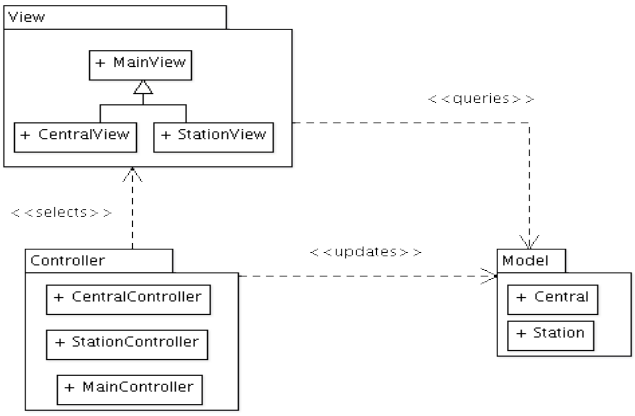

Figure 30: *Administration console class diagram*

```
   h    Present this command list
   q    Quit this program

[Not connected] Command (use 'h' for help): cc
Enter station name: UMa
Connection with server 'UMa' established

[Connected to UMa] Command (use 'h' for help): h
Command Action
   General Commands
   cc   Connect to a blue station
   ct   Connect to central station manager
   dc   Disconnect from station/central
   lc   Retrieve the list of registered blue stations
   h    Present this command list
   q    Quit this program

   Service Scheduler Commands
   ls   List all existing Services
   ds   Get the details about a specific Service
   lb   List all addresses present in the black list

   Scanner Commands
   lsc  List the current scanner settings
   ssl  Change the scanning length
   ssf  Change the frequency between scans
   ssn  Toogle the show device name flag
   sst  Toogle the show discovery time flag

   Delivery System Commands
   lsh  List the registered command handlers
   adh  Add a new command handler
   rdh  Replace an existing handler
   rmh  Remove an existing handler

[Connected to UMa] Command (use 'h' for help):
```

Figure 31: *Administration console help menu when connected to station 'UMa'*

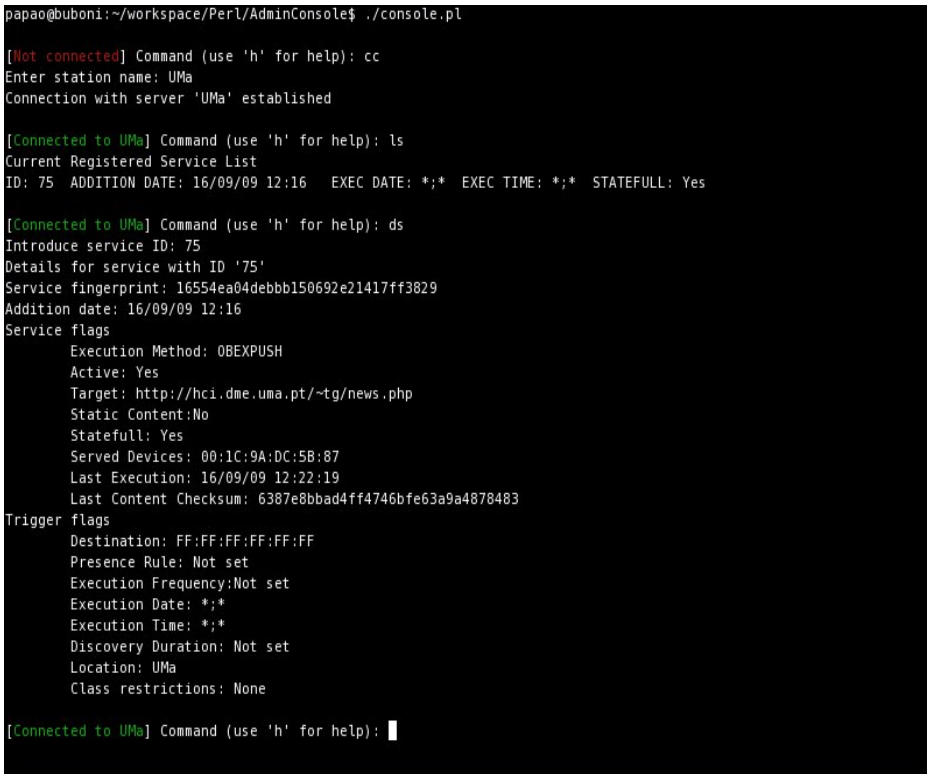

Figure 32: *Accessing service details using the administration console*

5.5 System Particularities

In the previous section a detailed overview of the system was given. With it, we demonstrated the philosophy of the system and described, with a relative level of detail, its components. A complete detailed of the system is beyond the scope of this section, but it is still important to delve deeper into some elements, as they are crucial for understanding the method of functionality.

5.5.1 Blue Station Configuration

Because the Blue Station was developed to run in Linux, and possibly in other UNIX-like operating systems, there was the clear need to devise some mechanism for system configuration. As always, it is not desirable to embed configuration details into the source code. Users should not have to edit any source tree file for modifying the configurable system aspects. Instead a better way is to use a *configuration file* to achieve this.

- The blue station component uses a shell variable and a configuration file for option specification. After unpacking the station component, the output content will include an installation file named *install.sh.* This file exports to the user *profile* the *BLUE_HOME* variable. This variable is used to determine the installation directory of the station component, where it is included the configuration file – *bluestation.conf*.
- Device reader options are the first to be presented, as the user has the ability to change the duration of the scan (*SCAN_LENGTH*), its frequency (*SCAN_FREQUENCY*), if friendly names should be output *(SHOW_DEVICE_NAME*), and if a time stamp should also be output (*SHOW_TIMESTAMP*). By default these options are set to 8, 5, 0, and 1 respectively.
- The delivery system options are presented next, as it is possible to define the command types supported by the system (*COMMAND_TYPE*), and the respective handling class (*COMMAND_HANDLER*). A maximum up to 10 distinct pairs may be set. By default, the system only supports the type *OBEXPUSH*, and the handler *BtCommand::ObexPushHandler*.
- Service scheduler option include setting the name of service DBM file (*SERVICE_DBM_FILE*), and of the blacklisted addresses (*BLACK_LIST_DBM_FILE*).
- The last set of options are overall settings. The administrator needs to set the name of the station correctly (*STATION_LOCATION*) so that registration is possible with the broker, which location is specified by the *NAMING_SERVER_LOCATION* and *NAMING_SERVER_PORT* options. If a SSH tunnel is to be used, then the option *USE_SSH_TUNNEL* needs to be enabled, and both *SSH_TUNNEL_USERNAME* and *SSH_KEYFILE* are used to specify SSH connection user name and private key authentication file. Also, the system currently makes use of the *autossh*[9] application so that tunnel reliability is ensured. On the other hand, if no tunnel is used, then the option *EXTERNAL_INTERFACE* indicates the interface used to connect to the outside world. The communication system uses this setting to monitor the interface and perform address replacement automatically when the address changes. The administrator indicates

[9] *www.harding.motd.ca/autossh/*

that multiple dongles are to be used, by enabling the *USE_MULTIPLE_DONGLES*, and by indicating the scanner (*SCANNER*) and delivery dongles (*DELIVERY_1, DELIVERY_2, etc*). Also, if multiple connections per dongle are allowed, the administrator needs to specify the number with the modification of *MAX_DONGLE_CONNECTIONS*.

Components will read the configuration file upon initialization, and use the settings throughout the process life. Processes also have the ability to modify the configuration file in run-time. This is easily done in Perl, as the dynamic and text-oriented nature of the language lets us treat text files as internal structures, facilitating the usage of the *regular expressions* engine.

Pertinent to say is that both reading and modification procedures are all done transparently to users, as all system (including the Broker and the central infrastructure) components run in the *background*, consequence of a procedure done upon initialization, known as *daemonization*.

5.5.2 Daemonization

Programs that reside in memory “permanently” are referred to as *daemons*. They are usually created upon operating system initialization, and are killed only when there is a shut down (e.g. restart or halt) procedure. One particularity of daemons is that they are non-interactive, autonomous programs that need no direct human intervention, and for that reason run in the background. Daemons run without a control terminal associated, and therefore other means of communication are used to control these processes, being signals one of the most common.

In order for a process to be daemonized, it must conform to a set of coding rules. Stevens & Rago [SR 05] define the following steps that should be performed in order to correctly daemonize a process:

- Disable the *umask* value. Umask is used so permissions can be enforced upon file creation. It is not wise to use such restrictions on a daemon, as it may have the need to create several types of files.
- Call *fork* to create a child process, and then call *exit* to kill the parent. This step is actually responsible for sending the process to the background.

- Call *setsid* for making the first child process as the session and group leader. This also allows for stating that the process needs no control terminal associated to it.
- Change the working directory. This could be root directory, or some other process related directory. What shouldn't be done is to *chdir* to a directory which can influence file system mounting operations.
- Close unnecessary file descriptors (or file handles in Perl), and redirect the standard streams. Many variations of standard stream redirection can be used here, as it all depends on the nature of the daemon. If for example a centralized log facility like *syslog* is being used, we'll want to redirect all standard streams to */dev/null*. On the other hand, if we maintain a separate log file for each daemon, we may want to void the standard input and output, and redirect the standard error to a log file.

Naturally, we followed these recommendations and implemented them in our system. Although these recommendations are made in reference to the C programming language, they can be easily transported to Perl.

As a form of exemplification let's consider the daemonization process of a blue station communication system. We start by using a set of constants that define the details of the daemon (Code 3). The first argument value is read ($ARGV), as this determines the need to proceed to daemonization. Furthermore, the process ID (PID), log file (LOG_FILE), and the component name (MODULE_NAME) are also defined.

Actual daemonization of the process happens after, as we first test for existence of a PID file to make sure that only one instance of the daemon exists. We then change the working directory, create the child and exit the parent, and finally make the child a session leader, redirect the streams, and create the PID file (Code 4).

Also important is the redefinition of signals (Code 5). We use Perl's special hash *$SIG* to define what kind of action should be performed upon reception of specific types of signals. A common step to take here, besides those shown in Code 5, is to define the *HUP* signal, and instruct the program to proceed to verification of the configuration file. This is done in several other elements of the system, like the device reader. It allows us to modify system

settings, update them in the configuration file, and instruct the component to restart and read the configuration file again.

As referred, the dameonization procedure dissociates processes from the controlling terminal. There is no need to allocate a terminal if no interaction is to happen between user and process. A common usage for daemon processes is to use them as a *server* components, that will wait for incoming requests made by users. This *client/server* architecture is indeed adopted by many of the system's elements, being the most evident case the service scheduler. When a user wants to create a service, he must connect to this element, and instruct for service creation. Many options are available to the user in this process, as this is the foci of discussion of the next section.

```
use constant      DAEMONIZE   =>    ($ARGV[0] =~ /^(-d)|(--daemon)$/) ?
1 : 0;
use constant      PID_FILE    =>    '/tmp/commsys.pid';
use constant      MODULE_NAME =>    'SysctlDaemon';
use constant      LOG_FILE    =>    $ENV{'BLUE_HOME'} .
"/log/comm_sys.log";
```

Code 3: *Constant definition for daemonization*

```
#Daemonize component
if (DAEMONIZE)    {

      #No two daemons can be running simultaneous
      if (-e PID_FILE)  {
            print_ts(${\MODULE_NAME},
                  "A daemon of this type is already running. If not
remove file '", PID_FILE,"'");
            exit(1);
      }

      #Change working directory
      chdir $WORKING_PATH;
      open STDIN, '/dev/null';
      open STDOUT, '>/dev/stdout';

      #Create child and kill parent
      defined(my $pid = fork) || die "Fatal: can't fork [$!]";
      exit if $pid;

      #Make child session leader, redirect standard error, and create
PID file
      setsid;
      $PARENT_PID = $$;
      open(STDERR, ">>", LOG_FILE);
      open(PID_FH, ">", PID_FILE);
      print PID_FH $$;
      close(PID_FH);
}
```

Code 4: *Daemonization procedure*

```
#Set up signal handlers
$SIG{CHLD} = 'IGNORE';
$SIG{INT} = $SIG{TERM} = sub {

      my $signal_type = shift;

      print_ts(${\MODULE_NAME}, "Got $signal_type signal. Quiting...");
      kill TERM => $_ for (@child_pids);
      unlink(PID_FILE);
      unlink(STATION_PROXY_SOCK);
      $sysctl->cleanUp();
      exit(0);
};
```

Code 5: *Redifinition of signals*

5.5.3 Service Specification Language

In order to offer greater flexibility to the user, our system supports the definition of a set of options upon service creation. These *flags* that we allow

the users to define, characterize the service, and therefore we see them as a *service specification language* (SSL).

Although it is a rather rudimentary language, the SSL allows for specification of diverse contextual dimensions. The flags are separated into two distinct groups: service flags and trigger flags. The first set refers to the definition of the service, and the second is related to the context that must be observed in order for the service to be eligible for execution. All these flags are set in the *ServiceFlags* class (see Figure 18). Additionally, the *Service* class also allows for indication of several options. In both cases, some options are supposed to be set directly by the user, and others updated according to service execution. The following list presents the currently supported flags by the system:

- **Main service flags**
  - CONTENT – If set directly, the service has static content. Note that only textual content can be set directly.
  - CONTENT_EXTENSION – This flag indicates the type of content. If it is simple text, then it would indicate *.txt*. On the other hand, if it is a jpeg image, it would indicate *.jpg*. This is set automatically by the system.
  - HASH_CODE – This refers to the *fingerprint* of the service. It is automatically set upon service creation. A *MD5* hash is used to generate a set of hexadecimal digits that will uniquely define a a service. We accomplish this by using *Data::Dumper* module, which *stringfies* Perl's internal data structures, making it suitable for applying the hash algorithm.
  - SERVICE_NAME – This defines the name of the service. It is used upon random file creation. End users will receive a name containing the string defined in this field, along with other random characters.
  - ADDITION_DATE – The date of addition to the service queue. Automatically set.
  - RETRY_VALUE – How many times has the service been executed? This is automatically set by the system
  - ID – Defines the unique local number of a service. Set by the system.

- CONTENT_STATIC - The system automatically sets this flag to 1 if the service has static content.

- **Service definition flags**
  - TARGET – This indicates the source where content is to be fetched from. When this is set, the content is said to be dynamic. The input must be in the form of a URI, and can be either local or remote.
  - EXECUTION_METHOD – The method used to deliver content to end users. Currently only OBEXPUSH is supported.
  - ACTIVE – This indicates if a service is currently active or not. This is automatically set by the system.
  - STATEFULL – This flag indicates if a service maintains state. In other words, if this is set, the service will remember the addresses to which content has already been delivered too, and will not deliver the same content again.
  - SERVED_DEVICES – An array of devices to which content was successfully delivered. If a service maintains state, then it won't try to deliver content to those devices contained in this array. This is automatically managed by the system.
  - LAST_EXECUTION – The date of the last service execution. This is set automatically by the system.
  - LAST_CONTENT_CHECKSUM – If the service maintains state, and the content is dynamic, then the system will periodically check the content to see if changes occurred. Analogously to the technique used for defining the service fingerprint, we use MD5 checksums to determine changes in the content. If the content has changed, then the system will clean the SERVED_DEVICES flag, and deliver content to all surrounding devices. This is automatically managed by the system.
  - BROADCAST – This flag is set to 1 if we want to deliver content to everyone. This is automatically set by the system.

- **Service contextual flags**

- DESTINATION – This defines the address(es) to which content is to be delivered to. The special address *FF:FF:FF:FF:FF:FF* indicates that this is a broadcast service. Setting this address will make the system enable the BROADCAST flag.
- PRESENCE_RULE – This flag accepts two options: *ANY* or *ALL*. Either of this flags one makes sense if multiple destinations have been set. If this is the case, then ANY indicates that the service is to be executed if *any* of the destination addresses comes into scanning range. On the other hand, if ALL is set, then the service will only execute when *all* destinations have come into scanning range.
- LOCATION – The location where content is to be delivered. Indicating more than one location indicates that the service is to be added in more than one location.
- DATE – This indicates the service's date of execution. Not only concrete are supported, as *wildcards* are also acceptable. If we want a service to run forever we would indicate *;*. If we wanted a service with a start date, but not an end date we would use *dd/mm/yy;*.* Contrarily, if we wanted a service which only runs for a day, we would indicate a concrete date with the format *dd/mm/yy*.
- TIME – Indicates the time of execution. The same wildcards supported in the DATE flag are also supported here.
- DISC_DURATION – The minimum time that a destination address must be in scanning range before service execution is tried.
- EXECUTION_FREQUENCY – This flag indicates that a service, independently of other flags, is only to be executed if the defined amount of time has elapsed since the last execution.
- CLASS – An array which restricts the classes of devices to deliver content to. We can indicate just the major class, or we can further specify the minor class. If we want to restrict our class of devices to just mobile equipment, we use the hexadecimal value 0x000200. It is advisable to impose this kind of restriction, as often we are only interested in delivering content to mobile equipment and personal computers.

With the flags here described users can build several types of services. The philosophy is simple: define the type of content a service uses, and thereafter define on which context should the service be executed. As an example, lets consider the creation of a *news* service. We want to fetch RSS headlines from a specific site, and transform and deliver them to public transit users. For accomplishing this task we do the follow:

- Create a server side script which fetches the content from a website. This script will then transform the content so that unnecessary text is filtered out.
- Create a service which points to the above script, that simply outputs the transformed content.

We start by building the script responsible for fetching and transforming the news content (Code 6). By using grep and sed capabilities, we easily remove HTML related information, and shape the content for textual delivery. Thereafter, a simple PHP (Code 7) will be responsible for outputting the transformed content. Finally, we define a service with the options seen in Code 8. It is seen that we point to the PHP script hypothetical location, by defining the TARGET flag. Also, the service maintains state, and delivers only to mobile equipment, such as cellphones and PDAs.

This overview of the service specification language, and in particular the contextual flags, set the foundations needed to understand the process of determining if a service is to be executed – the subject of the next section.

```
curl http://some_news_services.com/RSS/Feed/news/homepage  | grep
"<title>" | sed 1d | sed -e 's/\&amp\;/\&/g' -e 's/<title>//g' -e 's/<\/
title>//g' -e 's/^ *//g' -e ' i\
\
'
```

Code 6: *Script responsible for fetching and transforming news headlines*

```
<?php
echo "We're proud to present the news headlines for today\n";
echo "\n";
$news = shell_exec('./get_news.sh');
echo html_entity_decode($news, ENT_QUOTES,'UTF-8');
echo "\nFonte: NEWS PROVIDER";
?>
```

Code 7: *Script responsible for outputing transformed content*

```
my $service = {
      FLAGS =>    {
            SERVICE =>  {
                  EXECUTION_METHOD => 'OBEXPUSH',
                  TARGET => 'http://server.somewhere.com/news.php',
                  STATEFULL => 1
            },
            TRIGGER =>  {
                  DESTINATION => ['FF:FF:FF:FF:FF:FF'],
                  LOCATION => 'Bus_Stop',
                  DATE => '*;*',
                  TIME => '*;*',
                  CLASS => [
                        {
                              MAJOR => 0x000200,
                              MINOR => undef
                        }
                  ]
            }
      }
};
```

Code 8: *Creation of the news services*

5.5.4 Service Execution Process

After the creation and addition of services, it becomes the responsibility of the service scheduler to determine if the contextual scenario, defined by the services flags, has been met. As such, the scheduler constantly verifies the service queue, and analyses the objects included within.

Verification starts by first determining the number of Bluetooth dongles made available for delivery. If just one dongle is available, or one dongle is used for both scanning and delivery, then the system will conclude that just one service must be executed at a time. On the other hand, if two or more dongles are available, then the possibility of simultaneous service execution exists.

Only those services marked as active (e.g. *not* being *executed*) are considered for addition into the execution list. Of that list, the system chooses those services who conform to current contextual settings. The sequence of verification has already been seen in Figure 19. Before services are actually verified, they are *sorted* by ascending order of creation. This sorting is done at *insertion* time, and the objective is to provide higher precedence to older services for flag conformity verification. The kind of sorting used is a simple *straight insertion* technique, as defined in [Knuth 98].

After determining the services to execute, the system again analyses the list, and gives priority to stateless services. The number of times a service has been tried to be delivered is also taken into account, as the system sorts by ascending order of retry. This same procedure is repeated for stateful services.. In order to better understand the textual descriptions given, Figure 33 presents an activity diagram, that illustrates the algorithm used in the process of selecting the services to execute.

Subsequently, and after the execution list is determined, the system needs to identify to which devices content is to be disseminated. The first step of this process consists on the analysis of the target device list, created in the previous step. Since the system doesn't want to try and send content to devices to whom content is already being tried to be delivered to, it removes those devices from the target device list. Clearly, if the target device list is empty, then the service cannot be executed.

After the determination of the target device list, the system then proceeds to randomly choosing elements of that list. The simplest case is when a service has only one destination, in which the system does not perform additional computations. On the other hand, if the service has multiple destinations, then the maximum number of random devices to choose depends on the following factors:

- If the system is executing solely one service, then the maximum number of target devices is given by the number of available delivery dongles multiplying by the number of allowed simultaneous connections per dongle.

- If the system is executing more than one service, then the maximum number of target devices is given by the number of allowed simultaneous connections per dongle.

Basically, this means that although the system limits the use of a dongle per service, if just one service is in the execution list, the system can utilize multiple dongles to broaden the number of target devices to disseminate content to. This textual description is better comprehend by looking at the algorithm depicted in Figure 34.

Finally, and after marking all services in the execution list as inactive, the system executes the steps depicted in Figure 35. A new child process is created for each service, and consequently the child has the responsibility to communicate with the delivery system, and wait for service execution reply. It then marks the service accordingly (e.g. failure or success), changes its active status, and finally upgrades the service queue.

As we detailed the process of service execution, we saw that the scheduler is not the element responsible for actually delivering the content to the end-user. Instead, it simply translates from service to command objects, and passes them along to the delivery system. The actual process of delivery is what we detail in the next section.

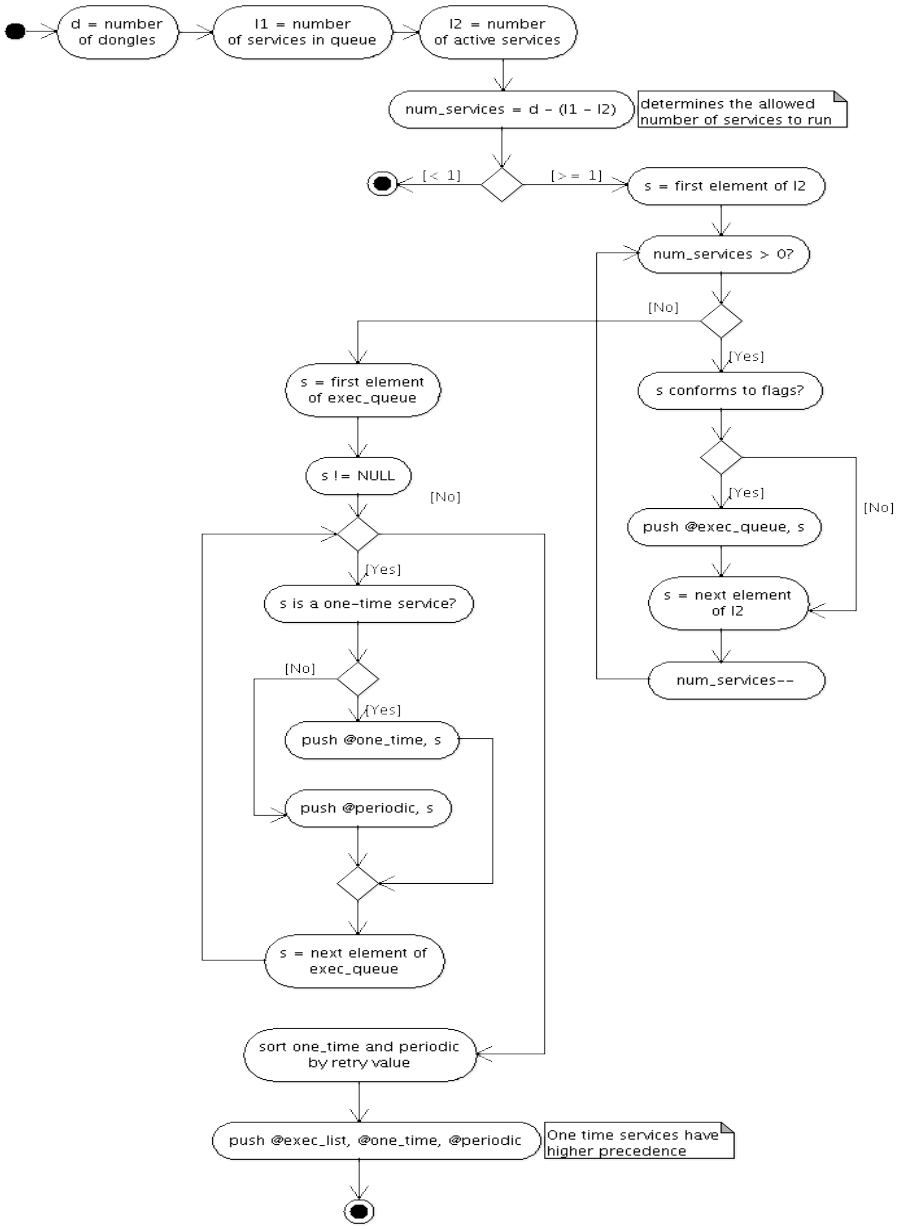

Figure 33: *Determining the list of services to execute*

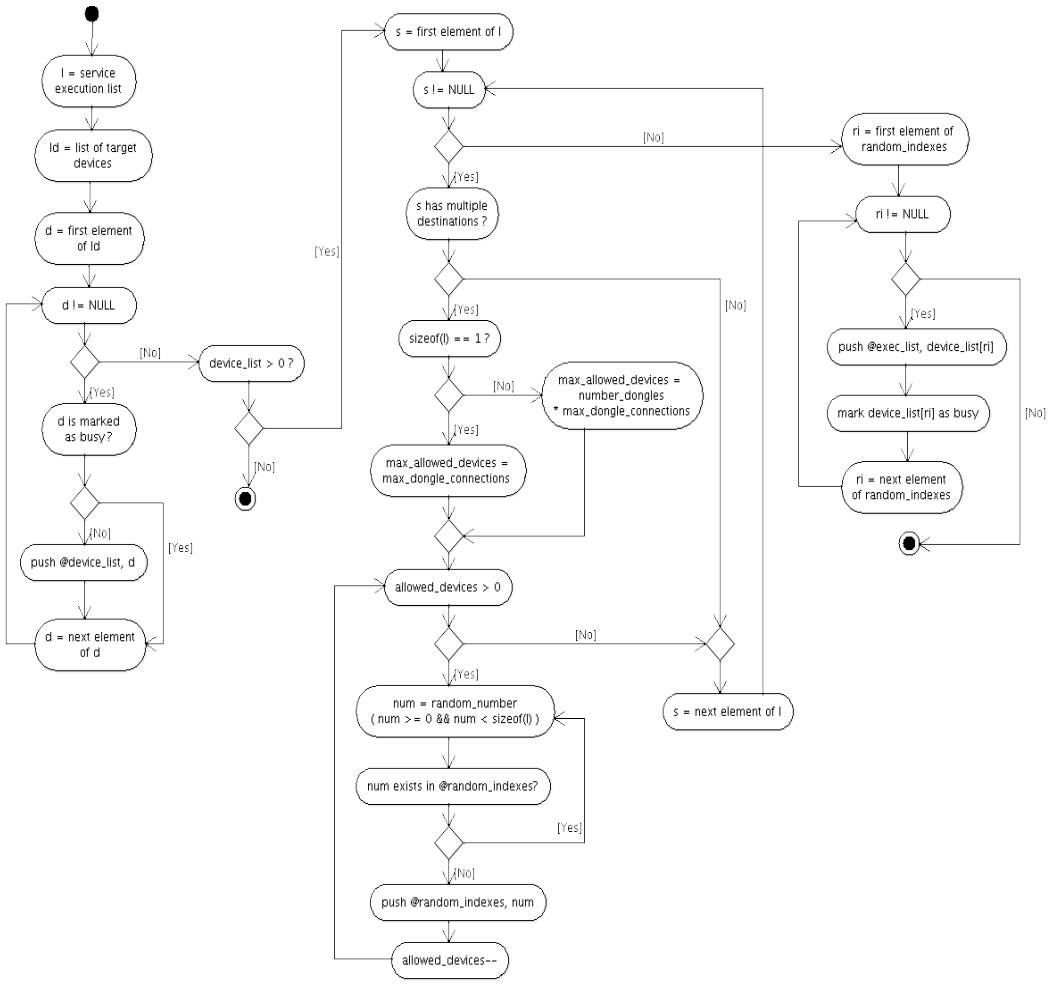

Figure 34: *Device filtering and dongle availability determination*

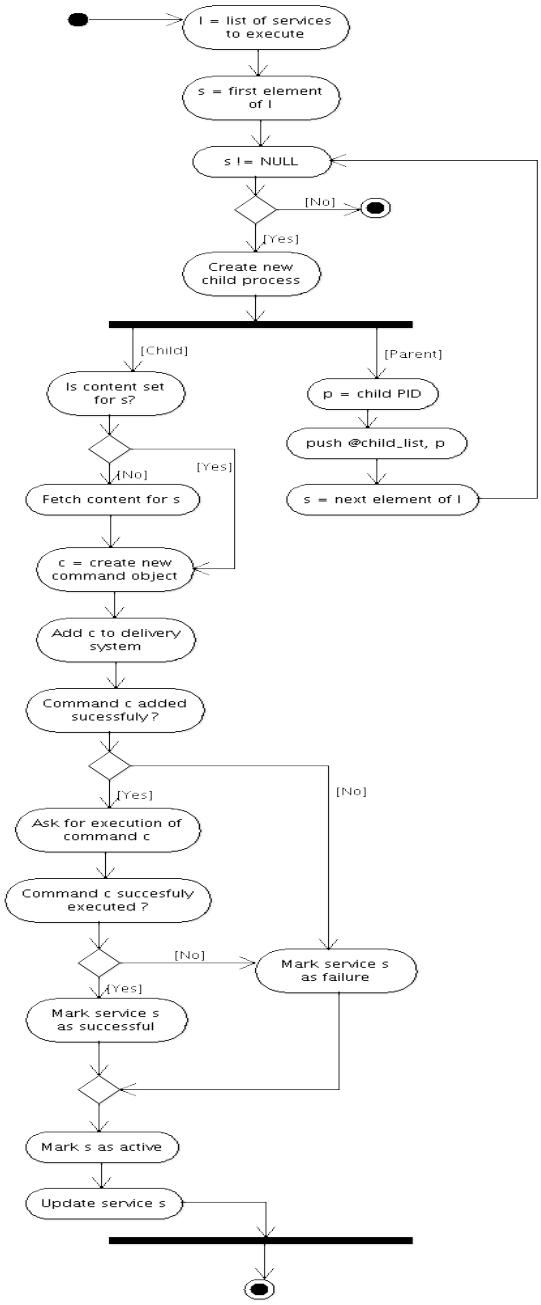

Figure 35: *Translation from Service into Command object, and consequent transmission to delivery system*

5.5.5 Command Execution Process

It is the delivery system the component responsible for sending the actual content to the end-user. It waits for requests from the service scheduler, in which this later component instructs for command addition or

execution. Upon request for execution, the delivery system analyses the command queue, and proceeds to command handling. To do so, it forks a new child process, and instructs this child to take care of command execution. After execution is finished, the child replies to the service scheduler, indicating the success or failure of the command. This process is depicted in Figure 36, but a more detailed description is necessary for understanding the command execution process.

After the child is forked, the first thing it does (besides signal redefinition), is to get an available Bluetooth interface. Depending on the type of command handler, the system may reserve more than one interface. Recalling Figure 20, we basically have two types of handlers: concrete handlers, like ObexPushHandler, and composite handlers. Composite handlers are implemented with the *Composite pattern* [GHJV 95, Martin 03], so that the details of using multiple dongles in simultaneous remain confined to a distinct class – the CompositeCommandHandler. This class will only be called when more than one Bluetooth interface is available, which is the case when the number of target devices of a command exceed the maximum simultaneous connections supported by one dongle.

Actual execution of the command always follows the same logic, as the composite handler implements the same abstract interface than the concrete handlers, allowing for algorithm sharing. When the interfaces are reserved, and the handlers created, the system forks new child processes, giving them the responsibility of delivering content to individual destinations. Each child therefore tries to deliver content, and reports back to the parent process the command execution status. The parent process waits until the completion of all spawned children, before sending the command status reply back to the service scheduler. This description is better comprehend with the help of Figure 37.

With this description we demonstrated the logic of the delivery system. A command is received from the service scheduler, and the delivery system creates the command handler necessary for execution. If multiple dongles are necessary for the execution of the command, then a composite handler is created. Finally, the delivery system sends a reply back to the service scheduler, indicating the success (or not) of the command, and consequently of the service.

The discussion of the method used for command execution brings to an end our discussion of the various particularities of the system. Throughout this section we discussed several relevant aspects of our system, demonstrating how configuration is achieved, how we daemonize the individual components, how we use the specification language, and how the service and command execution processes are executed. We now proceed to the demonstration of our system in real-world environments, as we present the results of our pilots and tests.

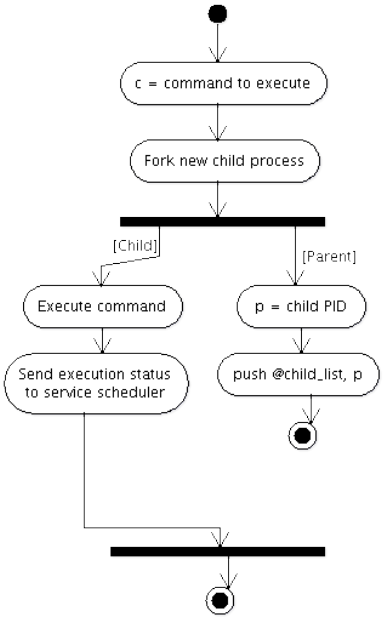

Figure 36: *Command handling process*

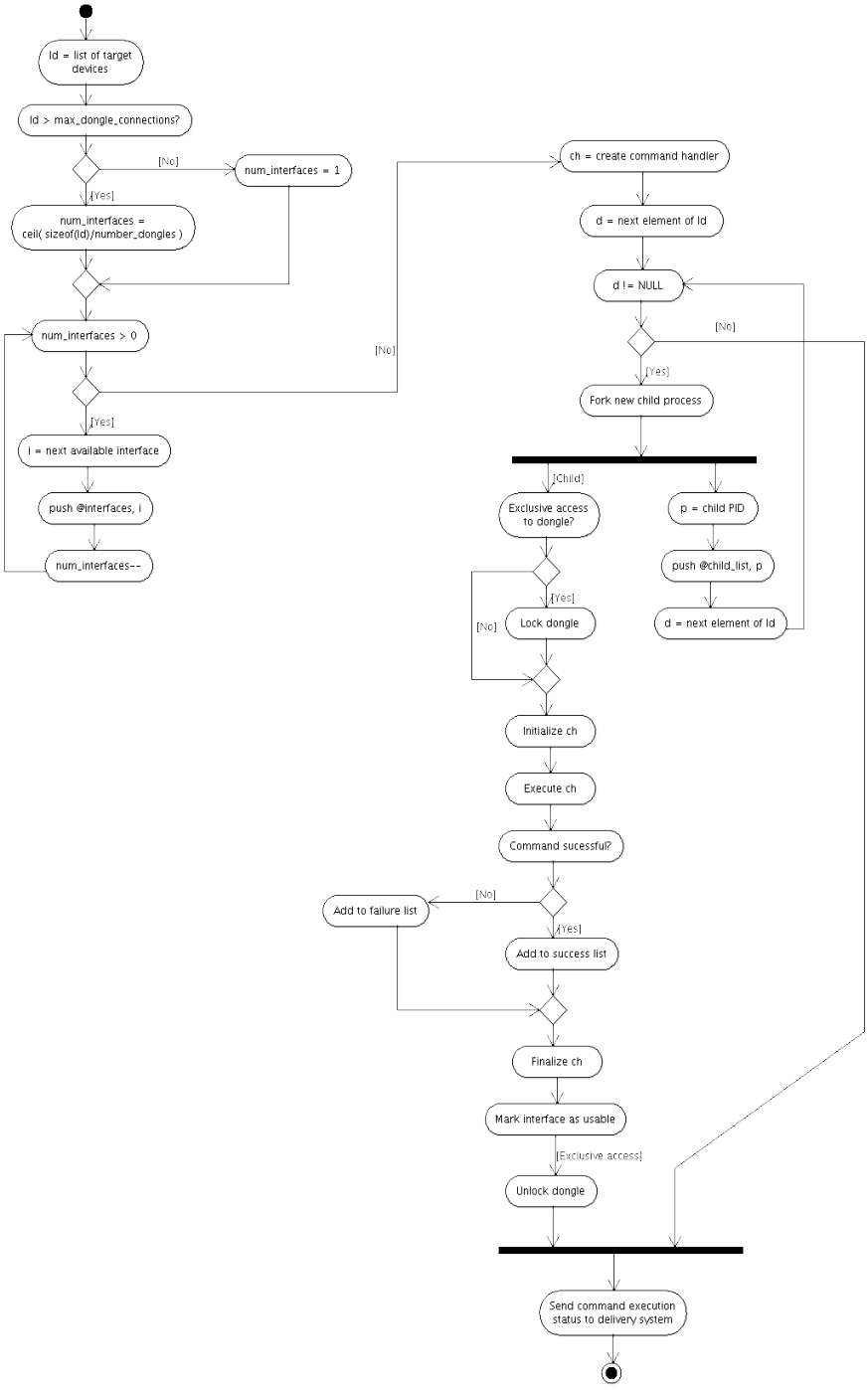

Figure 37: *Handler creation and content delivery to end-user*

6 Pilots & Test Results

One essential aspect when building software systems is to test them. This allows us to evaluate the system status, potentially finding problems and defects. Mainly, tests ensure that the system is working as expected and enables us to determine its effectiveness and efficiency.

Let's start by mentioning the fact that throughout this testing section we use a categorization for differentiating testing environments. As so, we define the *flow* of an environment to be either *static* or *dynamic*. The metric utilized to determine this is based on the data regarding the continuous time that devices remain in range of the scanner. In Figure 44 we have an example of a graph which uses 5 arbitrary classes to group devices' continuous time in range. We use the following formalization to determine the nature of the environment:

Let d be a device

Let $t(x)$ be a function which receives a device as input, and calculates the seconds which a device was continuously scanned

Let A be the bag of all devices d, where $0 \leq t(d) < 60$

Let N denote the bag of all devices such as $t(d) \in [0, +\infty[$

An environment E is said to be *dynamic* if and only if

$$0.55 \leq E \leq 1 : E = \frac{|A|}{|N|}, |N| \neq 0$$

An environment E is said to be *static* if and only if

$$0 \leq E < 0.55 : E = \frac{|A|}{|N|}, |N| \neq 0$$

We use another metric to further define the environment, but in terms of *affluence*. Three categories are used here: *low, medium*, and *high*. The following formalization is used to described the parameters used to determine the level of affluence:

Let d be the average number of devices seen in an environment, as defined by the following expression:

$$d = \left(\sum_{i=1}^{n} \sum_{j=0}^{23} \frac{x_{ij}}{n} \right) / 24$$

Where n is the number of days on which data was collected, and x the total number of devices seen in day i at hour range j.

An environment E is said to have a *low* affluence if and only if

$$0 \leq d \leq 10$$

An environment E is said to have a *medium* affluence if and only if

$$10 < d \leq 30$$

An environment E is said to have a *high* affluence if and only if

$$d > 30$$

6.1 Results

In this section we present the results obtained from the pilots. Early on the project we adopted a testing philosophy, and as such tests were executed constantly. We separate our pilots in four distinct scenarios: university, mobile, information center, and public transit ticket vending kiosk. As each of these scenarios presents unique characteristics, we believe that testing on each of these environments contributed to a better examination of the system.

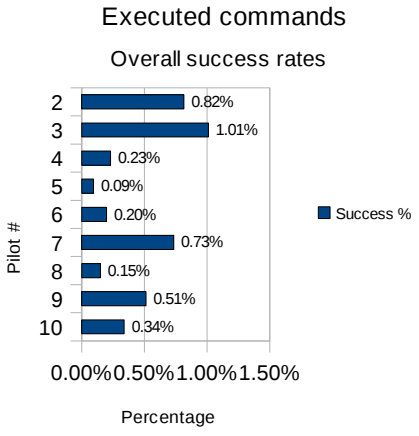

Figure 38: *Percentages of commands which delivered content to users*

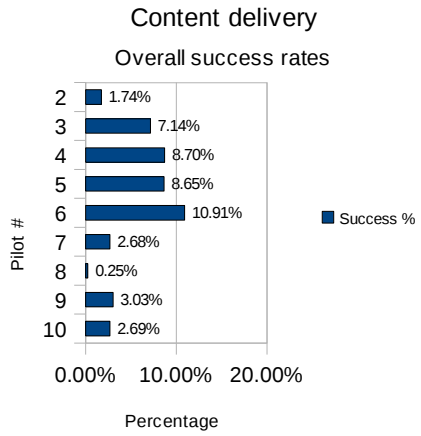

Figure 39: *Percentages of seen target devices to which content was successfully delivered*

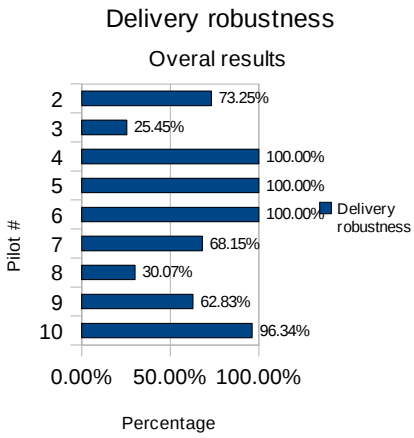

Figure 40: *Percentages of seen target devices to which content was tried to be delivered to*

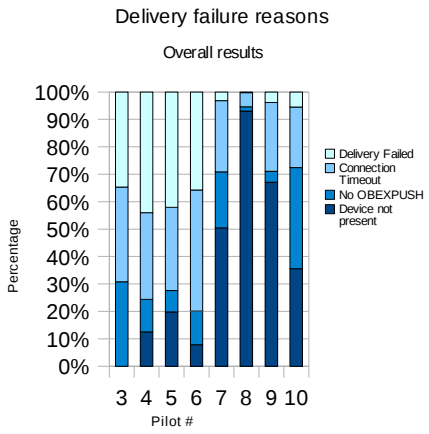

Figure 41: *Percentages comparison on the reason for content delivery failure*

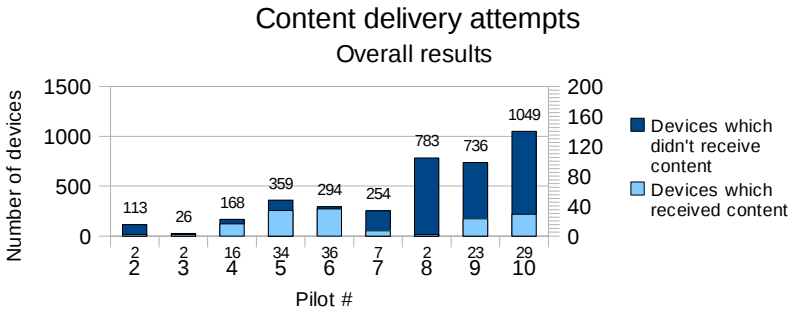

Figure 42: *Comparison between failed and successful delivery attempts*

6.1.1 University Pilots

We started our overall system testing early on the development stage. The computer was positioned at the local university near a entry/exit point (Figure 43). Overall six distinct pilots were made, contributing to problem identification, and consequently influencing several architectural decisions of our system.

Environment characterization is defined by looking at Figure 45. We verify that the average number of devices in the environment is of about 15 devices, and therefore we state that this is a medium affluence setting. Furthermore, Figure 44 indicates that around 40 % of all devices stay in range for less than a 1 minute. For that reason, and according to our pre-established

metric, this environment is termed as static. Given the characterization of the environment, we proceed to individual pilot description.

(a) Installation location

(b) Cafeteria near installation site

(c) Entry/Exit point

Figure 43: *University environment*

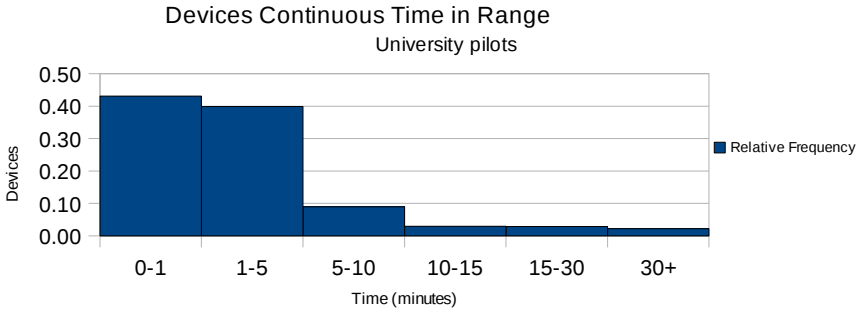

Figure 44: *Device continuous time in range for university pilots*

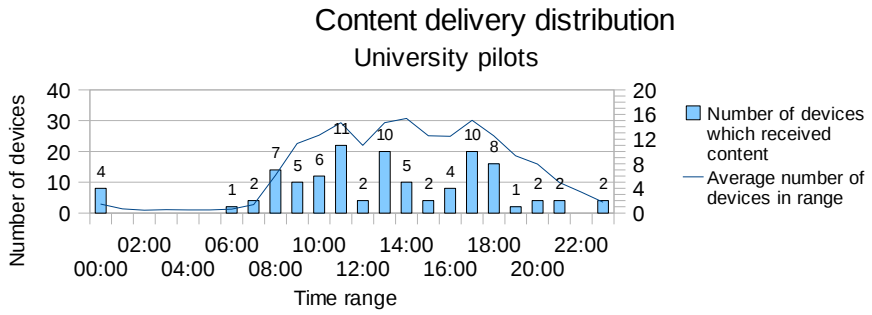

Figure 45: *University pilots average devices in range*

- **University pilots overall settings**
  - Limited functionality: The system was tested with only the blue station component working. The last pilot added the Broker component, but at a local level. No networking support was added during these trials, and no central infrastructure was used
  - Service to test: A service with textual static content. The service was set to delivery a simple “hello” text message. Delivery was filtered so that only mobile equipment and personal computers would receive content
  - Dongle configuration: Two dongles. One class 2 dongle for scanning and one class 1 dongle for content delivery
- **Pilot 1**
  - Concurrency mechanism: Multi-threading
  - Service and command execution: Many services, and consequently commands, can be run in simultaneous, not taking into account the available number of Bluetooth dongles
  - Scanning settings: 10.24 scan frequency and 5 seconds sleep between every scan
  - Multicast method: One thread responsible for creating a command, which would try and delivery content to all destinations sequentially.
  - Running time: One day

Pilot 1 was a unsuccessful test. The system ran only for a few hours before crashing due memory leaking problems. The collected data was pretty much inconclusive, and for that reason discarded.

- **Pilot 2**
  - Concurrency mechanism: Multi-threading
  - Service and command execution: Many services, and consequently commands, can be run in simultaneous, not taking into account the available number of Bluetooth dongles
  - Scanning settings: 10.24 scan frequency and 5 seconds sleep between every scan
  - Multicast method: One thread responsible for creating a command, which would try and delivery content to all destinations sequentially
  - **Running time**: Two days

Pilot 2 showed some improvements over the previous test. The running time improved, and allowed for useful data collection. Starting at 20/04/2009 around 13:00, and ending the next day at about 16:00 (Figure 46), pilot 2 allowed for useful data collection. Only 0.82% of commands were successful (Figure 38), and the content delivery success rate was of 1.74% (Figure 39). Delivery robustness rate was 73.25% (Figure 40), and content was delivered to 2 distinct devices (Figure 42).

The main issues verified during pilot execution related to serious memory leaking problems. This inevitably led to excessive memory consumption and consequent system crash. Additionally, a bug nullified our delivery failure report system. For that reason, the system couldn't collect delivery failure reasons.

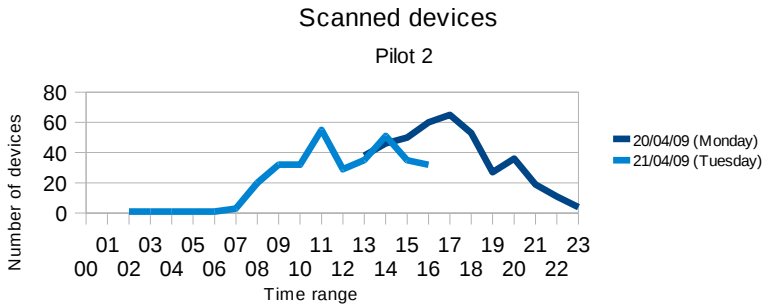

Figure 46: *Pilot 2 scanned devices*

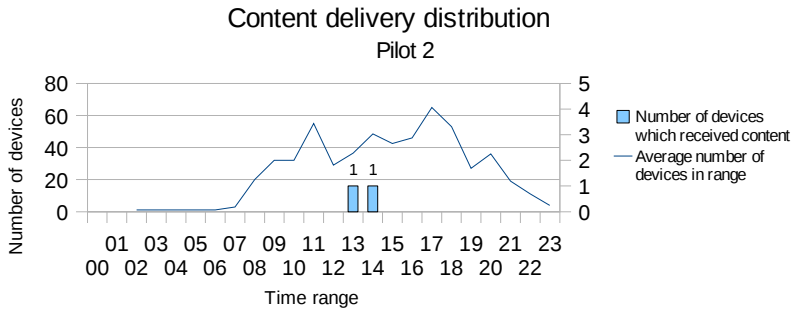

Figure 47: *Pilot 2 delivery distribution*

- **Pilot 3**
  - **Concurrency mechanism**: Multi-process
  - Service and command execution: Many services, and consequently commands, can be run in simultaneous, not taking into account the available number of Bluetooth dongles
  - **Scanning settings**: 10.24 scan frequency and 10 seconds sleep between every scan
  - Multicast method: One process responsible for delivering content to all destinations sequentially
  - Running time: 2 days

A significant architectural change was made in pilot 3. As we experienced unidentifiable memory leaking issues, we decided to switch from a multi-threading environment to a multi-process one. Naturally this involved a considerable amount of changes to the system, as several modules needed modification.

Running time of the pilot was again of two days. The pilot started around 17:00 of day 29/04/2009, and ended the next day around 16:00 (Figure 48). Command execution success rate was of 1.01% (Figure 38), and delivery success rate of 7.14% (Figure 39). Delivery robustness rate was of only 25.45% (Figure 40), and content was delivered to only 2 devices (Figure 42). As the delivery failure report system was operational, we verified that an even distribution between no OBEXPUSH port, connection timeout, and delivery failed existed for justifying delivery failure (Figure 41). Delivery failure for devices not being present wasn't still implemented at this time.

A serious problem encountered in this pilot was the occurrence of a deadlock which damaged the overall system functionality. Concretely, the delivery system was halted, and consequently the content dissemination infrastructure was made unavailable. As this moment the reasons for the occurrence of such issue couldn't be determined. We can verify the effect of this error on the system, as our delivery robustness rate was of only 25.45%.

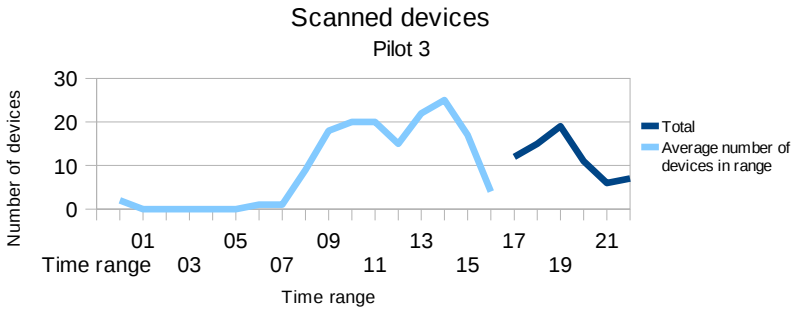

Figure 48: *Pilot 3 scanned devices*

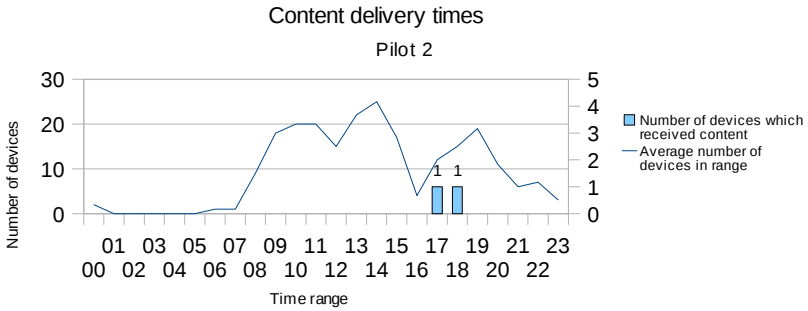

Figure 49: *Pilot 3 content delivery distribution*

- **Pilot 4**
  - Concurrency mechanism: Multi-process
  - Service and command execution: Many services, and consequently commands, can be run in simultaneous, not taking into account the available number of Bluetooth dongles
  - Scanning settings: 10.24 seconds for scanning and 10 seconds sleep between every scan
  - Multicast method: One process responsible for delivering content to all destinations sequentially
  - Running time: 4 days

In this pilot the system run without interruption for 4 days. Starting around 12:00 of day 08/05/2009, the system halted 4 days latter around 11:00 (Figure 50). Command success rate was of 0.23% (Figure 38), and content delivery success rate of 8.70% (Figure 39). The delivery robustness rate was of 100% (Figure 40), as the system executed 13 commands (Figure 51), delivering content to 16 distinct devices (Figure 42). Main reasons for delivery failure were associated with connections timeout, and rejection by the end user, as is seen in Figure 41.

The system was still negatively influenced by a deadlock situation. After further analysis, we came to the conclusion that the problem resided in OpenOBEX Perl binding libraries, which would raise a segmentation fault signal. This would kill off the child process responsible for command execution, and consequently the service scheduler would perpetually wait for a reply that would never arrive.

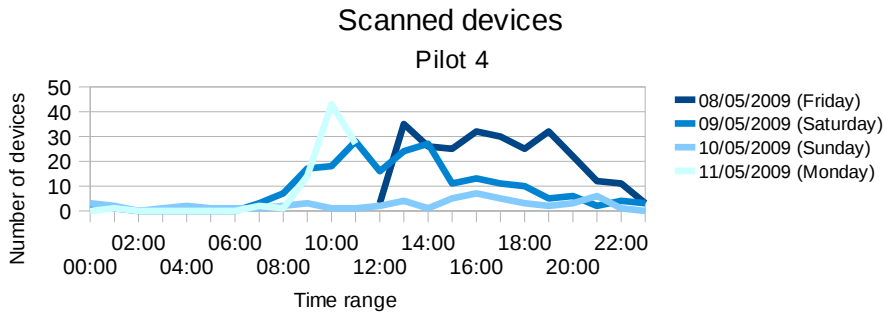

Figure 50: *Pilot 4 scanned devices*

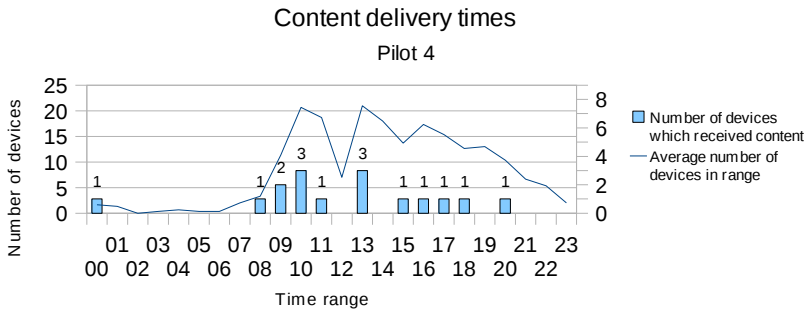

Figure 51: *Pilot 4 delivery distribution*

- **Pilot 5**
  - Concurrency mechanism: Multi-process
  - **Service and command execution**: The number of services, and consequently commands, that can be run in simultaneous is limited to the number of available dongles. If just one dongle is used for both scanning and delivery, or if just one dongle is used for delivery, then only one service can be executed at a time.
  - **Scanning settings**: 10.24 seconds for scanning and sleep for 3 seconds between every scan.
  - Multicast method: A process was responsible for delivering content to all devices sequentially.

First aspect to mention is that service and command execution methodology was changed in this pilot. We would now limit the number of services to execute in simultaneous to the number of available dongles. This would reduce the number of spawned processes, and alleviate the need for using synchronization mechanisms.

The system ran for 7 days uninterruptedly, starting at 19/05/2009 around 10:00, and ending around 12:00 at 25/05/2009 (Figure 52). Successful command execution rate was of 0.09% (Figure 38), and content delivery success rate of 8.69% (Figure 39). With a delivery robustness rate of 100% (Figure 40), the system executed 30 commands (Figure 53), delivering content to 34 distinct devices (Figure 42). Delivery failure reasons were mainly related with connections timing out (Figure 41).

As we diminished from 5 to 3 seconds the time that the scanner would sleep between scans, the device reader component started to complain that devices were constantly busy. The deadlock situation was resolved in this pilot, although segmentation faults were still getting raised due to OpenOBEX function bindings. Either way, the system maintained full functionality even when errors occurred, being only stopped by manual intervention.

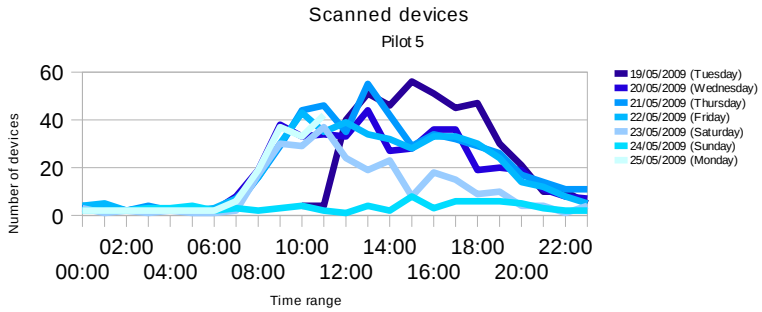

Figure 52: *Pilot 5 scanned devices*

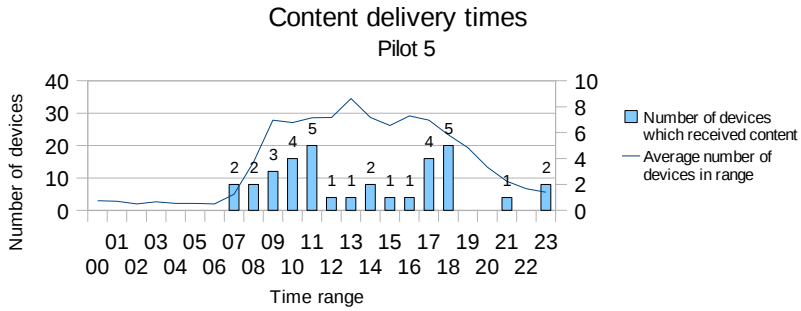

Figure 53: *Pilot 5 delivery distribution*

- **Pilot 6**
  - Concurrency mechanism: Multi-process
  - Service and command execution: Number of simultaneous services, and consequently commands, is limited to the number of available dongles.
  - **Scanning settings**: 10.24 seconds for scanning and 5 seconds interval between every scan
  - **Multicast method**: The multicast method was changed in this pilot. Instead of trying to deliver content, sequentially, to all present devices, the system will now use a delivery dongle to disseminate content solely to a single device. Multicast support is them achieved by using *N* delivery dongles, where each dongle would try and deliver content to a different device.
  - Broker component: In this pilot the intermediary component was introduced, althougth no networking was still available.

By changing the multicast method, our system would cease to deliver content to all nearby devices in a sequential manner. Now it would randomly choose a device from the list, and use a single dongle to deliver content to it. When the system was configured with just a dongle for scanning and delivery, content would be delivered to just one device at a time. This would be also the case when only a dongle would exist for delivery. On the other hand, if we had several dongles for delivery, then the system would try and disseminate content simultaneously.

The pilot started at 01/06/2009 around 12:00, and ended 5 days latter around 11:00 (Figure 54). The command success rate was of 0.20% (Figure 38), and delivery success rate of 10.91% (Figure 39). Delivery robustness rate was of 100% (Figure 40), as the system delivered content to 36 distinct devices (Figure 42), by executing the same number of commands (Figure 55). Failure on delivering content was mainly related to connection timeout and rejection by users, as is depicted in Figure 41.

Still remaining were the problems related to OpenOBEX library Perl bindings. Due to this, and to the fact that these function bindings don't allow choosing the dongle to use, we decided to utilize OpenOBEX external tools. Future versions of the system would therefore cease to use direct function calls to perform dissemination, and instead would make use of *obexput* binary. Also, the introduction of the Broker component – although at a local level – didn't harness the overall system functionality. The next pilots would be tested in a distributed way, with the Broker, central infrastructure and blue stations all fully functional and separated physically from each other.

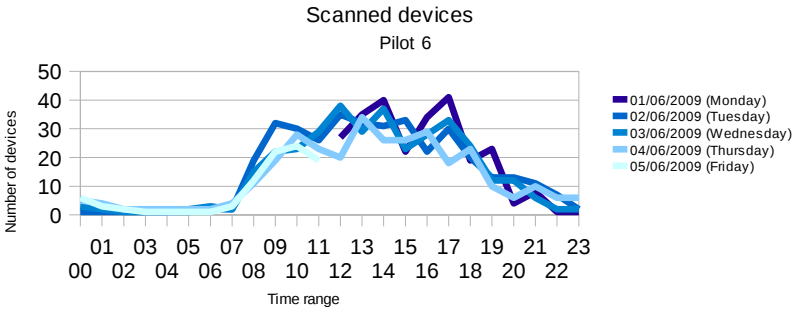

Figure 54: *Pilot 6 scanned devices*

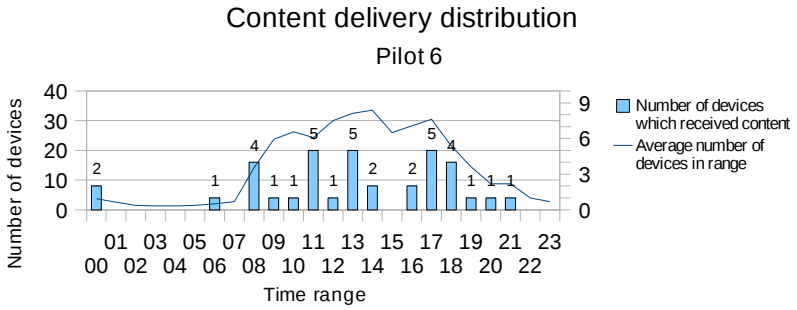

Figure 55: *Pilot 6 delivery distribution*

6.1.2 Mobile Pilot

At the time of the execution of this pilot, all components were already implemented, and network available. The pilot was made at two different festivals, separated apart by one day. The computer was set in a backpack and for that reason we termed this pilot as mobile. Since it was in the backpack, no electricity was available, which lead to battery consumption, and consequently to small up-time. Nevertheless, this allowed us to test the system in highly concentrated and dynamic settings.

As demonstrated by Figure 56, we are in presence of a dynamic environment, since more than 60% of devices remained in range for less than 1 minute. Also, we can term the settings where the pilot was executed as being of high affluence, since we have an average of more than 50 devices (Figure 57).

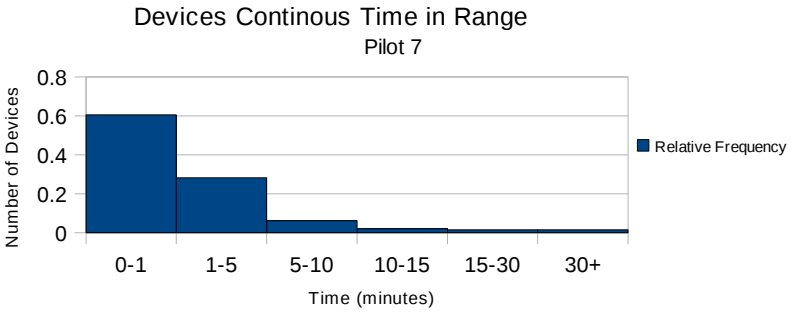

Figure 56: *Mobile pilot devices continuous time in range*

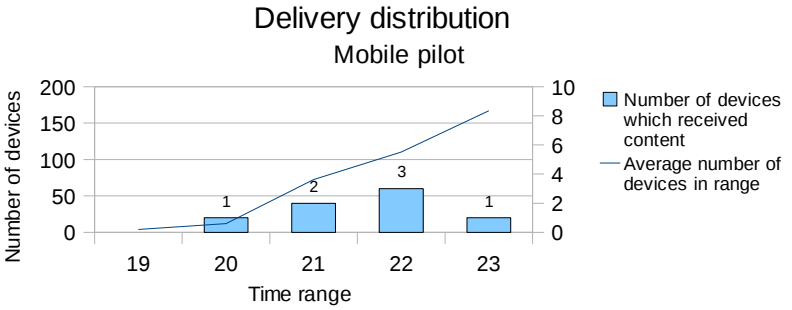

Figure 57: *Mobile pilot delivery distribution*

- **Pilot settings**
  - **System functionality**: All components functional. Both the Broker and central infrastructure were fully deployed, and networking was available through a 3G connection
  - Service to test: A service with textual static content
  - Dongle configuration: Two dongles. One class 2 dongle for scanning and one class 1 dongle for content delivery
  - Concurrency mechanism: Multi-process
  - Service and command execution: Number of simultaneous services, and consequently commands, restricted to the number of available dongles
  - Scanning settings: 10.24 seconds for scanning and 5 seconds sleep between scans
  - Multicast method: A dongle could only deliver content to one device at a time
  - Running time: 2 days non-continously

Due the nature of the testing, this pilot was the only one which was supposed to be run non-continuously. During two days, the computer was carried around, capturing and disseminating information in a backpack. This lead to natural high battery consumption, and as such run-time was low in both days.

The pilot started at 31/07/2009 around 19:00, and ended the next day at about 23:00 (Figure 58). With a command execution rate of 0.73% (Figure 38), the system delivery success rate was of 2.68% (Figure 39). The delivery robustness percentage was of 68.15% (Figure 40), as we delivered content to 7 distinct devices (Figure 42) through the execution of the same number of commands (Figure 57). Also, the main reason for delivery failure was related to devices not being present in the moment of delivery (Figure 41).

Although delivery success rates weren't too high, the pilot did allow us to discover and resolve several issues regarding the Broker and central infrastructure components. Also, executing this pilot gave us the chance to test station remote management, and how all components behaved when working in a cooperative and distributed way. Furthermore, this was an opportunity to analyse the behaviour of the system on a more crowded and dynamic environment than the previous pilots.

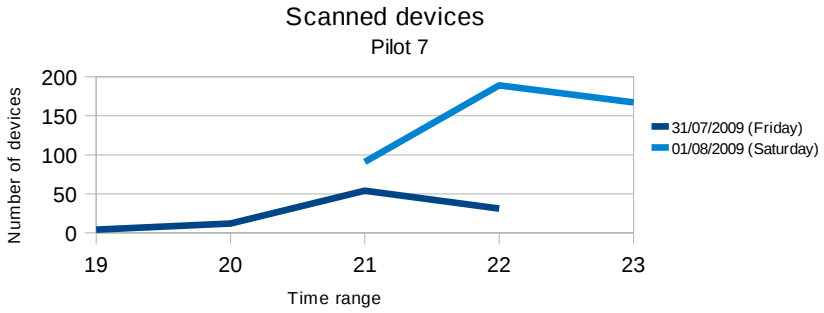

Figure 58: *Pilot 7 scanned devices*

6.1.3 Information Center Pilot

After completing our system structuring and initial tests, we proceeded to test the system in another environment.. We deployed the computer in a tourist information center, a point of high people flow (Figure 59).

The highly dynamic nature of the environment is observed by looking at Figure 60, where we verify that almost 90% of scanned devices spent less than 1 minute in range of the scanner. High people affluence also characterizes this environment, as it is seen in Figure 61, where we have an average of about 100 devices.

Contrary to previous tests, in this pilot we decided to set up the system to fetch dynamic content. We therefore created a news service, which is simply a service which fetched content from a PHP web page, situated in one of our servers, and disseminated the content to end-users.

(a) Installation location

(b) View from outside

(c) Another outside perspective

Figure 59: *Info center environment*

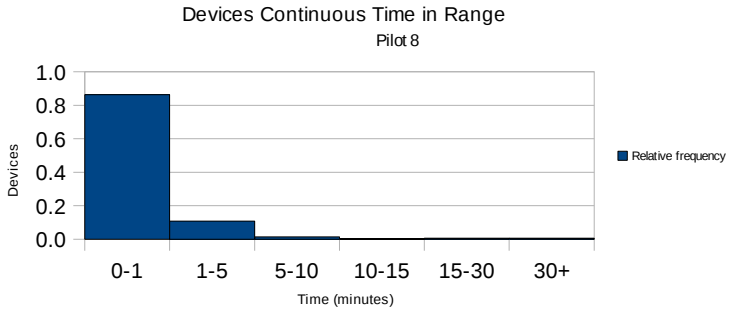

Figure 60: *Pilot 8 devices continuos time in range*

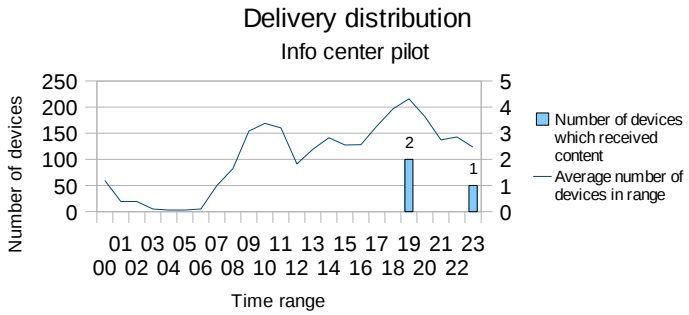

Figure 61: *Info center content delivery distribution*

- **Pilot Settings**
  - System functionality: All components were functional
  - Service to test: A service with dynamic content. The system would retrieve content from a web page before disseminating it to users. Also, the service would only be delivered to mobile equipment
  - Dongle configuration: 2 class 1 dongles. One for scanning and other for delivery. We decided to try and boost the scanning range, since the surroundings of the information center was surrounded by thick brick walls, damaging the signal range
  - Concurrency mechanism: Multi-process
  - Service and command execution: Number of simultaneous services and commands restricted to the number of available dongles
  - Scanning settings: 10.24 seconds for scanning and 5 seconds sleep between scans
  - Multicast method: A dongle could only deliver content to a device at a time

- Running-time: 3 days (with some interruptions)

The pilot started at 18/08/2009 around 20:00, ending 2 days latter around 23:00, as is observed in Figure 62. Successful command execution rate was of 0.15% (Figure 38), and successful delivery rate of only 0.25% (Figure 39). Delivery robustness rate was of about 30% (Figure 40), and content was delivered to 2 distinct devices (Figure 42), by means of 3 commands (Figure 61). Virtually all content delivery failure was associated with devices not being present, as is seen in Figure 41.

Overall this pilot was considered to be unsuccessful. First we had a power outrage problem which influenced the overall results. Looking at Figure 62 we verify a discontinuousness on the last of the pilot, as power failed making the power run on battery, and consequently shut down. Also, an error on the process of remote content retrieval led to further performance degrade , as this error made the dissemination infrastructure unusable during much of the pilot's execution time.

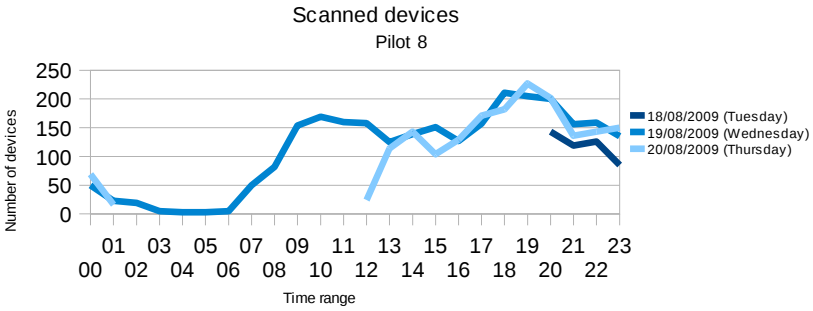

Figure 62: *Pilot 8 scanned devices*

6.1.4 Public Transit Infrastructure Pilots

Our last two pilots were made at a public transit infrastructure. We deployed the system at an automatic ticket vending kiosk, situated near several bus stops and a café (Figure 63). The type of content to deliver was the same as in the previous pilot, where a news service was created which fetched dynamic content from a web page. We placed the computer inside the kiosk's celling, but this wasn't the preferred scenario as the celling was surrounded by metal, which contributes to interference of radio signals.

The number of scanned devices which remained in range for less than 1 minute was of 55% (Figure 64). For that reason the environment is considered to be dynamic although marginally. Figure 65 demonstrates that the average number of devices in range was about 15, and for that reason the environment is characterized as a medium affluence one. Let's now proceed to individual pilot description.

(a) Installation location

(b) Nearby bus stops

(c) Outside perspective

Figure 63: *Public transit infrastructure environment*

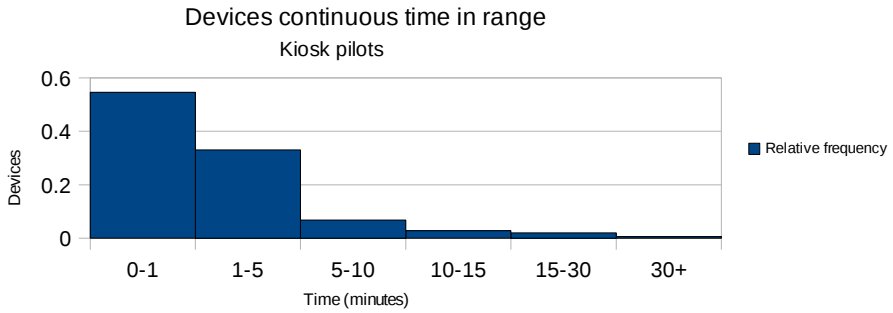

Figure 64: *Public transit kiosk devices continuous time in range*

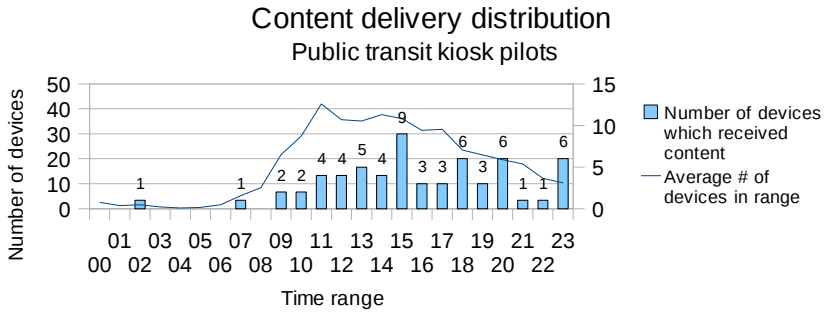

Figure 65: *Content delivery distribution for public transit kiosk pilots*

- **Overall kiosk pilots settings**
  - System functionality: All components were functional
  - Service to test: A service with dynamic content. The system would retrieve content from a web page before disseminating it to users. Also, the service would only be delivered to mobile equipment
  - Dongle configuration: 2 class 1 dongles. One for scanning and other for delivery. This was an attempt to boost the Bluetooth radio signal, due the presence of metal in the kiosk
  - Concurrency mechanism: Multi-process
  - Service and command execution: Number of simultaneous services and commands restricted to the number of available dongles
  - Scanning settings: 10.24 seconds for scanning and 5 seconds sleep between scans
- **Pilot 9**

- Multicast method: A dongle could only deliver content to a device at a time
- Running-time: 5 days

The pilot started at 25/08/2009 around 11:00, ending 5 days latter at about 23:00 (Figure 66). With a command execution rate of 0.51% (Figure 39), the system managed to deliver content to about 3% of the target group (Figure 67). Concretely, this means we had 27 successful deliveries (Figure 67) to 23 distinct devices (Figure 42). The dynamic labelling of the environment seems justified, as we verify in Figure 41 that the major reason for delivery failure was related to devices not being in range. Also, almost 37% of the target devices weren't even contacted, as is seen by our delivery robustness rate of about 63% depicted in Figure 40.

During the pilot some issues regarding remote content retrieval still remained. For this reason, overall performance was negatively influenced. After analysis we came to the conclusion that the problem was related to excessive number of opened files. The system creates temporary files so it can store the remote content it fetches. An anomaly in the system was leaving file handles open, and consequently the operating system would eventually complain that an excessive number of files were being maintained opened by the process. Although not harnessing the scanning process, this would nullify the dissemination infrastructure. During the pilot we were already aware of this, and so steps to minimize the effects were taken.

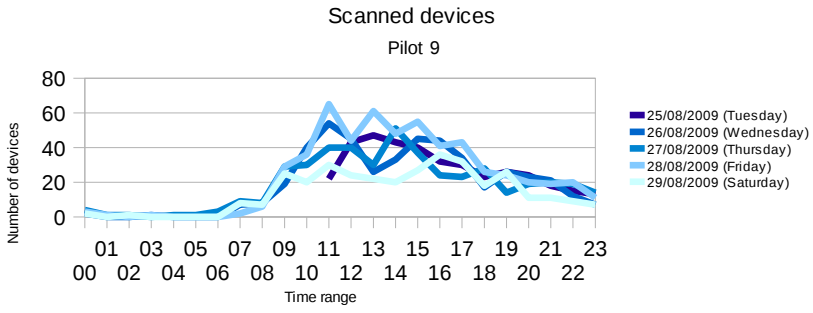

Figure 66: *Pilot 9 scanned devices*

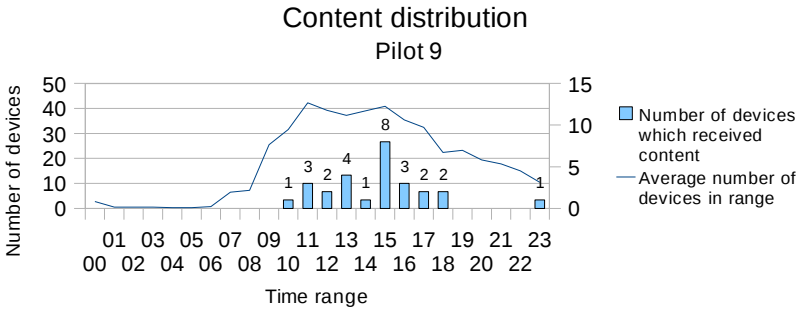

Figure 67: *Pilot 9 content distribution*

- **Pilot 10**
  - **Multicast method**: A dongle could deliver content to several devices at a time, being the maximum 7. By default we set this option to allow for 4 simultaneous connections to be established by a dongle
  - Running-time: 5 days

Pilot 10 was the last of our tests. Comparing to the previous pilot, the only relevant change was on the way that multicast commands were executed. Instead of limiting a dongle to deliver content to a device at a time, we introduced a configurable option that lets the administrator choose to deliver content up to 7 devices in simultaneous.

Running time was of 5 days. The pilot started at 30/08/2009 around 00:00, and ended 03/09/2009 around 22:00 (Figure 68). Successful command execution rate was of 0.34% (Figure 38). With a value of 2.69% (Figure 39), the content delivery success rate was also lower than the previous pilot. Nevertheless, the system managed to diminished considerably the delivery failure reason due devices not being present (Figure 41). Also, and comparing to the previous pilot, we obtained an increase of more than 30% on the delivery robustness rate (Figure 40). Furthermore, by looking at Figure 69 we verify that a total of 34 delivery hits were successful. These 34 hits consequently translated into content delivered to 29 distinct devices, as is depicted in Figure 42.

This last pilot allowed us to confirm that all identified major issues until date were resolved. Moreover, it was the only pilot in which we tested content dissemination using simultaneous connections. Overall the pilot was

successful, as no problems were identified, and the system behaved as expected.

Now that the data regarding the pilots was presented, it is now time to proceed to the discussion of obtained results, and consequent presentation of the withdrawn conclusions.

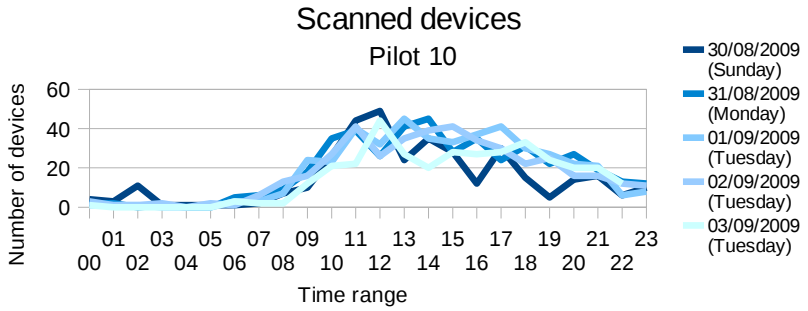

Figure 68: *Pilot 10 scanned devices*

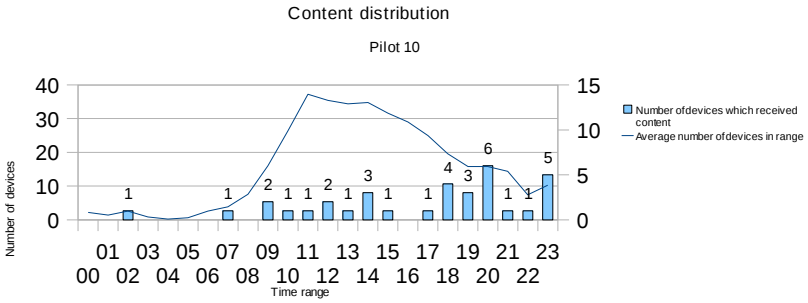

Figure 69: *Pilot 10 content distribution*

6.2 Discussion

In this section we'll be presenting the conclusions which we came to by way of analysis of the content presented in the previous section. The objective here is to show how the execution of the pilots helped us identify problems, and how it allowed us to make informed modifications to the system.

6.2.1 University Pilots

University pilots were the first to be executed and, naturally, allowed us to identify several issues. Many aspects of the system were influenced by analysis of these pilots, including architectural decisions.

The first thing to note is that we decided to switch from a multi-threading environment to a multi-process one. Mainly, this was due to memory leaking issues, whose origins couldn't be determined. The result was a more stable and consistent system. There were some concerns regarding the overall performance and memory consumption in using multi processes instead of threads, but we concluded that both resource consumption and system speed weren't visibly undermined by this switch, which in part is explained by the fact that modern UNIX and UNIX-like systems make use of *copy-on-write* techniques to spawn new processes, as described in [Bach 86, WCO 00].

It was also determined that setting the frequency between scans to a low value may lead to excessive interference and problems on the scanning process. This was what happened in pilot 5, where we set the frequency between scans to only 3 seconds, which lead to constant complains by the system. Either way, throughout our pilots we couldn't see any noticeable difference between the number of scanned devices either we used 3, 5, or 10 seconds between scans. Also, it should be taken into account that setting this value to a too low of a value will contribute to a more rapid device battery consumption.

More importantly is the fact that in a very similar configuration, we obtained an increase of content delivery success rate by modifying the way multicast commands were executed. Comparing pilots 5 and 6 in Figure 39, we verify that an increase of more than 2% in the delivery success rate happened. Furthermore, by looking at Figure 41 it is seen that delivery failure rate due devices not being present was diminished in pilot 6. Although both delivery robustness rates for pilot 5 and 6 were 100% (Figure 40), this value is misleading for pilot 5 as it worked sequentially, and therefore by the time the system would try to deliver content to a device, the device would probably be already out of range. In sum, it seems clear that performing multicast using a dongle for delivering content a time, in a static environment, is superior than trying to deliver content to all devices sequentially.

Another aspect to mention is that it seems that there is a tendency for delivering more content during times of greater affluence. By looking at Figure 45 we verify that a higher number of success delivery hits happened during peak times. This is a expected result in a static environment, since it seems

natural that there is a higher change of delivering content when a greater number of devices is present.

6.2.2 Mobile Pilot

The mobile pilot was executed in a more dynamic environment than the previous pilots. Basically, this pilot helped us understand that probably system efficiency would be heavily influenced by the nature of the environment. The high people affluence and mobility in the pilot's environment made the system behave below expected. This is substantiated by the delivery robustness rate of 68.15%, as is seen in Figure 40. Content was only delivered to 7 distinct devices out of 254 (Figure 42), and the fact that more than 50% of delivery failures were due devices not being present, made evident that using a dongle to deliver content to a device at a time, in dynamic environments, would probably lead to impoverished results.

As content distribution goes, by looking at Figure 57 it seems that there is a proportional tendency for successfully delivering content as the number of devices in range raises. This has already been verified in the university pilots, but the lack of data depth on this pilot doesn't allow us to further substantiate this claim.

6.2.3 Information Center Pilot

Deploying the system at the information center allowed us to test it at a very dynamic environment. With more than 90% of devices staying in range for less than 1 minute, opportunities to successfully establish connections, and disseminate content, were diminished. If doubts existed on the possible performance undermining of the system in dynamic environments, they were dissipated with the execution of this pilot. With a delivery robustness rate of only 30% (Figure 40), and a rate of delivery failure of more than 90% due devices being out of range (Figure 41), we verified that, under dynamic environments, trying to deliver content to a device at a time leads to poor results.

Interesting is the fact that although content delivery successful values are very small, the majority of the delivery hits happened during peak time (Figure 61), which comes to reinforce previous statements.

6.2.4 Public Transit Infrastructure Pilots

Being considered a dynamic environment, with 55% of devices staying in range for less than 1 minute, the kiosk pilots allowed us to test the use of simultaneous content dissemination in a dynamic setting. Pilot 9 was still executed with a dongle limited to delivery content to a device at a time, while pilot 10 used up to 4 simultaneous connections.

Comparing delivery robustness between these two pilots, we verify an increase of about 34% between pilot 9 and 10 (Figure 40). This, allied with a decrease of more than 30% on delivery failure due devices which were out of range (Figure 41), leads to the conclusion that much more devices were tried to be contacted in pilot 10. Although this didn't translate into a substantial increase on the number of delivery hits (Figure 69), or served devices (Figure 42), it seems that using simultaneous connections in dynamic environment leads to better results. Further data is necessary to substantiate this claim, as it is interesting to verify if results are influenced by the increase of allowed simultaneous connections.

After finishing our pilots we executed some interviews in order to gain a user's perspective of the system. The objective was to collect replies from users in order to identify possible flaws and areas on which improvements could be made.

6.3 Users Opinions

Due the nature of our application, gathering replies in a real-world test environment is difficult. It becomes almost impossible to detect and interview people how choose to accept the content disseminated by our system.

In an attempt to collect additional information about the system, from a user's point of view, we executed several small qualitative interviews. The service to test was a news headline dissemination service, similar to that used in the pilots tests. Content was fetched from a remote web page, and 10 different news headlines were delivered to users using OBEXPUSH. We asked 6 subjects to enable their devices' Bluetooth, and accept two consecutive messages from our system. As stated previously, the system will only resend dynamic content to an already served device, only when the content changes.

The interview was semi-structured and composed by open-ended questions, described as follows:

- What do you think of receiving content while waiting, through Bluetooth? Was the content of any interest to you?
- Did you have to wait long before receiving the content?
- What about the content's presentation? Were you happy to receive plain text, or would you prefer something else?
- What did you think of the refresh rate between the two consecutive messages?
- After you received the content, was is it easily accessible?

Overall, subjects thought the service as useful. Some subjects stated that as the service is free-of-charge, and accessible without the need for additional software, is very advantageous. More related to public transit, some subjects referred that it would be interesting to implement a service that warned public transit users when a bus was running late. Also, speed of dissemination was considered to be very good, although this would probably decline in more crowded environments.

On the other hand, subjects stated that the name of our dissemination dongle – *Mobile*, which appears on their mobile equipment screen, is of great importance on the decision of whether to accept content. Some even said that they “would disable Bluetooth” if the name didn't somehow transmit confidence to them. Also, subjects thought that, although the content disseminated was interesting, personalization of the content to receive was important, as “people have different preferences”.

Content presentation did divide opinions. Some subjects felt that receiving text was appropriate, as they could easily read it. Others thought textual content was advantageous, as “text is better for compatibility”. Some people also mentioned that adding a small message at the bottom of the disseminated content would contribute to a more pleasing service.

On the contrary, one participant, that owned a modern touch-screen mobile equipment, had difficulties viewing the textual content, as he had to perform zoom operations to clearly identify what was written. Others felt that textual content was admissible, but that it would be “more appealing” to

present it as an image. The majority of the subjects also thought that the system should adapt its content presentation type in accordance to devices' capabilities.

Overall subjects thought that the refresh rate between two consecutive messages was too fast. A subject stated that the disseminated content was too “repetitive, being very difficult to understand the changes”. Others thought that time restrictions should be imposed, and still others thought that the content should only be resent when more than one headline changes. A subject also stated that users should have the opportunity of “personalizing the refresh rate, according to the type of service”.

As for content access, overall people could easily view it, as the equipment would ask the user to open the content upon reception, as demonstrated in Figure 70. Still, two subjects experienced difficulties in finding the content, as their equipment would receive the content and store it without asking the user if he wanted to view it. This made users search for the content throughout the mobile equipment, as the location where it was stored was unclear.

Given the opinions of our test subjects, we proceed to the presentation of the implications that the pilot tests and qualitative interviews had in our present and future visioning of the system.

(a) Phone asks permission from user

(b) User has immediate access in some equipments

(c) One of the news headlines received by the user

Figure 70: *Example content acceptance and access using a Nokia mobile phone*

6.4 Implications

Here we present our inferred conclusions with regard to the pilot results and the replies collected from the qualitative interviews. Pilots directly influenced some development aspects of the system, as they were executed when the prototype was still in active development. The qualitative interviews were made posteriorly to system development, and as such the withdrawn conclusions will refer to possible future modifications.

The first thing to state is that the use of multiple simultaneous connections in dynamic environments seems to lead to better dissemination results. As we use various connections in parallel, the number of contacted

devices raises, increasing the probability of successful content delivery. Also, it seems that using multiple connections in static environments may lead to insignificant differences on content delivery. This is substantiated by looking at Figure 40 and Figure 41, where we verify that pilot 6 had a delivery robustness rate of 100%, and a small delivery failure rate due devices being out of range, although the delivery dongle was restricted to deliver content to a single device in each dissemination attempt.

The name we choose for the delivery dongle(s) also seems important. In our tests we choose to use the name *Mobile*, which really didn't transmit too much confidence to users, as some of them only accepted the content after we reassured them that it was our system disseminating the data. This is comprehensible if we relate back to the questionnaire distributed to public transit users, where about 35% of respondents stated that security was the reason for disabling Bluetooth (Table 3).

Furthermore, both content and presentation personalization seem important in convincing user adherence to the system. Some equipment lack support for more graphical content, while more modern equipment are better dealing with images. Also, people interests are distinct, and as such the disseminated content should be chosen by them. This indicates that a registration process should be performed, in which the user sets preferences such as type of service, type of equipment, refresh rate, etc.

Also, the method currently used for determining if dynamic content has been modified is inappropriate for all situations. As we use a MD5 hash function to determine if content has changed, this isn't desirable for all types of services. For example, if we use this method on a news service, users can receive very similar content, where the only difference may be a single news headline. On the other hand, this method is useful if we want, for example, to create a service that notifies users of changes in public transit bus schedules. Overall, it seems that the system has to support several methods for determining content modification, as the use of hash functions isn't suitable for all services.

With these implications we finalize the presentation of pilots and tests results. Along this section we showed the data collected by our pilots, and

presented the conclusions inferred by them. We now proceed to conclusive commentaries and the indication of possible future work.

7 Conclusion & Future Work

In this section we present our conclusive comments and possible future work orientation. We summarize the advantageous of using our system for both proximity sensing and content dissemination, and point out how the system may be improved in the future with the support for additional features.

7.1 Conclusive Comments

In this work we presented a prototype system that enables for proximity sensing and context-aware service offering. We defended that our system is a viable alternative to classical methods of information capture and content dissemination, as it offers a set of advantages over these.

We demonstrated that our system is suitable for proximity sensing, as it uses a scanning element for capture of Bluetooth enabled devices within the vicinity. We showed that the components responsible for scanning may be installed at many different settings, collecting data that is useful for several types of applications such as O/D matrices derivation. We also demonstrated that the Bluetooth data is useful for defining the environment on which the component is installed, as contextual data regarding flow and affluence are derived from analysis.

Furthermore, our system implements an infrastructure for the dissemination of context-aware services. With the usage of our *Service Specification Language* (SSL), we enable the creation of a rich set of services that disseminate both static and dynamic content. With the implementation of our system we lower the need for specialized dissemination components, as presentation is delegated to users' equipment. Also, we make our dissemination infrastructure flexible, as we remove hardware dependency. Finally, we make content and presentation personalization possible, a feature that has been demonstrated important to users.

7.2 Future Work

Our current infrastructure may be extended in several ways. First of all, our plans are to extend the system to support user registration and service creation. Thereafter, we plan to expand the system to take into account equipment type, as content format should be adapted accordingly.

Moreover, it is our intention to extend the current push-only dissemination system to an integrated content dissemination system. We would provide users with the possibility of communication initiation, where they would send requests to the stations using a pull back-channel. This would make *ad-hoc* registration and on-demand content dissemination possible. Also, our dissemination infrastructure current method of functionality only supports the use of a dongle per service. This means that if the system has only one delivery dongle, then it is bound to execute services sequentially, although it can deliver to multiple destinations simultaneously. An interesting addition would be to permit the usage of a dongle by distinct simultaneous services.

Additionally, the system may be extended to collect concrete localization information. With the use of Bluetooth radio signal strength, it is possible to calculate devices' relative distance. Also, with the use of triangulation concrete localization is possible, therefore adding new contextual dimensions to our system, enriching our specification language.

In conclusion, we refer that throughout our work security has not been a concern. For this reason, it is pertinent to state that attacks are made possible due the lack of implemented security mechanisms. If we use a direct connection between stations and the broker, for example, an attacker may easily eavesdrop the content. Also, we only use a simple hash code for recognition of stations, a method far from secure. These examples demonstrate that security measures should be implemented in future versions of our system.

Bibliography

[Kostakos 08] Kostakos, V.. *Using Bluetooth to Capture Trips on Public Transport Buses*. LabUSE, University of Madeira (2008) : .

[TC 03] Thapa, K. and Case, S. An Indoor Positioning Service for Bluetooth Ad Hoc Networks. In: Midewest Instruction and Computing Symposium. Duluth, MN, USA. 2003: .

[BCG 05] Bruno, R., Conti, M. and Gregori, E. *Mesh Networks: Commodity Multihop Ad Hoc Networks*. IEEE Communications Magazine (2005) Vol. 43 Num. 3: 123-131.

[SF 03] Shalaby, A. and Farhan, A. Bus Travel Time Prediction Model for Dynamic Operations Control and Passenger Information Systems. In: TRB 82nd Annual Meeting. . 2003: .

[OKKSPFJ 06] O'Neill, E., Kostakos V., Kindberg, T., Schieck, A., Penn A., Fraser, D., and Jones, T.. Instrumenting the City: Developing Methods for Observing and Understanding the Digital Cityscape. In: UbiComp 2006: 8th Int'l Conf. on Ubiquitous Computing. . 2006: 315-332.

[AGKO 04] Aalto, L., Gothlin, N., Korthonen, J. and Ojala, T.. Bluetooth and WAP Push Based Location-Aware Mobile Advertising System. In: Proc. of the 2nd International Conference on Mobile Systems, applications and services. . 2004: 49 - 58.

[PHJ 02] Podnar, J., Hauswrith, M., Jazayeri, M.. Mobile Push: Delivering Content to Mobile Users. In: 22nd International Conference on Distributed Computing Systems Workshop. Vienna, Austria. 2002: 563 - 570.

[VK 07] Vishik, C. and Kartha, G. *Intelligent RFID Information Management System*. US Patent App. 11/855,478, Google Patents (2007) : .

[Norris 02] Norris, A. Current Trends and Challenges in Health Informatics. In: iSHIMR: Proceedings of the 7th International Symposium on Health Information Management Research. Sheffield. 2002: 2-15.

[TO 00] Tan, K. and Ooi, B.. *Data Dissemination in Wireless Computing Environments (Advances in Database Systems)* . Springer. July 2000.

Bibliography

[AFZ 97] Acharya, S., Franklin, M. and Zdonik, S. Balacing Push and Pull for Data Broadcast. In: Proceedings of the ACM SIGMOD Conference on Management of Data. Tucson, AZ. May 1997: 183-194.

[AFZ 95] Acharya, S., Franklin, M and Zdonik, S. *Dissemination-Based Data Delivery Using Broadcast Disks*. IEEE Personal Communications (December, 1995) : 50-60.

[FJMFC 01] Figueiredo, L., Jesus, I., Machado, J., Ferreira, J. and Carvalho, M. Towards the Development of Intelligent Transportation Systems. In: Proc. 4th IEEE Intelligent Transportation Systems. Oakland, CA. January 2001: 1206-1211.

[Shibata 99] Shibata, J. *Progress in Intelligent Transportation Systems in the US, Europe and Japan*. ISATA Magazine (1999) : 27-29.

[Zhao 00] Zhao, Y. *Mobile Phone Location Determination and Its Impact on Intelligent Transportation Systems*. IEEE Transactions on Intelligent Transportation Systems (March 2000) Vol. 1 Num. 1: 55-67.

[LHHR 00] Levine, J., Hong, Q., Hug, G. and Rodriguez, D. *Impacts of an Advanced Public Transportation System - Demonstration Project*. Transportation Research Record: Journal of the Transportation Research Board (2000) Num. 1735: 169-177.

[MOJ 95] Morgan-Owen, J. and Johson, G. *Differential GPS Positioning*. Electronics & Communication Engineering Journal (February 1995) Vol. 7 : 11-21.

[MD 01] Maclean, M. and Dailey, D. Busview: a graphical transit information system. In: Intelligent Transportation Systems, Proceedings. 2001 IEEE. Oakland, CA, USA. 2001: 1073-1078.

[HQR 94] Hansen, M., Qureshi, M. and Rydzewski, D. *Improving Transit Performance With Advanced Public Transportation Systems Technologies*. Insitute of Transportation Studies (1994) : .

[Wilson 06] Wilson, N. *Public Transporation Service and Operations Planning Lecture Notes*. (2006) : .

[Ben-Akiva 87] Ben-Akiva, M. *Methods to Combine Different Data Sources and Estimate Origin-Destination Matrices*. Transportation and Traffic Theory, N.H. Gartner and N.H.M. Wilson (eds) (1987) : 459-481.

[Cui 06] Cui, A. *Bus Passenger Origin-Destination Matrix Estimation Using Automated Data Collection Systems*. Massachusetts Institute of Technology. September 2006. Master thesis

[ZRW 07] Zhao, J., Rahbee, A. and Wilson, N. *Estimating a Rail Passenger Trip Origin Destination Matrix Using Automatic Data Collection Systems*. Computer-Aided Civil and Infrastructure Engineering (2007) 22 : 376-378.

[Gordillo 06] Gordillo, F. *The Value of Automated Fare Collection Data for Transit Planning: An Example of Rail Transit OD Matrix Estimation*. Massachusetts Institute of Technology. September 2006. Master thesis

[Weiser 93] Weiser, M.. *Some Computer Science Issues in Ubiquitous Computing*. Communications of the ACM (July 1993) Vol. 36 Num. 7: 91-102.

[ASSC 02] Akyildiz, I., Su, W., Sankarasubramaniam Y. and Cayirci E.. *A Survey on Sensor Networks*. IEEE Communications Magazine (August, 2002) Vol. 40 Num. 8: 102-114.

[HKB 99] Heinzelman, W., Kulik, J. and Balakrishnan, H. Adaptive Protocols for Information Dissemination in Wireless Sensor Networks. In: Proc. of the 5th Annual ACM/IEEE International Conference on Mobile Computing (MobiCom' 99). Seattle, WA. 1999: 174-185.

[LDB 03] Leopold, M., Dydensborg, M. and Bonnet, P.. Bluetooth and Sensor Networks: A Reality Check. In: SenSys' 03, Los Angels, California. . November 2003: 103-133.

[SSJ 01] Shen, C., Srisathapornphat, C. and Jaikaeo, C. *Sensor Information Networking Architecture and Applications*. IEEE Peers. Commun. (August 2001) : 52-59.

[HSA 00] Hoblos, G., Staroswiecki, M. and Aitouche A. Optimal Design of Fault Tolerant Sensor Networks. In: IEEE Int'l. Conf.Cont. Apps.. Anchorage, AK. September 2000: 467-472.

[CDK 05] Dollimore, J., Kindberg, T. and Coulouris, G.. *Distributed Systems: Concepts and Design, 4th Edition* . Addison Wesley. June 2005.

[Bolton 01] Bolton, F.. *Pure CORBA* . Sams. July 2001.

[OV 99] Oszu, M. and Valduriez, P. *Principles of Distributed Databases Systems, 2nd Edition* . Prentice Hall. February 1999.

[Kleinrock 95] Kleinrock, L.. *Nomadic Computing - An Opportunity*. ACM SIGCOMM Computer Communication Review (January 1995) Vol. 25 Num. 1: 36-40.

[BN 92] Brachman, B. Neufeld, G. *TDBM: A DBM Library with Atomic Transactions*. Dept. Of Computer Science, University of British Columbia (1992) : . Technical Report

[Bach 86] Bach, M.. *The Design of The UNIX Operating System* . Prentice-Hall, Inc.. 1986.

[SR 05] Stevens, W. and Rago, S.. *Advanced Programming in the UNIX Environment* . Addison-Wesley . 2005.

[WS 95] Wright, G. and Stevens, W. *TCP/IP Illustrated, Volume 2: The Implementation* . Addison Wesley. January 1995.

[SFR 03] Stevens, W., Fenner, B. and Rudoff, A. *UNIX Network Programming Volume 1, 3rd Edition: The Sockets Networking API* . Addison-Wesley. November 2003.

[Stevens 94] Stevens, W. *TCP/IP Illustrated: Protocols, Volume 1* . Addison-Wesley. February 1994.

[Bisdikan 01] Bisdikan, C. *An Overview of the Bluetooth Wireless Technology*. IBM Research Divison (June 2001) : . Technical Report

[Wang 01] Wang, H. *Overview of Bluetooth Technology*. Penn State, Dept. of Electrical Engineering (July 2001) : . Technical Report

[GCR 03] Golmie, N., Chevrollier, N. and Rebala, O. *Bluetooth and WLAN Coexistence: Challenges and Solutions*. IEEE Wireless Communications (December 2003) Vol. 10 Num. 6: 22-29.

[GPS 04] Gehrmann, C., Persson, J. and Smeets, B. *Bluetooth Security (Artech House Computer Security Series)* . Artech House. June 2004.

[AK 05] Asthana, S. and Kalofonos, D. The Problem of Bluetooth Pollution and Accelerating Connectivity in Bluetooth Ad-Hoc Networks. In: 3rd IEEE Pervasive Computing and Communications (PerCom). Kauai Island, HI. March, 2005: 200-207.

[AL 00] Andersson, H. and Lundgren, L. *WAP Over Bluetooth*. Lund Institute of Technology, Lund University. 2000. Master Thesis

[SLBGMLBHI 03] Schilit, N., LaMarca, A., Borriello, G., Griswold, G., McDonald, D., Lazowska, E., Balachandran, A., Hong, J. and Iverson V. Challenge: Ubiquitous Location-Aware Computing and the "Place Lab" Initiative. In: Proceedings of the 1st ACM International Workshop on Wireless Mobile Applications and Services on WLAN. . 2003: 29-35.

[DFNI 07] Dikaiakos, M., Florides, A., Nadeem, T., Iftode, L. *Location-Aware Services Over Vehicular Ad-hoc Networks Using Car-to-Car Communication*. IEEE Journal on Selected Areas in Communication (2007) Vol. 25 Num. 8: 1590-1602.

[HHSWW 99] Harter, A., Hopper, A., Steggles, P., Ward, A. and Webster, P. The Anatomy of a Context-Aware Application. In: Proc. 5th ACM MOBICOM Conference. Seattle, WA, USA. August, 1999: 59-68.

[SAW 94] Schilit, B., Adams, N. and Want, R. Context-Aware Computing Applications. In: Proceedings of IEEE Workshop on Mobile Computing Systems and Applications. Santa Cruz, California, USA. 1994: 85-90.

[CK 00] Chen, G. and Kotz, D. *A Survey on Context-Aware Mobile Computing Research*. (2000) : . Technical Report

[SBG 99] Schmidt, A., Beigl, M. and Gellersen, H. *There is More to Context Than Location*. Computers & Graphics (1999) Vol. 23 Num. 6: 893-901.

[DA 99] Dey, A. and Abowd, G. *Towards a Better Understanding of Context and Context-Awareness*. Georgia Institue of Technology, College of Computing (June, 1999) : . Technical Report

[WHFG 92] Want, R., Hopper, A., Falcão, V. and Gibbons, J. *The Active Badge Location System*. ACM Transcations on Information Systems (1992) : 91-102.

[BRH 94] Bennet, F., Richardson, T. and Harter, A. Teleporting - Making Applications Mobile. In: Proceedings of IEEE Workshop on Mobile Computing Systems and Applications. Santa Cruz, California, USA. Decemeber, 1994: 82-84.

[PCB 00] Priyantha, N., Chakraborty, A. and Balakrishnan, H. The Cricket Location-Support System. In: 6th ACM International Conference on Mobile Computing and Networking (ACM MOBICOM. Boston, MA, USA. August, 2000: 32-43.

[RSH 04] Rukzio, E., Schmidt, A. and Hussmann, H. Physical Posters as Gateways to Context-Aware Services for Mobile Devices. In: 6th IEEE Workshop on Mobile Computing Systems and Applications . English Lake District, UK. 2004: 10-19.

[LC 06] LeBrun, J. and Chuah, C. Bluetooth Content Distribution Stations on Public Transit. In: ACM MobiShare' 06. Los Angels, CA, USA. September 2006: 63-65.

[LLSFC 06] Leguay, J., Lindgren, A., Scott, J., Friedman, T. and Crowcroft, J. Opportunistic Content Distribution in an Urban Setting. In: Proceedings of the ACM CHANTS. . 2006: 205-212.

[RSDI 05] Ravi, N., Stern, P., Desai, N. and Iftode, L. Accessing Ubiquitous Services Using Smart Phones. In: Proceeding of the 3rd International Conference on Pervasive Computing and Communications. . 2005: 383-393.

[PLGCLOWFK 04] Patterson. D., Liao, L., Gajos, K., Collier, M., Livic, N., Olson, K., Wang, S., Fox, D. and Kautz H. Opportunity Knocks: a System to Provide Cognitive Assistance with Transportation Services. In: Proceedings of UBICOMP 2004: The Sixth International Conference on Ubiquitous Computing. . 2004: .

[EP 04] Eagle, N. and Pentland, A. *Social Serendipity: Proximity Sensing and Cueing*. MIT Media Laboratory (May 2004) : . Technical Report

[SCCM 08] Sánchez, J., Cano, J., Calafate. C. and Manzoni, P. BlueMall: A Bluetooth-based Advertisement System for Commercial Areas. In: Proc. of the 3rd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired. Vancouver, British Columbia, Canada. 2008: 17-22.

[CMT 06] Cano, J., Manzoni, P. and Toh, C. *Ubiqmuseum: A Bluetooth and Java Based Context-Aware System for Ubiquitous Computing*. Wireless Personal Communications, Springer (2006) : . Accepted for publication

[GHJV 95] Gamma, E., Helm, R., Johnson, R. and Vlissidies, J. *Design Patterns: Ellements of Reusable Object-Oriented Software* . Addison-Wesley. 1995.

[Martin 03] Fowler, M. *Agile Software Development: Principles, Patterns, and Practices* . Prentice Hall. 2003.

Bibliography

[DF 01] Dupire, B. and Fernandez, E. *The Command Dispatcher Pattern*. Department of Computer Science and Engineering, Florida Atlantic University (2001) : .

[Knuth 98] Knuth, D. *The Art of Computer Programming. Volume 3: Sorting and Searching. 2nd Edition* . Addison Wesley. June 1998.

[WCO 00] Wall, L., Christiansen, T. and Orwant, J. *Programming Perl, 3rd Edition* . O'Reilly Media, Inc.. July 2000.

[Hayes 05] Hayes, A.. *Statistical Methods for Communication Science* . Routledge. 2005.

[Scheaffer 99] Scheaffer, R.. *Categorical Data Analysis*. NCSSM Statistics Leadership Institute (July 1999) : .

Appendix A

In this section we present several informations related to thequestionnaire that was executed. Some general considerations are shown as well statistical formulas used, result tables and images, questions coding table and the actual questionnaire.

A.1 General Considerations

- **General Population** – A concrete number of users using Horarios do Funchal public transportation service is needed. In the absence of such information, let's therefore assume that about one third of the general population residing in Madeira uses this service. This would correspond to about 100,000 users.
- **Sample Size** – The sample size is of 105 users.
- **Data Type** – The questions that where asked along the questionnaire captured characteristics and preferences about the individuals. Therefore, the data gathered by the questionnaires is *categorical.*
- **Confidence Level –** Assuming that we are in presence of a normal distribution, our selected confidence level is 95%. Any assumption made in this section is done with 95% of certainty.
- **Confidence Interva**l – Since we are working with categorical data, the formula used to calculate the error margin associated with a specific question is:

$$C.I. = Z \times \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

The meaning of the above variables is the following:

C.I – The Confidence Interval

Z – Z-Value for a confidence level of 95%. This corresponds to the *constant* value of 1.96.

$\hat{p}$ – The sample proportion. This corresponds to the percentage of users that belong to a certain group.

n – The sample population

A.2 Statistical Formulas

The Chi-Square Test of Independence is used in order to discover associations between variables. This test is commonly used when the data under analysis is categorical (e.g. it fits a specific group). The formula we used to calculate the Chi-Square value is the one that follows [Hayes 05]:

$$\chi^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} (O_{ij} - E_{ij})^2 / E_{ij}$$

- O – Represents the observed value on a given cell
- E – Represents the excepted value on a given cell

The Chi-Square is not a flawless independence test [Scheaffer 99, Hayes 05]. In fact when there exists one or more cells in a given contingency table with a excepted value (E) below 5 (e.g. E < 5) the tests becomes unreliable at best. To counter this problem we decided that in those contingency tables were E < 5, or where the obtained χ^2 raised some doubts, we would use the Yates Corrected Chi-Square, which formula is given by the expression [Hayes 05]:

$$\chi^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} [|(O_{ij} - E_{ij})| - 0.5]^2 / E_{ij}$$

Important is the fact that after applying the Yates Correction we must recalculate the value of χ^2. In order to help us achieve this we used an external mechanism which can be found at [5].

Throughout the tests we assume that $\alpha = 0{,}05$, where α represents the alpha level of significance. In order to better understand and compare the obtained results Table 7 was added.

It should also be noticed that although the above tests show us that a kind of association exit between variables, the tests themselves cannot clearly indicate how strong is that association, and what in fact the factors that influence such association. In order to obtain a more detailed data analysis we are going to make use of the *standardized residuals* (**Section A.3**) to unveil patterns that the Chi-Square tests cannot. As we are working with a normal distribution with a confidence level of 95%, then we are going to compare the obtained residuals to the constant Z-value of 1,96.

		Alpha Significance Level (α)					
		0.5	0.10	0.05	0.02	0.01	0,001
Degrees of Freedom (dF)	1	0.455	2,706	3.841	5,412	6.635	10,827
	2	1,386	4,605	5,991	7,824	9,210	13,815
	3	2,366	6,251	7,815	9,837	11,345	16,268
	4	3,357	7,779	9,488	11,668	13,277	18,465
	5	4,351	9,236	11,070	13,388	15,086	20,517

Table 6: Chi-Square Distribution Table

Question/Answer/Information Code	Question/Answer/Information
I1	The date when the questionnaire was answered
I2	Time when the questionnaire was answered
I3	Location where the questionnaire was answered
Q1.A	Sex
Q1.B	Age (in years)
Q2	How frequently do you use public transport buses?
Q3	How much time do you spend waiting at the bus stop?
Q4	While waiting at the bus stop or while traveling do you usually make use of a portable communication device (PDA, Cellphone, etc)?
Q5	If you answered **yes** in question **4**, please indicate in what ways you use your portable device:
Q5.A	Making/Receiving Phone Calls
Q5.B	Sending/Receiving SMS/MMS
Q5.C	Entertainment(Music, Games, etc)
Q5.D	Accessing On-line Content(Internet)
Q5.E	Other
Q6	While waiting at the bus stop, is there any kind of information/service that you would like to have access to, but for some reason can't or won't?
Q7	Does your portable communication device support the Bluetooth wireless technology?
Q8	If you answered **yes** in question **7**, do you usually keep your device with the Bluetooth option turned on?
Q9	If you answered **no** in Question **8**, please indicate the reason(s) for not enabling the Bluetooth option of your device:
Q9.A	Security
Q9.B	Power Consumption
Q9.C	No Particular Reason
Q9.D	Other
Q10	Please indicate the type of services of more interest to you:
Q10.A	While waiting at the bus stop
Q10.A1	Detailed bus schedules
Q10.A2	Buses arriving time
Q10.A3	Position indication of nearby bus stops
Q10.A4	Updated news headlines
Q10.A5	Entertainment(Games, Music, etc)

Q10.A6	Other
Q10.B	While traveling
Q10.B1	Detailed bus schedules
Q10.A2	Arriving time on destination stop
Q10.A3	Position indication of nearby bus stops
Q10.A4	Updated news headlines
Q10.A5	Entertainment(Games, Music, etc)
Q10.A6	Other
Q11	Please indicate your preferred way of accessing the services chosen at question **10**:

Table 7: Question, Answers and Informations Coding

A.3 Variable Associations

As we applied the Chi-Square test, several hypothesis were tested. The concrete details regarding the used formula as well the questions' coding that is used in this section, is present in the previous section. Here we summarize the existing associations as we present them into categories.

Location

- An association between I3 and Q10.A3 exists. $(\chi^2 = 9.65, dF = 3, p < 0.05)$

Demographics

- An association exists between Q1.A and Q5.B. $(\chi^2 = 6.507, dF = 1, p < 0.05)$
- An association between Q1.A and Q10.A3 exists. $(\chi^2 = 5{,}559, dF = 1, p < 0.05)$
- Q1.B and Q4 are related. $(\chi^2 = 10.09, dF = 4, p < 0.05)$
- An association between Q1.B and Q5.B was obtained. $(\chi^2 = 21.69, dF = 4, p < 0.01)$
- Q1.B is related to Q10.A4. $(\chi^2 = 10.17, dF = 4, p < 0.05)$

Bus Usage and Waiting Time

- An association was obtained between Q2 and Q9.C. $(\chi^2 = 12.87, dF = 4, p < 0.05)$
- Q3 and Q4 are related to each other. $(\chi^2 = 12.83, dF = 4, p < 0.05)$
- An association between Q3 and Q5.B exists. $(\chi^2 = 11.04, dF = 4, p < 0.05)$

Mobile Devices Pratices

- An association exists between Q5.A and Q10.A4. (χ^2=11.59,dF = 1, p < 0.01)
- Q5.A is associated with Q10.B4. (χ^2=9.22,dF = 1, p < 0.01)
- Q5.B and Q5.C are related to each other. (χ^2=5.85,dF = 1, p < 0.05)
- Q5.B and Q9.A are related. (χ^2=8.041,dF = 1, p < 0.05)
- Q5.B and Q10.A4 are associated with each other. (χ^2=5.064,dF = 1, p < 0.05)
- Q5.C and Q9.B have an association bounding them. (χ^2=5.63,dF = 1, p < 0.05)

Types of Services

- An association exists between Q10.A2 and both Q10.B1 and Q10.B2. (χ^2=6.726,dF = 1, p < 0.05)
- An association exists between Q10.A3 and Q10.B3. (χ^2=9.192,dF = 1, p < 0.01)
- Q10.A3 and Q10.B4 are related. (χ^2=8.47,dF = 1, p < 0.01)
- An association exists between Q10.A4 and Q10.B4. (χ^2=37.46,dF = 1, p < 0.01)
- Q10.B1 and Q10.B2 are related. (χ^2=10.33,dF = 1, p < 0.01)
- Q10.B2 and Q11 are related. (χ^2=10.1,dF = 3, p < 0.05)
- Q10.B3 and Q11 are associated. (χ^2=10.06,dF = 3, p < 0.05)

A.4 Result Tables and Images

Rows are levels of	loc_nearby_stops				
Columns are levels of	loc				
No Selector					
	Av. Mar	**Lido**	**Monte**	**UMa**	**total**
Doesn't Want Service	49	8	12	11	80
	86	72.7	75	52.4	76.2
	46.7	7.62	11.4	10.5	76.2
	0.845432	-0.13159	-0.0545545	-1.25	0
Want Service	8	3	4	10	25
	14	27.3	25	47.6	23.8
	7.62	2.86	3.81	9.52	23.8
	-1.51236	0.235396	0.09759	2.23607	0
total	57	11	16	21	105
	100	100	100	100	100
	54.3	10.5	15.2	20	100
	0	0	0	0	0

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 71: Users that want the locations of nearby bus stops while waiting for the bus

Rows are levels of	sends_msgs	
Columns are levels of	sex	
No Selector		
	Female	**Male**
Doesn't send msgs	14	28
	27.5	51.9
	13.3	26.7
	-1.41698	1.37706
Sends msgs	37	26
	72.5	48.1
	35.2	24.8
	1.15696	-1.12437

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 72: Users that use their device for messaging

Rows are levels of **loc_nearby_stops**
Columns are levels of **sex**
No Selector

	Female	Male
Doesn't Want Service	44	36
	86.3	66.7
	41.9	34.3
	0.825029	-0.801784
Want Service	7	18
	13.7	33.3
	6.67	17.1
	-1.47586	1.43427

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 73: Female and Male users who wan't the location of nearby bus stops

Rows are levels of **news_at_bus_stop**
Columns are levels of **device_for_calls**
No Selector

	Doesn't use device for calls	Uses device for calls
Doesn't want news at bus stops	51	22
	82.3	51.2
	48.6	21
	1.20255	-1.44399
Wants news at bus stops	11	21
	17.7	48.8
	10.5	20
	-1.81631	2.18097

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 74: Device usage for phone calling and news service at bus stops association

Rows are levels of **news_travel**
Columns are levels of **device_for_calls**
No Selector

	Doesn't use device for calls	Uses device for calls
Doens't want news while travl	43	17
	69.4	39.5
	41	16.2
	1.27284	-1.52743
Want news while travl	19	26
	30.6	60.5
	18.1	24.8
	-1.46883	1.76373

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 75: Device usage for phone calling and news service while traveling *associatio*n

Rows are levels of **device_for_entr**
Columns are levels of **device_for_msgs**
No Selector

	Doesn't use device for msgs	Uses device for msgs
Doesn't use device for entr.	33	35
	78.6	55.6
	31.4	33.3
	1.1121	-0.900825
Uses device for entr.	9	28
	21.4	44.4
	8.57	26.7
	-1.50764	1.23098

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 76: Device usage for messaging and entertainment association

Rows are levels of **bt_off_security**
Columns are levels of **device_for_msgs**
No Selector

	Doesn't use device for msgs	Uses device for msgs
Not for security reasons	34	34
	81	54
	1.30384	-1.06458
Security reasons	8	29
	19	46
	-1.76758	1.44322

table contents:
Count
Percent of Column Total
Standardized Residuals

Figure 77: Device usage for messaging and disconnecting Bluetooth for security reasons association

Rows are levels of **news_at_bus_stop**
Columns are levels of **device_for_msgs**
No Selector

	Doesn't use device for msgs	Uses device for msgs
Doesn't want news at bus stops	24	49
	57.1	77.8
	22.9	45.7
	-0.962383	0.785717
Wants news at bus stops	18	14
	42.9	22.2
	17.1	13.3
	1.45344	-1.18673

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 78: Device messaging usage and news service at bus stops associaiton

Rows are levels of **bt_off_power**
Columns are levels of **device_for_entr**
No Selector

	Doesn't use device for entr.	Uses device for entr.
Not power consumption reasons	55	22
	80.9	59.5
	52.4	21
	0.726933	-0.98548
Power consumption reasons	13	15
	19.1	40.5
	12.4	14.3
	-1.20548	1.63423

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 79: Turning off Bluetooth for power consumption reasons and using it for entertainment

Rows are levels of **sched_traveling**
Columns are levels of **bus_arrival_time**
No Selector

	Bus arrival time	Doesn't want service
Doesn't want service	56	29
	74.7	96.7
	53.3	27.6
	-0.605021	0.956622
Schedules traveling	19	1
	25.3	3.33
	18.1	0.952
	1.24728	-1.97213

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 80: Users that want to know the bus arrival time and the detailed bus schedules while traveling

Rows are levels of **dest_arrival_time**
Columns are levels of **bus_arrival_time**
No Selector

	Bus arrival time	Doesn't want service
Destinity arrival time	49	12
	65.3	40
	46.7	11.4
	0.822403	-1.30033
Doesn't want service	26	18
	34.7	60
	24.8	17.1
	-0.96833	1.53106

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 81: Users that want the bus arrival time and the destiny arrival time

Rows are levels of **stop_loc_travl**
Columns are levels of **stop_loc_bstop**
No Selector

	Bus stop loc. at bstop	Doesn't want service
Bus stop loc travel	14	19
	56	23.8
	13.3	18.1
	2.19148	-1.22508
Doesn't want service	11	61
	44	76.3
	10.5	58.1
	-1.48364	0.829381

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 82: Bus stops locations both while waiting for the bus and traveling

Rows are levels of **news_travel**
Columns are levels of **stop_loc_bstop**
No Selector

	Bus stop loc. at bstop	Doesn't want service
Doens't want news while travl	8	52
	32	65
	7.62	49.5
	-1.66304	0.92967
Want news while travl	17	28
	68	35
	16.2	26.7
	1.92032	-1.07349

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 83: Nearby bus stops locations while at the bus stop and news service while traveling association

Rows are levels of **news_travel**
Columns are levels of **news_at_bus_stop**
No Selector

	Doesn't want news at bus stops	Wants news at bus stops
Doens't want news while travl	56	4
	76.7	12.5
	53.3	3.81
	2.21187	-3.34077
Want news while travl	17	28
	23.3	87.5
	16.2	26.7
	-2.55405	3.85758

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 84: Users that news service in bus stops and while traveling

Rows are levels of **dest_arrival_time**
Columns are levels of **sched_traveling**
No Selector

	Doesn't want service	Schedules traveling
Destinity arrival time	43	18
	50.6	90
	41	17.1
	-0.908042	1.87198
Doesn't want service	42	2
	49.4	10
	40	1.9
	1.06916	-2.20414

table contents:
Count
Percent of Column Total
Percent of Table Total
Standardized Residuals

Figure 85: Schedules while traveling and destiny arrival time association

	10 – 20	20 – 30	30 – 40	40 – 50	50+
Uses Device	37 (0,72)	38 (-0,67)	8 (0,4)	2 (0,09)	0 (-0,88)
Doesn't use Device	3 (-1,49)	15 (1,38)	0 (-0,83)	0 (-0,19)	2 (1,82)

Table 8: *Frequency and standardized residuals of age/device usage association*

	10 – 20	20 – 30	30 – 40	40 – 50	50+
Doesn't use	4 (-2,88)	28 (1,37)	6 (1,29)	2 (0,78)	2 (-0,78)
Uses	36 (2,35)	25 (-1,12)	2 (-1,05)	0 (-0,64)	2 (-0,64)

Table 9: *Users that use their device for messaging purposes*

	10 – 20	20 – 30	30 – 40	40 – 50	50+
Doesn't Want	35 (1,27)	34 (-0,39)	3 (-0,87)	1 (-0,09)	0 (-0,75)
Wants	5 (-1,92)	19 (0,58)	5 (1,32)	1 (0,14)	2 (1,14)

Table 10: *Users that want a news service at the bus stops*

	1 – 5	5 – 10	10 – 15	15 – 30	30+
Doesn't use	2 (-1,65)	23 (1,87)	13 (-0,53)	4 (0,05)	0 (-0,34)
Uses	15 (1,35)	15 (-1,53)	26 (0,43)	5 (-0,04)	2 (0,27)

Table 11: *Waiting time and device usage for messaging purposes*

	1 – 5 min	5 – 10 min	10 – 15 min	15 – 30 min	30+ min
Uses Device	16 (0,47)	23 (-1,31)	36 (0,7)	8 (0,08)	2 (0,09)
Doesn't use device	1 (-0,97)	15 (2,7)	3 (-1,44)	1 (-0,16)	0 (-0,19)

Table 12: *Waiting time and device usage association*

	2+ day	1 day	1 – 5 week	1 – 5 month	Very rarely
Has reason	60 (0,46)	7 (-0,1)	11 (-0,99)	9 (0,13)	8 (-0,13)
Doesn't have reason	2 (-1,4)	1 (0,3)	6 (3,05)	0 (-0,39)	1 (0,39)

Table 13: *Users frequency usage and reason for disabling Bluetooth discoverable mode*

A.5 Questionnaire

UNIVERSIDADE da MADEIRA

Questionnaire about public transport-related services

v 0.4P – Out/08 – EN

As an effort to better understand the needs of users of public transports, and in hope to make your experience a better one while using them, please consider answering the following questions

1. In order to comprehend and contextualize the gathered information, please indicate the following information:

   a) Sex: ☐ Masculine ☐ Feminine

   b) Age (in years): ☐ 10 - 20 ☐ 20 - 30 ☐ 30 - 40 ☐ 40 - 50 ☐ 50+

2. How frequently do you use public transport buses?

   ☐ 2 or more times a day ☐ 1 time a day
   ☐ 1 - 5 times a week ☐ 1 - 5 times a month
   ☐ very rarely

3. How much time do you spend waiting at the bus stop?

   ☐ 1 - 5 minutes ☐ 5 - 10 minutes
   ☐ 10 - 15 minutes ☐ 15 - 30 minutes
   ☐ 30+ minutes

4. While waiting at the bus stop do you usually make use of a portable communication device (PDA, Cellphone, etc)?

   ☐ Yes
   ☐ No

5. If you answered **yes** in question **4**, please indicate in what ways you use your portable device:

   ☐ Making/Receiving phone calls
   ☐ Sending/Recieving SMS/MMS
   ☐ Entertainment (Music, Games, etc)
   ☐ Accessing on-line content (Internet)
   ☐ Other (Please specify)

   _____
   _____

6. While waiting at the bus stop, is there any kind of information/service that you would like to have access to, but for some reason can't or won't?

   ☐ No
   ☐ Yes (Please specify)

_____

_____

7. Does your portable communication device support the Bluetooth wireless technology?

☐ Yes ☐ I don't know

☐ No

8. If you answered **yes** in question **7**, do you usually keep your device with the Bluetooth option turned on?

☐ Yes ☐ I don't know

☐ No

9. If you answered **no** in Question **8**, please indicate the reason(s) for not enabling the Bluetooth option of your device:

☐ Security

☐ Power consumption

☐ No specific reason

☐ Other (Please specify)

_____

_____

10. Please indicate the type of services of more interest to you:

While waiting at the bus stop	**While traveling**
☐ Detailed bus schedules	☐ Detailed bus schedules
☐ Buses arriving time	☐ Arriving time on destination stop
☐ Indiaction of nearby bus stops	☐ Indiaction of nearby bus stops
☐ Updated news headlines	☐ Updated news headlines
☐ Leisure (Games, Music, etc)	☐ Leisure (Games, Music, etc)
☐ Others (Please specify)	☐ Others (Please specify)
_____	_____
_____	_____

11. Please indicate your preferred way of accessing the services chosen at question **10**:

☐ I want to be responsible for accessing the services

☐ I want to register for the service, and then my device will automatically access it

☐ My device will automatically access the service without my previous consent

☐ I really don't care as long as the services are of any relevance to me