FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Open Location Based Services Application Platform

Pedro Luís de Faria e Coelho

Report of Project/Dissertation
Master in Informatics and Computing Engineering

Supervisor: João Correia Lopes (Professor Auxiliar)

Supervisor: Ana Cristina Aguiar (Professor Auxiliar Convidado)

28^{th} June, 2010

Open Location Based Services Application Platform

Pedro Luís de Faria e Coelho

Report of Project/Dissertation
Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Jorge Manuel Gomes Barbosa (Professor Auxiliar)
External Examiner: Rui João Peixoto José (Professor Auxiliar)
Internal Examiner: João António Correia Lopes (Professor Auxiliar)

26^{st} July, 2010

Abstract

According to Virrantaus, Location Based Services are information services accessible with mobile devices through the mobile network and utilizing the ability to make use of the location of the mobile device.

While there certainly isn't a shortage of Internet based applications that provide Location Based Services, problems such as bad connectivity, high Internet access costs and technological impairments keep those applications from running properly. Moreover, nternet connectivity is not yet ubiquitous. Besides that, systems that have off-line functionality use closed and proprietary formats to convey data to the end user, denying him the chance to produce his own content.

The aim of the Open LBS Application Platform is to provide user generated localized content to a user without the need for the user to be online. In order to achieve this goal, a cache of the content is stored in the user's mobile device, and it's kept up to date using a web application with a central repository. That cache is divided into Packages, each composed of various Locations that represent a physical spot on the world. Each Location provides its own Contents that may be any type of file.

To prove the validity of this concept, a prototype regarding a mobile application for the Android Operating System and a central repository that also acts as a content generation platform for this project was fully implemented.

Evaluating the implemented prototype using test criteria regarding the size of the resulting Content Packages and the platform's bandwidth, CPU and battery usage revealed satisfactory results, thus confirming both the feasibility and purpose of this project.

Resumo

Segundo Virrantaus, Serviços Baseados na Localização são serviços de informação accessíveis a partir da rede móvel num dispositivo portátil que tiram partido da informação de localização desse mesmo dispositivo.

Enquanto não há certamente escassez de aplicações baseadas na Internet que fornecem Serviços Baseados na Localização, problemas como a falta de conectividade, alto custo do acesso online móvel e limitações tecnológicas impedem essas aplicações de funcionarem adequadamente. Para além disso, a conectividade à Internet ainda não é omnipresente. Para além disso, os sistemas que funcionam em modo offline usam maioritariamente formatos fechados quando encaminham conteúdo para o utilizador, negando-lhe a hipótese de criar o seu próprio conteúdo.

O objectivo deste trabalho é criar um mecanismo que forneça conteúdo localizado gerado pelo utilizador sem que seja necessário um acesso à Internet. De maneira a conseguir-se o proposto é criada uma *cache* no dispositivo móvel do utilizador contendo esses mesmos conteúdos localizados. Esta é mantida actualizada de acordo com um repositório de conteúdos central. A *cache* é dividida em Pacotes, sendo cada um constituído por Localizações, que representam um ponto geográfico no mundo. A cada Localização estão associados diferentes Conteúdos que podem ser de qualquer tipo.

Para provar a validade desta ideia foi implementado um protótipo composto por uma aplicação móvel para o Sistema Operativo *Android* e por um repositório central que também funciona como plataforma de geração e publicação de conteúdos localizados.

A avaliação resultante desse mesmo protótipo, usando critérios como o tamanho do Pacote de Conteúdos resultante e o gasto de banda, bateria e processador da plataforma móvel revelaram resultados bastante satisfactórios, confirmando a viabilidade e propósito deste mesmo projecto.

Acknowledgements

For this and all associated work, I blame:
My parents for ever letting me out of the house;
My two supervisors for the insightful and interesting discussions;
Myself for all the patience, clairvoyance and inspiration I ever gave me.

Pedro Luís de Faria e Coelho

"There is no knowledge that is not power."

Ralph Waldo Emerson

Contents

List of Figures

List of Tables

LIST OF TABLES

Abbreviations

FRR	*First Read Rates*
GPS	*Global Positioning System*
HTTP	*Hypertext Transfer Protocol*
LBS	*Location based services*
MPEG	*Moving Picture Experts Group*
RAD	*Rapid Application Development*
RSS	*Really Simple Syndication*
SQL	*Structured Query Language*
UDDI	*Universal Description, Discovery and Integration*
ULI	*Universal Location Identifier*
URI	*Universal Resource Identifier*
URL	*Universal Resource Location*
XML	*Extensible Markup Language*
XSLT	*Extensible Stylesheet Language Transformations*
WSDL	*Web Services Description Language*
WWW	*World Wide Web*

Chapter 1

Introduction

Mobile telephony and the Internet have long ago revolutionized people's lifestyle. For a long time now, Humanity has struggled to create a universe where one could have an ubiquitous capacity to connect to the digital world and create a bridge to the real one, crossing information relative with context with interactions with other humans connected.

1.1 Context and motivation

The road to pervasive connectivity is still long. Several barriers to this goal emerge everyday, keeping humans from *the global village* experience. One of these impediments is 3G/EDGE coverage. Even in a small country like Portugal reports show[1] low coverage in areas that are not as remote as one would thought they would be: the country's main roads. Should one try to, for example, consult an Internet based device to navigate in these kind of situations, one will encounter severe difficulties doing so.

Other major issue relates to communication's billing. Several mobile applications target the area of tourism, providing information related to the user's location context. Most of these applications rely on constant online access, which paired with costly abroad roaming taxes leads to heavy costs on the user.

Finally, there is the technological impairment. Technologies such as GPS are very useful, but they're confined to outdoor use. Should a user go to a large building, he has no way to make use of GPS positioning, because nor the technology offers such functionality and precision, nor exist maps and guides for that particular building provided by the GPS device's manufacturer.

Concluding, today's false notion of ubiquitous connectivity causes several problems to end-users, allowing for unexpected or unpleasant situations to occur sometimes at the worst of the timings — on a holiday trip, for example.

What this project proposes is to use the device's internal storage capacity to save some of that contextual information so it can be retrieved later, without the use of online connectivity. It is also

[1] See http://www.anacom.pt/render.jsp?contentId=832231

focus of research the specification of an open data format, so that location based information may be produced by interested end-users.

1.2 Location Based Services and Context-awareness

As this research's main produce will be a Location Based Services platform, we should consider first defining the concept of Location Based Services. A definition, proposed by Virrantaus et al. states:

> "Location Based Services are information services accessible with mobile devices through the mobile network and utilizing the ability to make use of the location of the mobile device [VMG+01]."

While it may seem this definition contradicts this project's purpose, it merely needs some clarification in order to adapt it flawlessly to this context: though it states that the services are available through the mobile network, it does not specify if that communication is synchronous — the case of a network dependent application, or asynchronous — this particular case.

Location Based Services applications are inserted in the broader field of Context-awareness. Context awareness is a technique for developing pervasive computer applications that are flexible, adaptable and capable of autonomously on behalf of users [HI06]. Context-aware applications use the environment information around themselves and the user to adapt and perform accordingly.

1.3 Objectives

This research's main goals involve:

- Researching previous works based on Context gathering, understanding and architectures;
- Designing a generic solution for the problem depicted, taking into account both mobile equipment and bandwidth limitations;
- Developing a prototype application platform that provides location based information upstream to applications using it, based on the conceptual solution presented;
- Implementing a back-office solution providing web services allowing for the location based information maintenance and retrieval;
- Testing the functionality of the application in a proof-of-concept end-user scenario;
- Proving its feasibility and usefulness through the results of the tests performed.

1.4 Document structure

Chapter 2 exposes the state of the art regarding this project. It starts with an introduction in Context-awareness, followed by a small survey on context-aware computing models and architectures.

Next, in Chapter 3, a close up describing the technological aspects of this project, along with a small comparison regarding 2D bar-codes, is done.

Afterwards, in Chapter 4, the problem is properly stated and a conceptual solution to that problem is presented, dividing that solution in smaller areas in order to achieve a better understanding of it.

In Chapter 5, the implemented prototype is properly presented, explaining its architecture, work flows, database implementation and development methodology.

Results of the evaluation performed on the prototype are available in Chapter 6, along with a small discussion on the topic.

Final thoughts and future work may be found in Chapter 7.

Chapter 2

Location Based Services

In this chapter the state of the art regarding this project is presented. First, a brief introduction to Context-awareness is regarded, following a small survey on context-aware computing models. Surveys on context-aware architectures are also found afterwards.

2.1 Context-aware computing models

Context awareness is an important factor when considering a mobile LBS application. Location context is vital to this project so a brief research on context awareness was conducted in order to better understand its definition and implications.

Anind K. Dey and Gregory D. Abowd [ADB+99] embraced this endeavor with their work on the subject, entitled "Towards a Better Understanding of Context and Context-Awareness". In their pursue, they have managed to fully understand what constitutes a context-aware application and what context is. Their work starts by finding and analyzing previous definitions of the "context-aware" term.

2.1.1 Previous definitions of Context

Context began being defined as location, identities of nearby people and objects, and changes to those objects — Schilit and Theimer [ST94]. Similar definitions were proposed by Brown et al. [ST94], Ryan et al. [RPM97] and Dey [Dey98], being all very similar, differing only on other context details as the time of day, season, temperature, etc. Soon, other definitions risen providing synonyms for context; for example, referring to context as the environment or situation. Some considered context to be the user's environment — such as Franklin & Flaschbart [JFF98], while others considered it to be the application's environment — such as Brown.

Rising in complexity, further definitions by Schilit et al. [ST94], Dey et al. [DAW98] and Pascoe [Pas98] added other Context elements such as where you are, who you are with, and what resources are nearby, closing in on the definition proposed by Dey and Abowd [ADB+99].

2.1.2 A broader definition of context

Citing Abowd et al. [ADB+99, 2.2 Our definition of context], "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

This definition broads the concept of Context, stretching it to the area between the User entity and the Application entity, inclusive. The example given by the authors is quite elucidative:

> "Take the task of using a spreadsheet to add a list of weights as an example. The entities in this example are the user and the application. We will look at two pieces of information — presence of other people and location — and use the definition to determine whether either one is context. The presence of other people in the room does not affect the user or the application for the purpose of this task. Therefore, it is not context. The user's location, however, can be used to characterize the user's situation. If the user is located in the United States, the sum of the weights will be presented in terms of pounds and ounces. If the user is located in Canada, the sum of the weights will be presented in terms of kilograms. Therefore, the user's location is context because it can be used to characterize the user's situation."

2.1.3 Characterizing context

Further reading reveals a categorization regarding context. Abowd et al. [ADB+99] believed a division would help application designers uncover the most likely pieces of context that will be useful in their applications.

Following their previous conclusions, the authors searched for the answers to the following questions regarding Context:

- Who are you and who are you with?
- Where you are?
- What are you doing?
- When are you doing it?

As to be seen, asking those questions does make sense. A small survey conducted by the same authors revealed many context-based applications striving to answer these questions. Therefore, the answer to these questions must be the categorization searched:

Identity answering Who are you and who are you with?

Location answering Where you are?

Activity answering What are you doing?

Time answering When are you doing it?

These are the primary types for characterizing the environment of the situation. These context types not only answer those questions, but also act as indexes into other sources of contextual information. For example, given a person's identity, we can acquire many pieces of related information such as phone numbers, addresses, email addresses, birth-date, list of friends, relationships to other people in the environment, etc. Following the author's train of thought, all this information that can be gathered through these primary types become a secondary type of information.

2.1.4 Defining Context-aware Computing

Following the definition of context-awareness, it's now only natural to arrange a proper definition for its computer counterpart. That is, how will the artifacts of this project react to stimuli provided by its context-awareness.

Following a path analog to defining context-awareness, a serious of previously stated definitions are analyzed in the search of a broader approach. Context-aware has become somewhat synonymous with other terms: adaptive, reactive, responsive, situated, context-sensitive and environment directed. Previous definitions of context-aware computing fall into two categories: using context and adapting to context. Seen that various authors differ on that same grounds, a natural approach to broaden and relax the definition is to merge both points of view. Therefore, Abowd et al. [ADB+99] define that a system is context-aware if:

> "[...] it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."

This general definition allows for a wider characterization of context-aware applications. As the authors state, using any other of the two major established definitions will cause applications not fit in the definition. One good example is perhaps an application that simply displays the context of the user's environment to the user not adapting its behavior to context, as Abowd et al. states [ADB+99]. This application would not fall into any of the former two specific definitions. However, the broader definition considers it as being context-aware.

2.1.5 Context modeling

To represent context data in a machine processable form, a context model should be considered. Strang and Linnhoff-Popien [SP04] summarized the most relevant context modeling approaches, which are based on data structures in order to represent contextual data in a system.

The simplest way to model contextual data is using key-value models. Due to its simplicity, this method is widely used in contextual service frameworks. Service discovery is quickly implemented using key-value matching algorithms.

Some frameworks use markup to model their contextual data. In this kind of modeling, data is arranged hierarchically, in a structure consisting of markup tags with attributes and content. In particular, the content of some markup tags is recursively defined by other markup tags. These markup schemes are often referred as Profiles. They are usually based upon the XML language and variants. These kinds of context modeling approaches usually extend and complete the basic modeling vocabulary and procedures to try to cover higher complexity of contextual information when compared to static key-value models.

One can also use The Unified Modeling Language (UML) as a suitable model for modeling context, due to its generic structure, thus achieving what is called a graphical models of contextual data.

Modeling context by using object-orientation is also a usual technique, thus harnessing of the full power of object orientation (e.g., encapsulation, inheritance, polymorphism). Existing approaches use various objects to represent different context types (such as temperature, location, etc.), and encapsulate the details of context processing and representation. The access to information is then done through defined interfaces.

The next level in modeling should then be to use a logical approximation, which are inherently highly formal. Facts, expressions and rules are used to define a context model. A logic based system is then used to manage contextual information and allows to add, update, remove and relate new facts. Due to its nature, new facts may be derived based on existing rules in the system.

Finally, there are the ontology based models. Ontologies represent a concept vocabulary and the relationships between them. Ontologies are then a very promising instrument for modeling contextual information due to their high possibility for applying ontology reasoning techniques in order to gather new relevant information. Strang and Linnhoff-Popien [SP04] show that ontologies are the most expressive models. In order to these becoming most effective, Korpipää et al. [KMK+03] proposed some requirements and goals having designed an ontology for context-aware systems: it should be as simple as possible, thus easing the development and implementation process; should be flexible and extensible, supporting the addition of new terms without too much hassle; should be generic, not limited to a specific area but to a broad context type range; and it should be expressive, describing as much as possible with arbitrary detail.

2.1.6 Some proposed architectures

In order to aid the development of context-aware applications, several architectures and models have been proposed to deal with numerous related problems: mainly the heterogeneousness of sensor equipments, the applications' distributed nature and the variety of context-related information. Furthermore, the method of acquiring context data from the environment plays a very important role, because it normally predefines the architecture of the system to be designed [BDR07].

There may be three main different approaches to gather context information, according to Baldauf et al. [BDR07].

Direct sensor access

This approach is normally used in devices with embedded sensing equipment. The data is directly gathered by the application from the sensors, which normally causes the application to need built-in drivers for the direct access. This fact makes the application tightly coupled with the device, which makes this approach not very desirable when designing a multi-platform application, or a distributed system. This model also needs the device to allow direct sensor access, which is not the case, in this project.

Middleware infrastructure
In this approach, application logic and low level components are separated, the latter usually encapsulated in order to provide easier access, as well as some extensibility. Middleware approaches are known by their common layered architecture, providing scalability and re-usability to the system, as well as hiding low level components to the application developer. This method becomes desirable as it's neither too complex for a simple hand-held device implementation, neither too simple or sloppy to limit the implementation to one kind of device.

Context server
This is the most complex approximation. It defines the implementation of a context server, allowing various applications to fetch contextual data from this source. The gathering sensor data process is moved to this component to facilitate concurrent multiple access, thus relieving clients of resource intensive operations. This approach is highly desirable for distributed systems relying on low-end endpoints with not much processing power.

Similarly, architectures for context proposes some models to deal with the interaction between components.

Widgets
A context data source abstraction that can be used as a software component in order to retrieve and call information related to a single source of information.

Networked Services
Addressing a similar model as widgets but abstracting them as network services, allowing for a more robust architecture — each data source would function as a different process — trading for performance.

Blackboards
Adopting a data centric point of view. Rather than sending requests to distributed components and getting callbacks from them, a process posts messages to a common shared message board. Therefore, there is only a single data source that holds all the contextual data, responding accordingly with the request performed.

Due to this project's hardware requirements, the Middleware approach becomes the most relevant. The operating system this experiment will use already implements some of the concepts allied to that approach, and so it will deserve most of our attention onwards.

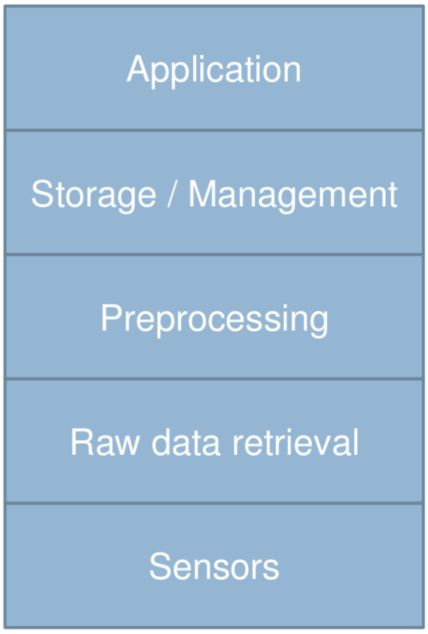

Figure 2.1: A hierarchical model

2.2 Context-aware architectures

In a survey considering Context-aware systems by Baldauf et al. [BDR07] a brief study is made regarding some middleware framework models. In this section, some of those architectures are analyzed. Several architectures have been left behind due to their different purpose — distributed systems. As this project will be running on a single hand-held device, having as a requirement not enforcing constant on-line access, we're more interested in architectures that can be implemented as local components, as opposed to remote components.

The first analyzed architecture resumes itself as a simple hierarchical model in which its tiers represent a level of complexity in the application, starting with the low level controls and drivers below, up to the application logic layer, on top. A hierarchical model concept can be seen in Figure 2.1, similar to the one found on the work of Baldauf et al. [BDR07].

This is an useful approach for an architecture to use with devices that lack the processing power, due to it's simplicity, but lacks robustness. All of the components lack redundancy, so they all constitute a single point of failure.

The Context Managing Framework, presented by Korpipää et al. [KMK+03], has a similar approach to architecture, but provides distributed components on the low level controls. Then, the Context Manager acts like a blackboard to all the services below. This is depicted in Figure 2.2.

When entities communicate, the context manager functions as a central server while other entities act as clients and use services the server provides. The context manager, any resource servers, and applications run on the mobile device itself.

Another architecture worth mentioning is the one used by Salber et al. [SDA99], which we will

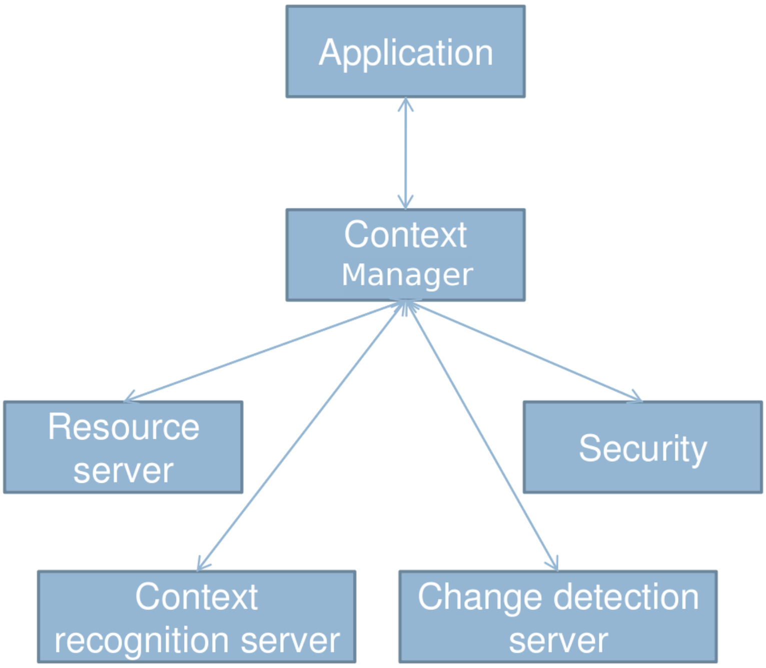

Figure 2.2: A hierarchical model variant

describe in more detail further on. This project takes a step towards a peer-to-peer architecture, where each node represents a sensor unit (called widgets). Though distributed, this can be easily implemented abstracting nodes as threads, providing with the same principle a network deprived environment, such as this project's.

A representation of an architecture using widgets is found on Figure 2.3.

Besides these three models, there are several other architectures that can't apply to this project, due to its centralized nature. One worth mentioning is the architecture used in the Hydrogen approach, by Hofer et al. [HSP+03]. This follows a similar layered pattern, but implements a distinction between local and remote contextual objects, in an attempt to overcome the inherent one point failure that layered patterns suffer. Local objects may be accessed locally, representing information the local device knows about. Remote objects may be unavailable, representing something that some other device knows. When two devices are in physical proximity they are able to exchange these contexts in a peer-to-peer manner, this process being called context sharing. This architecture is oriented to this kind of behavior, so it is not really interesting to this project's context.

Another honorable mention is the sentient object model, by Biegel et al. [BC04]. Similar to a neural network, a sentient object communicates with sensors which produce software events and actuators which consume software events. This sentient object may be hooked to a sensor, or to another sentient object, allowing for more complex and intricate relationships between context data, providing the application with highly refined information. Unfortunately, this architecture has been done with large systems, with multiple and diversified sensors in mind. Since our appli-

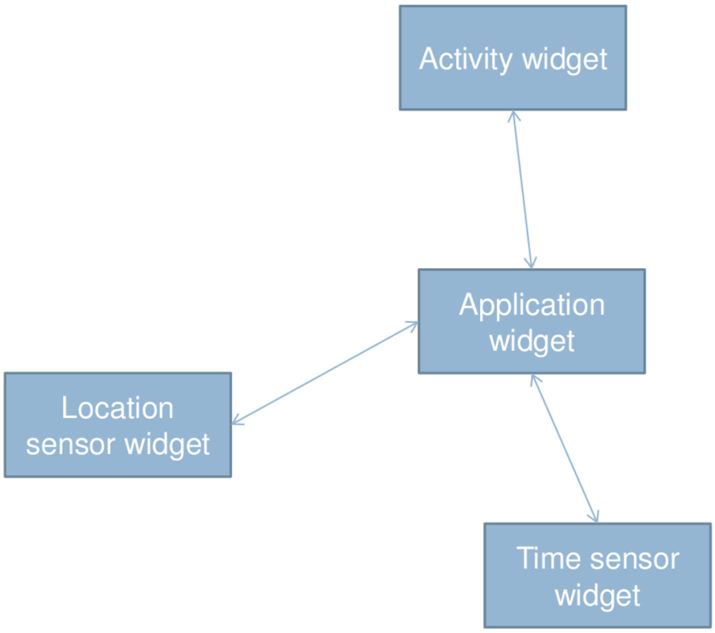

Figure 2.3: Widget based architecture

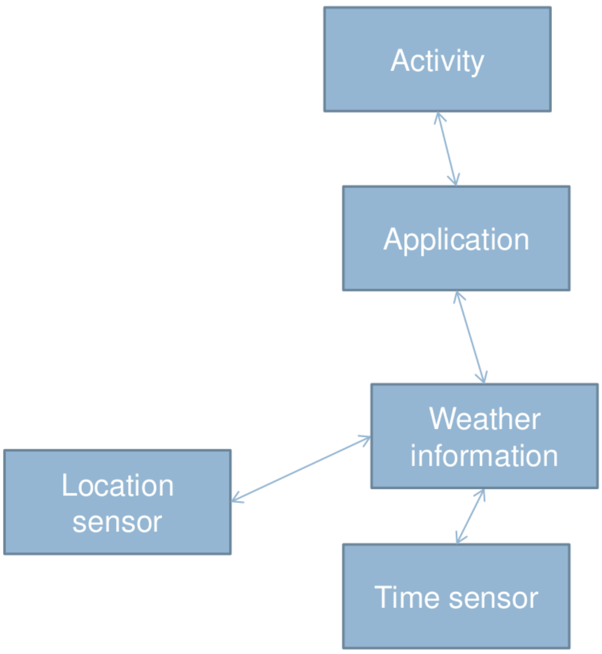

Figure 2.4: Sentient object based architecture

cation will fetch context data from a few data sources, applying this kind of architecture to this project will seem like overkill.

2.3 Context-aware applications

What follows is the result of a survey on existing work on the subject of applications using context-awareness to provide relevant information to the user.

From that survey, the three applications that relate the most to this project's context are presented. They all share several characteristics: they're intended to be used by the regular population, they all make use of location context in order to provide with relevant information and they are designed to run on mobile and low power devices.

2.4 CATIS

CATIS, by Pashtan et al. [PBA+03], is a mobile application that uses context information to deliver specific content to its user. It uses Web services and XML technologies for its implementation, and relies on constant on-line access to achieve its goal.

Its purpose is to deliver guide information to tourists, using its location-aware functionalities to generate proper content relevant to the user's position, speed and time of day. It also features user customization, allowing the user to rate places the application suggests, in order to provide the user with his best preference.

The system is also device-aware. Depending on the device the application runs, information may be provided using different techniques, according to the device's capabilities.

2.4.1 Architecture

Relying on constant on-line access, CATIS's architecture spreads content and context-information amongst different entities.

Since the technology is web based, web services are used, and a web browser on the client device is needed to perform all the interaction with the system. From here we may conclude the client does not store any kind of information. Location contextual data is sent in the HTTP request, and the rest of the system does the rest. An application web server is also needed in order to receive the requests, exchange information with other contextual and service data sources and process the user's request.

A UDDI (Universal Description, Discovery and Integration) services directory is used in order to perform context-indexed searches on user requested content, providing users with a centralized registry of tourist information services (e.g., a restaurant finder service).

To handle the user's context information, a context manager is used in order to store it and process it. It also stores the user's preferences for future reference.

Finally, a collection of Web Services that deliver tourist content (e.g., landmark information, restaurant locations, etc.) is used in order to serve as the UDDI's database. Each Web service has a WSDL (Web Services Description Language) document in XML format that describes the Web service's interface [PBA+03].

2.4.2 General usage

While the application runs, it sends context information to the context server repeatedly over a time interval. Every time a user requests for information, he connects to the application server to request the information. The application server then queries the context manager for the last information on the user's context and preferences.

After context data is retrieved, a search is performed in the UDDI in order to find the addresses of the available restaurant Web Services. A search is then performed on their databases for the appropriate addresses and after filtering out those that are not eligible (e.g. too far away), an XML list is returned to the application server.

The application server filters the XML documents according to the user's preferences, converts the information into a format supported by the client's device and sends it back to the client.

This system relies heavily on XML and related technologies to perform. As an example, transforming the XML list of locations to a format that is supported by the client's device is done using XSLT, and searching the Web Services is done using XQuery [PBA+03]. It also needs constant web connection, as well as a device that is ready to probe for location context.

2.5 Signpost

Signpost [MWBS09] is an application developed by Alessandro Mulloni et al. in order to aid in indoor navigation using camera-enabled devices and tags — 2D bar-codes.

The main motivation behind this project is both GPS's lack of connectivity indoors, and the high cost of populating a building with localization sensors. GPS's connectivity is restricted to outdoor operation since the required satellite links are blocked or unreliable inside buildings. As for location sensing, there are many types of sensors capable of such operation. Unfortunately, they also bring power consumption, wiring and overall infrastructure costs, turning them expensive to place and maintain.

This system operates by scanning continuously the area for strategically placed markers. These markers are not electronic but mere bar-codes placed on, for example, a wall, hence providing location hints. This technique has been covered mostly by in the robotics field of investigation, providing localization and orientation to robots rather than humans. The goal of our project is to adapt the same principles for human indoor navigation and localization.

2.5.1 Architecture

The system's architecture is rather simple. It assumes constant on-line access, not needed for its navigation system, but for extra content provided by the user's localization.

A series of markers are strategically spread across a building. The exact location of these markers is known, and compiled into a map generated by the authors of the application. A new software release is then created, containing the map along with its database of known markers. This way, the system will not need to update its information as it runs, discarding the constant need for on-line access. Later updates to the map may be, however, deployed over the air [MWBS09].

Depending on the user's current localization, updates or relevant information may be retrieved over the air relevant to that specific locale. The project's authors suggest RSS for the content updating.

Signpost features various modes of map visualization, including a 3D view and a 3D level overlay [MWBS09].

2.5.2 General usage

A typical usage pattern was envisioned by the authors as follows: in a conference of some sorts, a user selects a talk he desires to attend. Selecting the talk, the application displays its location on the map. Next, when Signpost detects a marker in the vicinity of the user it displays the user's current position on the monitor, helping him reaching the desired talk. The user may choose among several map visualization techniques [MWBS09].

Signpost has a noticeable impact on the device's battery, since it relies on a live feed of camera images from the phone. It may also consume a high amount of processing power due to the fact that it does a real time analysis to the environment in search of markers — checking at about 15Hz [MWBS09].

Relative to this project, the fact that Signpost is a battery intensive application isn't much of a concern seen the fact that Signpost requires a constant stream of video in order to progressively detect the markers and provide with location hints. Since our application will only rely on a photograph to provide location context information, the impact on the battery will be significantly lower, considering that case alone.

2.6 The Context Toolkit

Context Toolkit is a context-aware project developed by Dey and Abowd [SDA99]. The motivation behind it is that the authors could translate their theoretical knowledge on context-awareness into an application architecture they would see fit.

In order to achieve that objective, they turned into software widgets, due to their apparent success in other software projects. In this project's context, a widget manages the sensing of a particular piece of context. The idea behind this concept is to overcome unconventional sensors and the resulting distributed nature of most context-aware experiments.

2.6.1 Architecture

Context Toolkit's architecture is similar to a peer to peer network architecture, where a widget sits at each node, acting as a peer. Each one of those widgets have a particular state depending on what context information they are monitoring. A widget state is a set of attributes that can be queried by applications. Applications can also register to be notified of context changes detected by the widget. The widget triggers callbacks to the application when changes in the environment are detected.

Beyond that, composite widgets may also be used. In this case, the widget's function is to combine the state of other widgets. For example, a widget designed to detect the kind of activity people in a classroom are engaged in could combine the information provided by presence widgets and activity sensing widgets using, for instance, audio and video analysis. Based on information provided by widgets the composite widget would detect activities such as lecture, group study, exam, etc. [SDA99].

Though being inspired by GUI Widgets, Context Widgets differ on some points worth mentioning: Widgets rely on three kinds of distributed components: generators that acquire context information, interpreters that abstract it and servers that aggregate information. Context widgets monitor environmental information that may be needed at any time by an application. Thus a context widget is active all the time, and its activation is not, as with GUI widgets, driven by applications.

2.6.2 General usage

This architecture has been put to use in some test cases. One of the experiments worth mentioning is the In/Out Board application. The board is used to indicate which members of the office are currently in the building and which are not. Occasionally, everyone feels the need to check if someone else is at the office at a determined time.

Only a single instance of a widget providing information on who leaves and enters the building is required, located at the entrance to the building. Through the use of this widget, the context sensing infrastructure is successful in hiding the details of how the context is sensed from the application developer [SDA99].

2.7 Gowalla and Fourquare

Gowalla and Fourquare are two similar applications that allow a user to share his location with his social group. Their features are quite alike, differing only on some features they provide. They both require the user to check-in at various public places, such as bars, coffee shops, malls, landmarks and such.

Gowalla is a location based social application, created by Alamofire. It was created with the concept of geocaching[1] in mind. The main idea revolved around one being able to check-in at a given location, find out who in one's social circle was or is in that place and receive or drop virtual items — resembling the idea of geocaching. One can also receive physical prizes when checking in at some locations.

Foursquare, on the other hand, was created on the basis of allowing the user to explore and rate locations, based on his experience regarding that place. The concept of checking-in is still present, and a necessary step to rate, comment and leave notes — here called tips — to whoever explores that same location next. This application also features digital prizes such as badges won when the user achieves a pre-defined objective, such as checking-in a certain number of times globally, checking-in at a distant location from the user's home or usual area or checking-in multiple times at a given place.

2.7.1 Architecture

Both services use a client-server architecture in order to function. One may check-in using any mobile device with a mobile connection, or, if a US resident at the time of this writing, by sending an SMS.

Though one may check-in without the need to be online, to access the information stored in a given location one must have online access. Therefore, both these applications need online connectivity to function properly.

2.7.2 General usage

Gowalla and Foursquare are, at the time of this writing, the two major competing geo-location services[2]. Due to the popularization of mobile smartphones such as the iPhone or Android based mobile devices, and the effort of the companies that produce these to services to provide applications for each environment, their user number has been escalating lately.

2.8 Google Latitude

Google Latitude is a Location Based Services application provided by Google. Its main purpose is to share a user's location and let the user find out other users' location [GLA10b].

In order for the user to transmit his location, he must connect to the application URL. Therefore, this application needs online access to update the user's current location. The user may access the application by many ways: location is updated on demand by HTTP access, as he sends an email message using Google's email service, GMail, or as he uses Google's chat service, GTalk [GLA10a].

[1] See http://www.geocaching.com/

[2] See http://gigaom.com/2010/03/21/foursquare-gowalla-subscribers/

Though the main target of Latitude's is the mobile segment, updating the location information in a personal computer, such as a desktop or a laptop, is also possible, by accessing the application's URL or by using Google Gears [GLA10b].

This application aims to take advantage of the user's physical location in order to present an array of contextual services, such as location aware web search. It also enables the user to check the location of friends, assuming they are also using this service, thus providing the user with contextual information.

2.8.1 Architecture

Google Latitude follows a centralized architecture. Each client updates his status to a central location, being a server or a cloud, and each client connects to the same location to check out for information.

In this scenario the client functions as an information consumer. This implies that the client may be a regular user, but may as well be Google, as stated on Latitude's terms of service. This allows for location based services to be provided by google to the Latitude's user.

2.8.2 General usage

Google claims Latitude's users by the end of 2009 rounded 3 million[3]. Though Google reports a high number of users, it is also known that the social circle of the majority of users is null — they have zero friends in the application — so the actual usage of this application is still a disputed subject.

The motivation behind this project remains as unclear as that behind many other Google products. Google's philosophy long has been to aglomerate all the information possible about everything [GOO10], so, without any further knowledge about how the localization information is used, one can easily doubt if this application's purpose is merely ludic to the final user.

[3] See http://news.softpedia.com/news/Google-Latitude-Has-3-Million-Users-141518.shtml

Chapter 3

Technologies

This Section will provide with information regarding the base technology to be used in this project. It starts by describing Android, the application's operating system and programming environment of choice, being followed by a study on 2D bar-codes to be used.

The reasoning behind choosing Android for the mobile application is due to its open source nature and community support, strong support from the community, wide acceptance, and the existence of a complete development environment. Besides that, it's a novelty system with strong support for localization context information.

Ruby on Rails was the choice for the web back-office, due to its outstanding ability to enable a developer to bring up web applications in a quick way.

The project will also include bar-code scanning. Hence, a study on 2D bar-codes is mandatory in order to achieve success. 2D bar-codes were chosen over 1D due to their superior data capacity, as further explained in detail.

3.1 The Android Operating System

The Android OS is a mobile Operating System running on top of a Linux kernel. It was originally developed by Android Inc., later purchased by Google Inc. and lastly by the Open Handset Alliance[1].

Android was designed with openness in mind. It would become the first widespread general-purpose open operating system designed specifically for the mobile device. In the spirit of keeping the entire project open, the Android team has a few design goals to help support this ideal.

Open

Android tries to be especiallydeveloper-friendly, thereby making the device completely open for their applications to utilize. As well, Android is open source and open to implement on most any hardware.

[1] See http://d.android.com/guide/basics/what-is-android.html

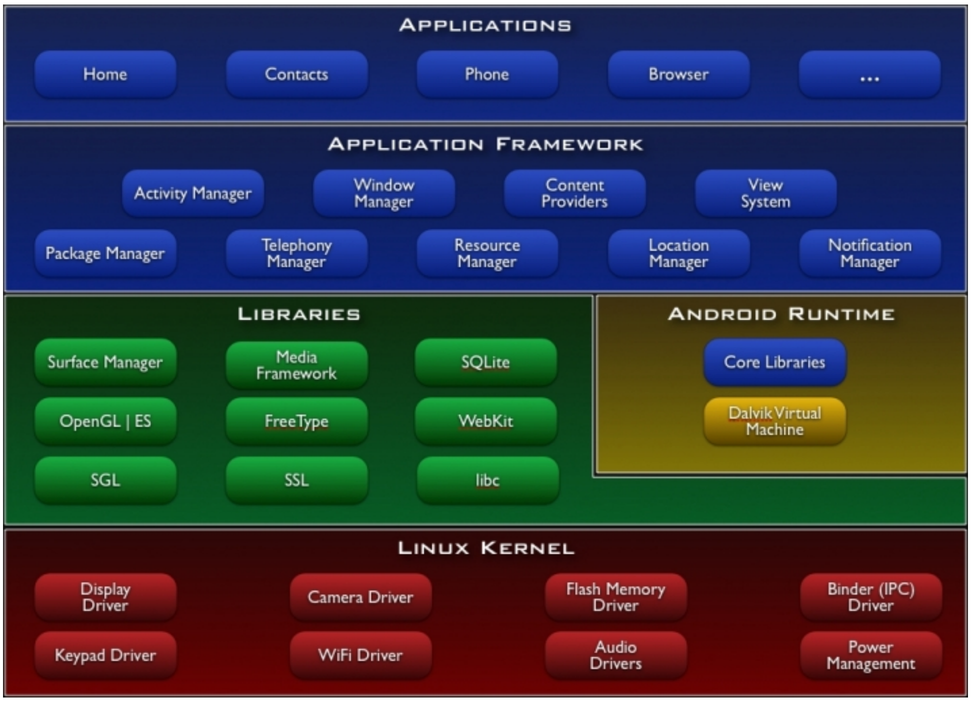

Figure 3.1: Android's layered architecture

All applications are created equal

Android's core applications and third-party applications are on equal playing ground. They share the same API, have the same levels of access, and can do anything that the other does. Any core application can be replaced by any third-party application.

Breaking down applications boundaries

Android does not try to hide any functionality from the developer. The developer should easily be able to tap into the telephony system, GPS system, or essentially any thing that has been implemented in the applications framework.

Fast and easy application development

The applications framework makes it very easy to do some fairly sophisticated tricks like reporting the phone's current location and much more. This speeds up the development process for the developer and makes it easier to focus on design rather than implementation details.

3.1.1 Android OS Layers

Android is subdivided into four main layers: the kernel, libraries, applications framework, and applications. As previously, mentioned the kernel is Linux. The libraries that come with Android provide much of the graphics, data storage, and media capabilities. Embedded within the libraries layer is the Android runtime which contains the Dalvik virtual machine, which powers the applications. The applications framework is the API that all applications will use to access the lowest level of the architecture[2].

3.1.2 The Kernel

Linux was chosen as Android's kernel since it has proved its functionality and stability in desktop systems and in most cases did not require mobile hardware drivers to be rewritten. The kernel provides such things as virtual memory, networking, drivers, and power management. Upon examining the kernel shipped with the Android source code, there are not any significant changes to the core functions of the kernel.

3.1.3 The system libraries

Android's system libraries are all written in C/C++. They provide core Android functionalities and will be called through a Java interface. These includes the Surface Manager — for compositing windows, 2D and 3D graphics — via ES and OpenGL, Media Codecs like MPEG-4 and MP3, the SQL database SQLite and the web browser engine WebKit.

Of particular interest is the Dalvik Virtual Machine which is a part of this layer. The Dalvik Virtual Machine is a bytecode interpreter which is highly optimized for executing on the mobile platform. Running DEX files, which are converted at compile time from standard class and jar files, it allows for more compact and efficient binaries than their class file counterpart, an important consideration when running in a limited memory and battery powered devices such as those Android targets.

3.1.4 The application framework layer

This layer and the layer above it are written completely in Java. Here, Android provides a substantial subset of the Java 5 Standard Edition packages, including Collections, I/O, and so forth. It also provides all of the specific Android OS APIs that the applications will use including things like sharing data, accessing the telephony system, and receiving notifications.

An important point to note about Android OS is that all applications use this same framework. This allows for a better uniformity on the application code, reducing all types of problems related to updating the underlaying layers and application reuse — a flagship in Android OS application development.

3.1.5 The application layer

All of Android's applications are written in Java, which is interpreted by the Dalvik virtual machine. Even the most core features such as the phone and the contacts application reside in this layer. This layer contains software written by the Android team as well as any third-party software

[2] See http://www.youtube.com/watch?v=Mm6Ju0xhUW8

that is installed on the device. An effect of allowing third-party developers access to this layer is that the user interface can be overhauled comparatively easily. Third party applications can handle any event that the Android team's application could see (such as the phone ringing). This means that so long as there is a replacement application for the dialer application, anyone could potentially write their own. Given this model we might expect that, as Android becomes more robust, the user will be able to specify what applications should handle which events.

3.2 Ruby on Rails

Ruby on Rails, also known as Rails, is a "*framework that makes it easier to develop, deploy, and maintain web applications*" [RTH09]. It was created when David Heinemeier Hansson was working on Basecamp, an online project management service owned by 37signals, an american company which creates web applications. Ruby on Rails was part of Basecamp, from which it was extracted and released as an open source framework in July 2004. Since its creation, Rails has become a mature and popular framework, which powers well-known application such as Twitter, Hulu, iLike, Scribd, Basecamp, Shopify and Github [Rai10].

3.2.1 Philosophy

Rails' uses two very important principles that guided the Rails framework and their users, which are *convention over configuration* and *don't repeat yourself* [RTH09].

Convention over configuration
: seeks to decrease the number of decisions and configurations that a programmer has to do. By complying its default behavior and values, programmers only have to specify what is unconventional. Having a set of default values and procedures, Rails' programmers can enjoy its simplicity while not losing any flexibility.

Don't repeat yourself
: means that any information should be located in only one place. Not only this is simpler and easier to do, both in terms of architecture and code, but it's better prepared for the future—programmers can change something without the need to rewrite and reorganize many parts of the application, which done incorrectly may lead to inconsistency.

3.2.2 MVC

Ruby on Rails makes it easier for programmers to create and mantain applications because it imposes the usage of the *model-view-controller* (MVC) architectural pattern. This way, applications are better organized, with everything in its right place. Every Rails application is split into *models*, *views* and *controllers*.

Model
: A *model* is the part responsible for maintaining the state of an application. Most times, this

state is stored in a database. Besides maintaining the state of the data, it sets *constraints* on that data, e.g. a blog *Article* which must have a *title*. By keeping these constraints in the model, there's no way to break them anywhere throughout the entire application, so the data will always be consistent and valid.

View

A view generates what is displayed to users—a user interface. This is usually a visual representation of a model, but a model may have multiple views. For example, a blog will have many articles that are accessible through the model and displayed using a view. A view can have multiple formats, e.g. a RSS feed, and is used exclusively for output. Although a view may provide users with ways to input data, it never handles that same data.

Controller

The controller *controls* the entire application. It handles events from users, interacts with the model, and renders a view. In Rails, requests pass through a routing component before reaching the controller. This component selects the appropriate controller based on the requested URL.

3.2.3 ORM

Object-relational mapping is a programming technique used for abstracting database access. ORM libraries map database tables to classes. This way, programmers can use databases as though they were using simple classes, without worrying about data type conversions and database queries. A table is mapped to a class, a row is mapped to an object, and a column is mapped to objects' attributes [Amb10]. So, if a programmer wants to change the name of the first user named *John*, he may do

```
johns = User.all(:name => "John")
john = johns.first
john.name = "Johnny"
john.save
```

The first and last lines of code are mapped from Ruby code to a database query. In the first line, the ORM maps the *selected* table rows to `johns`, an array of `User` objects. The last line *updates* the name of the first user to "Johnny".

Ruby on Rails includes its own ORM library, called ActiveRecord [RTH09], which also adds support for table constraints, triggers, relationships, inheritance and much more so that the programmer doesn't have to bother with complex database queries.

The first line of code is also an example of the philosophy of Rails. When the database schema file is created, the above code sample works even if the User model is defined as

```
class User < ActiveRecord::Base
end
```

One of the many Rails *conventions* is that database table names are in the plural form of model names. The model class `User` works because the table is named `users`. Because Rails *doesn't repeat itself*, the `User` model knows that a `name` field exists because that's already defined the database schema file.

Rails and ActiveRecord are capable of working regardless of the database system in use if a database adapter is available. Available adapters include MySQL, SQLite, PostgreSQL and Oracle.

3.3 Notation and data interchange formats

In order to echange information in this work, a quick survey on popular existing data interchange formats was performed.

3.3.1 XML

XML is a text-based general-purpose markup language. It's considered an extensible language, because it allows users to create custom markup languages. It's fundamentally important on the Web, where it's used for data exchange [W3C10]. In the context of Ruby on Rails, it's used for providing and consuming web services. Take for example this code sample:

```
<?xml version="1.0" encoding="UTF-8"?>
<user>
  <id type="integer">1</id>
  <name>John</name>
  <age type="integer">34</age>
</user>
```

This code is very easy to read, and represents a user named "John", aged 34 and whose *id* is 1. As XML is supported by most programming languages, it's a common format for an API that want to broaden the potential set of third-party applications using its services.

3.3.2 JSON

JSON is a data interchange format, based on a subset of JavaScript, a popular client-side scripting language. It's a text-based very lightweight alternative to XML for data exchange [JSO10a]. Take the following code as an example:

```
{"user": {"id": 1, "name": "John", "age": 34}}
```

Compared to XML, JSON is simpler and smaller in size. Unlike XML, JSON doesn't need to specify the data type of some attributes — being a subset of JavaScript's object literal notation, a value can hold strings, numbers, booleans, a null value, other JSON objects, or arrays composed of any of the previous — but it isn't extensible, so it's limited to these types [JSO10b].

JSON is rising in popularity for simple data exchange when extensibility isn't required. JSON code is easy to read and write, and much smaller and faster to parse when compared to XML.

3.4 2D bar-code comparative analysis

In order to properly choose the right bar-code to use with this application, a study was made regarding their types and characteristics. Below are the most prominent candidates as well as details about them, followed by a small conclusion on a possible choice.

Having their data capacity into consideration, 2D bar-codes may be divided into two distinct groups: database bar-codes and index-based bar-codes. The latter features less data capacity and is primarily used for inserting a distinctive identification number in a taggable object, resembling their 1D counterpart, whereas the first may hold more significant data, hence being called database 2D bar-codes.

These kind of codes can also fit into another two categories: stacked and matrix-based. Stacked symbologies consist of a given linear symbology repeated vertically multiple times, whereas matrix symbologies feature square or dot-shaped modules arranged on a grid pattern. The following research cover only matrix-based bar-codes, as their versatility gives them the capability of being captured and decoded at very odd angles, making them preferable for the use cases foreseen for this project. Also, they tend to have a higher capacity than the stacked ones.

3.4.1 QR Code

QR code is a 2D design developed by Denso Wave around 1994. Its main objective was to improve the overall reading speed of existing bar-code implementations. It accomplishes that by using distinctive position detection patterns, located at three of the symbol's corners, as depicted in Figure 3.2. This enables readers to quickly detect the bar-code, resulting in a snappy reading from any angle, which can be an advantage for use with mobile camera phones [KT05].

Being a database bar-code, it may encode all types of data, including symbols, binary data, control codes, and multimedia data. Its maximum data capacities are 7,089 characters for numeric data, 4.296 characters for alphanumeric data and 2,953 bytes for binary data. QR codes may also vary in size, ranging from version 1 — with a size of 21×21 cells — to version 40 — with a size of 177×177 cells. Former maximum capacity refers to the bar-code's version 40. Due to size and reader limitation, cell phones generally use versions 1-10 of the bar-code, thus reducing its actual maximum capacity to 652 characters for numeric data, 395 characters for alphanumeric data and 271 bytes for binary data, when applied to this kind of problem[3].

QR code also features error correcting codes, in order to allow the symbol to be read even if damaged, in 4 levels of error-correction. The higher the level, the more space is dedicated to error-correcting in order to preserve data integrity. This presents as a trade-off problem as we might

[3] See http://www.denso-wave.com/qrcode/qrgene2-e.html

Figure 3.2: A QR Code sample

be able to choose between more information or the guarantee that it will last. The 4 different levels are L (consuming approximately 7 percent of the data), M (approximately 15 percent), Q (approximately 25 percent), and H (approximately 30 percent) [KT07].

Google Inc. uses this format for their Android Market and Favorite Places applications[4].

3.4.2 Data Matrix

A Data Matrix code is a two-dimensional matrix bar-code consisting of black and white square modules arranged in either a square or rectangular pattern. The information to be encoded can be text or raw data. Usual data size ranges from a few bytes up to 2 kilobytes, depending on the symbol dimension used. Data Matrix's symbol allows a read from all angles due to its finder pattern comprising two solid lines and two alternating dark and light lines on the symbol's perimeter, which is surrounded by a quiet-zone border [KT07].

Data Matrix's symbol size ranges from 10×10 modules to 144×144 modules, where it reaches its maximum capacity. Sizes commonly used range from 10×10 to 40×40. In this last scenario, Data Matrix has a maximum capacity of 288 characters for numeric data and 214 characters for alphanumeric data [BHO+09].

There are two types of error correction algorithms available in the Data Matrix specification, which can be chosen depending on the error checking and correcting level employed. ECC levels 000 to 140 offer five different error-correction levels and use convolutional code-error correction. However they have recently become deprecated. ECC-200, which is Data Matrix's last version, commonly in use, uses Reed-Solomon error correction. The symbol size determines ECC200's

[4] As stated on http://www.google.com/help/maps/favoriteplaces/business/barcode.html

Figure 3.3: A Data Matrix sample

correction level: a 10×10 symbol has a 62.5 percent of codewords dedicated to error correction, and a 40×40 symbol has about 30 percent [KT07].

Semacode[5] has adopted the Data Matrix format to encode plain-text URLs and implement them as physical hyper-links. An example of a Data Matrix 2D bar-code can be seen on Figure 3.3.

3.4.3 VSCode

VSCode, by Veritec, is a small, checkerboard-like matrix capable of holding more than 4,151 bytes of data. Its elements consist of a solid border frame, which serves as the outside border of a data matrix field — not to be confused with the Data Matrix bar-code. Within this data frame is the internal data field, comprised of single cell units representing binary data. There is an invisible border surrounding the data field called the “quiet zone”, a border having the width of at least one data cell. As with other bar code implementations, this zone is critical to separating the data field from surrounding image noise [KT05].

Veritec also offers additional security options through custom designs that use unique encryption techniques. So, VSCode may be used with other electronic media, such as an RFID tag or a smart chip, for data access and storage [KT07].

VSCode stores up to 4,151 bytes of data, when at the size of 192×192 cells. Due to its cell density the bar-code does not grow as big as other 2D bar-codes when at this cell size, thus becoming the most compact 2D bar-code. It is also the one that holds the biggest data capacity. Furthermore, it can be either square or rectangular in shape to make it fit the space available[6].

[5] See http://semacode.org/about/technical

[6] See http://www.veritecinc.com/code_comparisons.html

Figure 3.4: A VSCode sample

Regarding error detection, VSCode uses Reed-Solomon encoding, having a high percentage (15 to 25 percent) of error detection and correction (EDAC) capability. This lets it restore all the encoded data even if up to 35 percent of the symbol is damaged [KT07].

A depiction of a VSCode can be observed in Figure 3.4.

3.4.4 Readability tests

A readability test performed by Kato et al. [KT07] hands out interesting results comparing these three bar-code standards.

Their method is based on calculating the FRR (First Read Rates) of each one of the symbols in certain lighting conditions. Four samples for each of the three 2D bar-codes were created in order to do this test: two samples using different symbol sizes (2.5cm and 5.0cm) and the same sizes using different data to encode. For the QR Code and VSCode samples, error correction level at 15 percent was used in order to reflect the most common setting regarding their use. The Data Matrix error correction levels are automatically set, so they were 28–39 percent for the first data sample and 22–34 percent for the second data sample.

The data to be encoded in the tests are two URLs: "http://www.scis.ecu.edu.au/current/" and "www.k.st/". The photo shots were taken by two camera equipped phones: a Nokia 6600 with a VGA camera and a Nokia 6630 with a 1.3-megapixel camera. This difference in camera resolution let the testers verify if there was any relationship between camera resolution and better FRR.

The shots were captured at a distance deemed reasonable by the authors — between 5 and 25 cm away from the target. The illumination was carried by Cold Cathode Fluorescent lights under three lighting conditions: half power, full power and using ambient lighting only.

Two decoding algorithms were used to decode Data Matrix symbols, as well as QR code symbols. For Data Matrix, Semacode's reader and Kaywa Reader were used. For QR Code Quick Mark and Kaywa Reader were used. As for VSCode a PC based decoder was used as no smartphone decoder was found.

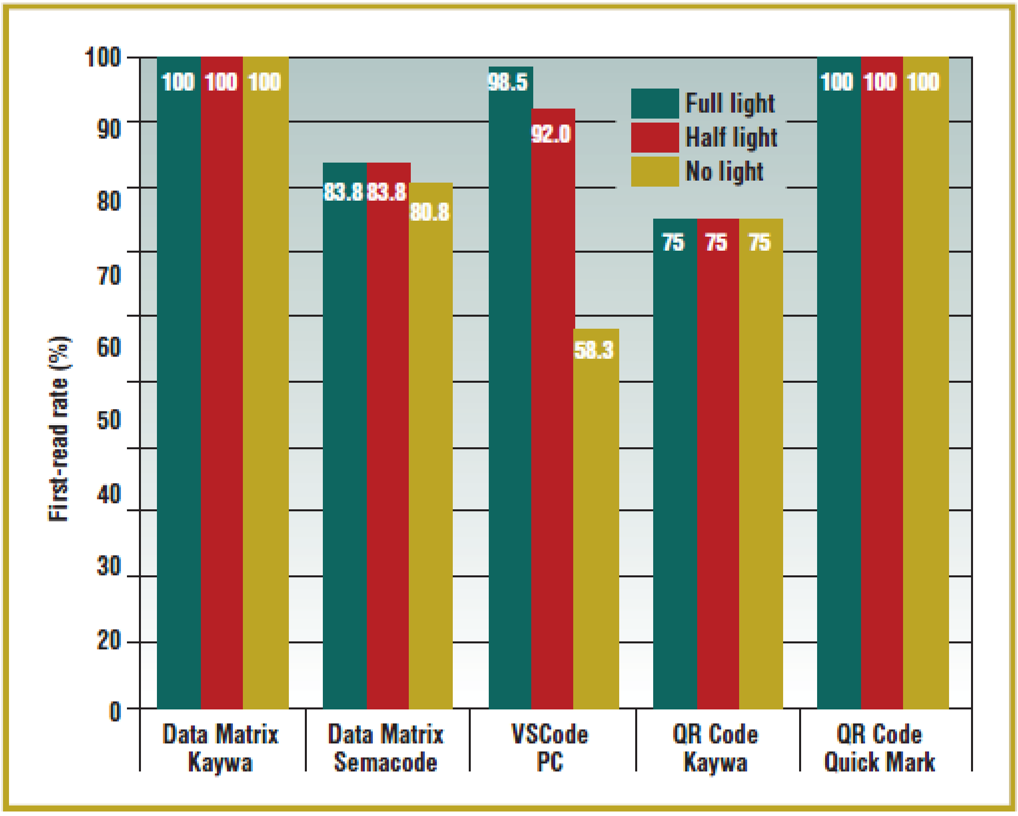

Figure 3.5: Results of the readability tests

Unfortunately no open source alternatives were used in the decryption of the symbols. However, as this is a readability test, its purpose is to provide with valuable hints about symbol readability, in which it performs rather well.

As for result conclusions, from Figure 3.5, adapted from the work of Kato et al., we may derive the following:

- QR Code and Data Matrix's performance was solid. FRRs of the Quick Mark QR Code reader and the Kaywa reader used for decoding Data Matrix reached 100 percent, and their lowest scores with Kaywa Reader and Semacode respectively were rather high so we might assume they'll behave well on a real world use basis.

- While VSCode's performance was nothing but very good in good lighting conditions, at ambient lighting it tends to fail. Since this project's requirements include outdoor tag scanning, this proves as an impediment.

- Data Matrix and QR Code don't seem to suffer from this poor lighting condition, performing rather well in all occasions.

Also, deriving again from the work of Kato et al. [KT07], some other relevant conclusions were taken from the test:

Table 3.1: Bar code comparison table

Characteristic	**QR Code**	**Data Matrix**	**VSCode**
Max. Capacity[a]	395 characters	214 characters	4,151 characters
Readability tests[b]	Very good	Very good	Satisfactory
Property rights	Public Domain	Public Domain	Property of Veritec Inc.
Wideness of use[c]	End-consumers	Heavy industry, end-consumers	Heavy industry
OS[d] *implementation*	Yes	Yes	No

[a] Max alphanumeric capacity relative to this project's context. In this analysis symbol sizes superior to those reasonable for mobile phone scanning are not considered.
[b] Readability test results according to Kato et al. [KT07].
[c] According to Kato et al. [KT05] [KT07].
[d] Open source

- Camera resolution is statistically irrelevant. Both the VGA and the 1.3MP camera produced shots that were decoded with very similar FRRs.

- Symbol size allowed for further symbol readings. 5cm symbols could be read from a further distance than the 2.5cm.

3.4.5 Public domain algorithm implementation availability

Open source algorithms are available for both Data Matrix and QR Code symbols. One of particular interest is zxing[7], standing for Zebra Crossing. Zxing is a multiple bar-code format scanner that supports both the Data Matrix and QR Code symbols. Besides that, its written in Java and completely Android compliant.

There are other Java libraries that support the above proposed formats, namely open source QR Code Library[8]. This library is also used by many developers, so it was also focus of investigation.

Even other libraries have been found capable of decoding the QR Code and Data Matrix format, such as libdatamatrix C, libdecodeqr C++, libqrencode C, PyQrCodec, pyxing. Though interesting for future reference, they're either written on another language or lack features. One would prefer to use a COTS component to implement this feature in this application, since this is not the focus of this project's work.

As for VSCode, no open source algorithms or compatible closed source components were found. Kato et al. [KT07] had already no luck finding an implementation of this symbol's decoder and seemingly the situation hasn't changed in the last years.

Concluding this investigation, it is found that zxing is a strong choice in order to implement the decoding of the proposed 2D bar-code symbols.

3.4.6 Bar code showdown

Taking into account all the condensed data in Table 3.1, our choice relies on the QR Code. Not only it performs rather well on the readability tests, it has a large and meaningful user base, a modest capacity which hopefully will allow to store the information wanted, and, most importantly, is a Public Domain property with a vast number of open source implementations. This last factor will ease the development of this project.

The second choice would be Data Matrix. The key factors of this decision are the reduced storage capacity and the lesser use among end user consumers.

VSCode stands as a last resort solution. Although it has a higher data density, it is that same factor that makes it less legible, besides its proprietary license implications.

3.5 Conclusions

While it is tempting to go for a complex model in order to implement this project, such complexity becomes overkill when considering the context data that is going to be managed. Complex models such as ontologies coupled with a non-layered architecture are fit to handle a large variety of context information, not to mention the heavy workload they drop on the system.

However, it's good to have in mind a possible scale on the system in the future. So adopting the simplest of models just because of its initial requirements might not be the best route to take. A possible paradigm shift may cause this application to accept other contextual data not related to web access, such as time context, or even location context based on ubiquitous free systems, such as GPS signaling.

Taking into account the specificity of the devices to be implemented, especially when discussing Android OS specifics, one can notice part of the architecture is already predefined by the API design. Even if wanted, implementing a direct sensor data access model would be impossible, due to constraints imposed by the OS. That will be analyzed in more detail further on.

[7] See http://code.google.com/p/zxing

[8] See http://qrcode.sourceforge.jp/

Chapter 4

Approach

This chapter focuses on explaining both the problem at hands and the solution proposed to solve it.

The problem is hereby explained with more detail in order to contextualize the reader before reading a possible solution, so that the reader may identify more clearly the scope of the solution.

Next, the solution is presented, divided into different stages. Each stage proposes improvements incrementally, so a sequential read is recommended. The solution is presented at a conceptual level, in this chapter. A hands on approach is done in the next chapters.

4.1 The focus of this investigation

The main problem consists on avoiding the need to access the Internet in order to get content based on the user's location. The whole idea revolves around building a mobile application platform, so space and computing power are somewhat limited to how mobile devices currently perform.

The issue caused by the lack of space and computing power turns the process to find a feasible solution more challenging. Certain precautions should be considered before transmitting content, along with the effort the mobile device will have to undertake in order to generate it.

As stated earlier in Section 1.1, the motivation around this project is based on the fact that Internet access is not yet ubiquitous. Though mostly available in highly populated areas, such as cities and other permanent settlements, online access is still deficient in lesser populated areas, such as country roads, small villages or desert areas. Should one try to access knowledge stored online in those areas, one may find himself held back by this fact.

Mobile connection to the Internet is also somewhat limited and costly, even for the user with a data plan. Each day the volume of information mobile users consume from the Internet increases, where operator's data download limits don't. This project aims to reduce that trend in some situations where the same information is repeatedly fetched, due to its often off-line unavailability.

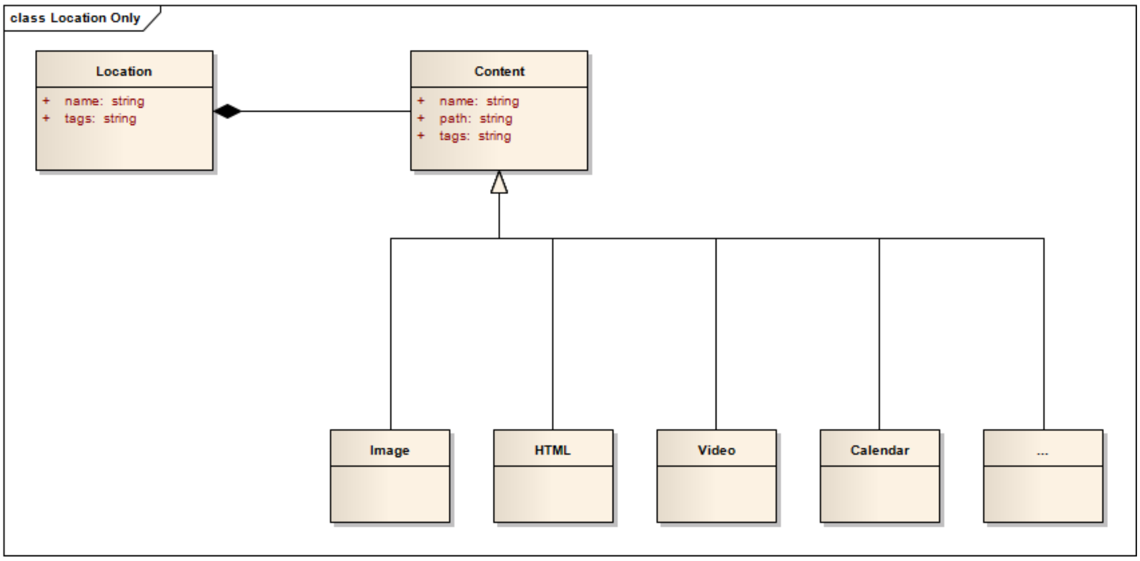

Figure 4.1: The relationships between a Location and its Contents

Moreover, in many situations, mobile data plans present extra costs to the user, such as roaming charges. This scenario happens quite frequently when the user is travelling abroad, often to locations unknown to him that most certainly will require the user to consult relevant location based information to navigate.

Another problem this project tackles is the inability for the end-user to create new content for these kind of platforms, most of the times. This is somewhat a novelty in these kind of systems. Usually content format is closed, and new content propositions are sent to the controller of the closed system for moderation and integration. This project proposes a model that allows user-generated content, thus becoming a location based open content platform.

The following sections address a possible solution for the problem described. Firstly some concepts related to the solution should be defined, such as what the words “location” and “package” represent in this context. Then, the issue of how to locally store and fetch content is tackled.

4.2 Defining Location and Package

From this point on, that concept of location will be key. Each location means an exact spot on the world. Its granularity may vary: one may say a whole building is a location, or a small spot near a vending machine.

Each location may provide the user with a variety of contents — both in number and in type. A content object may be an HTML or PDF document, an image, video, calendar, map or simply a pinpoint of the user’s current location.

In this project, the concept of package represents a collection of locations and their respective contents. Along with that information, a table that translates the position received by the platform

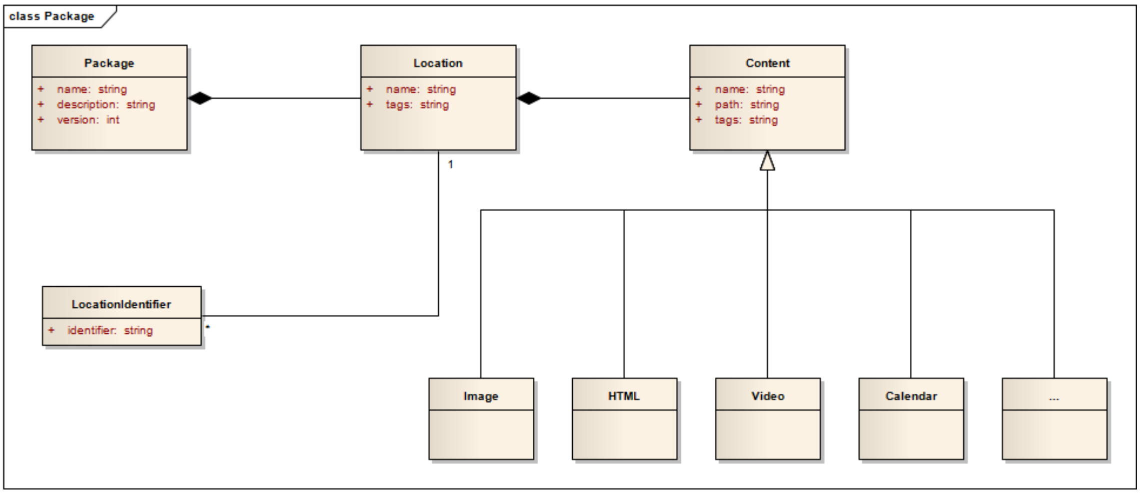

Figure 4.2: The relationship between a Location and the Package it is inserted in

into a location should also be appended, to support multiple pinpointing methods should be used (both GPS signaling and QRCode tagging, as an example).

Location naming should be unequivocal. Each location should have a unique identifier, that solely pinpoints to that specific item. This is to prevent name clashing when fetching a specific location using the location's unique identifier as the search token — e.g. when not using GPS coordinates to search for the right package. In fact, when using tags such as QRCodes to identify locations using this system, the contents encoded on the tag should be the location unique identifier.

A more detailed look at a recommended package naming schema is described ahead. In order to better understand that recommendation, some concepts about this proposed solution should be learned beforehand. Next sections will include those concepts.

4.3 Content caching and access

The mobile device user should be able to access content without requiring a connection to the Internet. Therefore, content should be pre-loaded in the users device, stored in a local database, prior to its consumption.

Access to the content happens when a location event is triggered. Similar to a database, the platform receives a request bearing a location identifier — that may be GPS coordinates or a unique key coming from a bar-code, and responds displaying the contents stored in the local database regarding that same location.

4.3.1 A use case scenario regarding content access

As a possible usage scenario: some user fiddling with an application built on this platform finds itself in Oporto's downtown. He has already pre-loaded some information regarding the city's

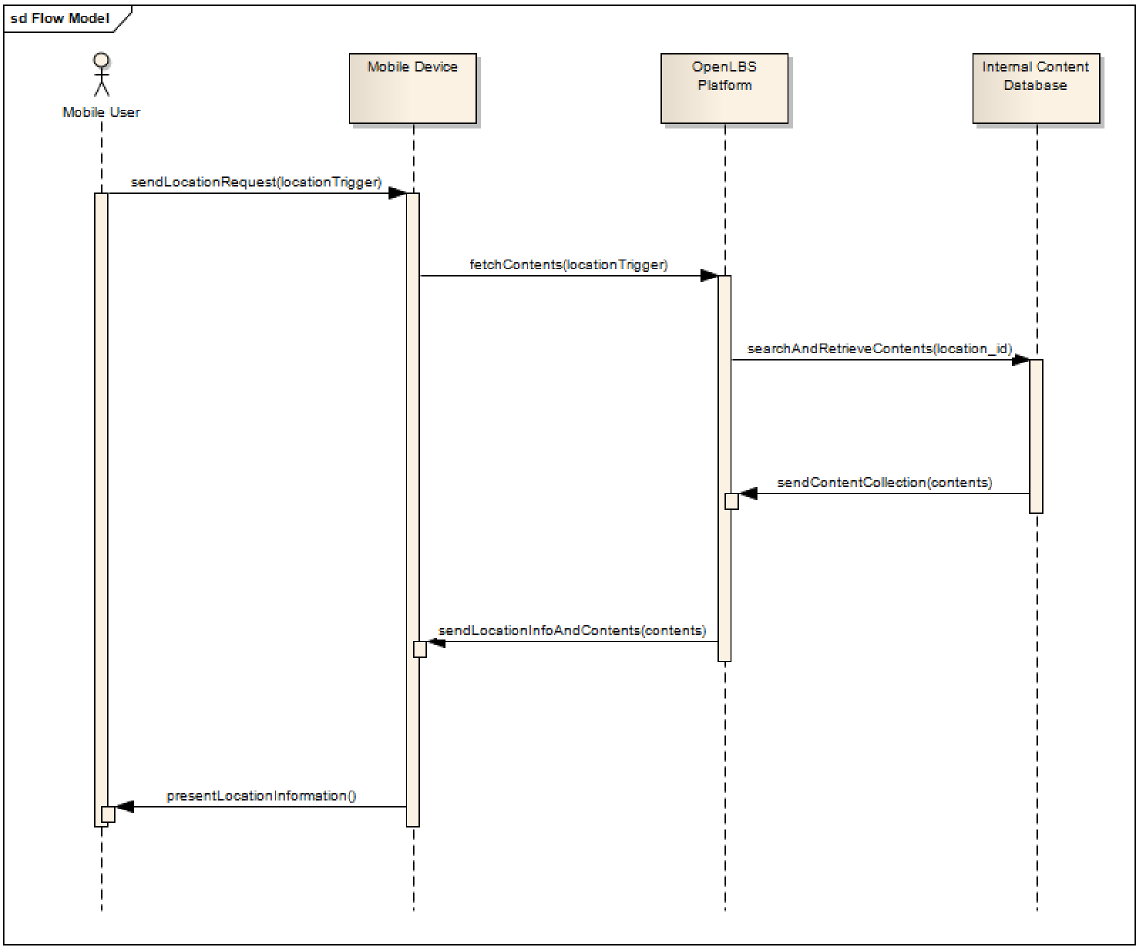

Figure 4.3: The process of content fetching in the local mobile database

points of interest, and now faces a landmark of some sorts. Feeding the platform some kind of information on the location — let it be the user's GPS coordinates, a reading of a QRCode placed on the landmark or even a picture of the location, the platform will return all the information it possesses about that location. That may include historical facts, the user's location on a provided map, other pictures of the landmark, and so on.

4.4 Content delivery

Content should be distributed to the user prior to its use, remain cached in the user's device and update as necessary, when a connection is available. It should also be organized respecting its location context.

Regarding these kind of requirements, an approach based on existing Linux package distribution systems will be followed. A package management system is a collection of tools to automate the process of installing, upgrading, configuring, and removing software packages from a computer. Linux systems are popular with this kind of architecture, having successfully adopted it, becoming the most prominent users of this type of technology.

In a similar fashion, a user would pre-install a package from a repository organized by location and content, where the user could connect to the Internet. Afterwards, all that information would be available off-line per user request.

An example of a real use case package is one describing various locations at the Faculty of Engineering of the University of Porto. Downloading this package should enable the user to know details about locations inside FEUP, such as timetables in the classrooms, instructions on how to use the printing system near any printer, their location on FEUP's map in any location, and so on.

A package may suffer updates, additions or deletions. Also, new packages may be added into the system. Therefore, a package manager is be included in the platform, in order to manage package transactions.

4.4.0.1 The package manager

The system here described features an integrated package manager, to deal with content package transactions. It is its responsibility to properly install packaged contents and make sure they are kept up to date.

Whenever an internet connection is available, the package manager should ask the user if he would be interested in checking for updates to the packages he has got installed on his system. The user may also proceed to do this manually, or ask the package manager for a list of packages available for install or removal.

In order to maintain all available packages, as well as to provide a service that allows access to them, a repository should be implemented. It will be that repository, remote to the user, that will contain all the packages available on the system, as well as all the data. Users may connect to this location whenever an internet connection is available in order to add or update packages, as well

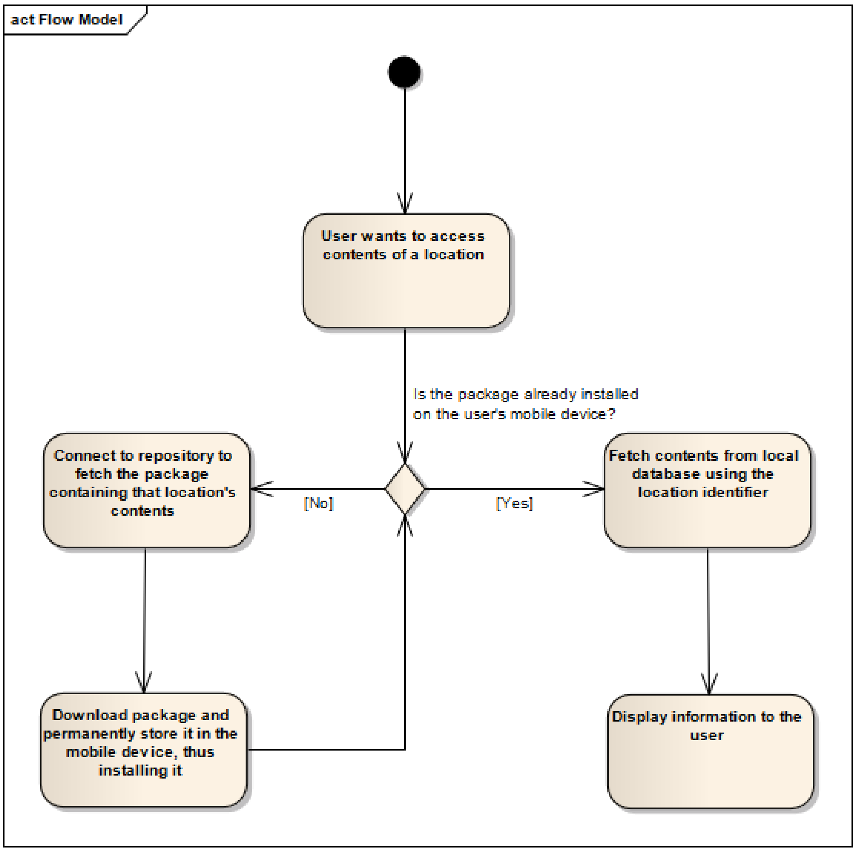

Figure 4.4: Interaction provided by the Package system

as to update their knowledge of all the existing packages — check if new are available or if some they possess have been purged.

Package retrieval, update and installation should be fully automatic. The user should only concern himself on what packages will he need.

4.5 Content publication

To deal with the kind of operations involving package building and generation, which a regular user is not familiar with, a back-office should be implemented, automating the complicated bits of this interaction with the user.

This back-office is tightly coupled with the main package-repository. They might even be the exact same application, offering, as an example, web services both to manage the creation of new packages and to offer mobile devices package installation and updating.

Package contents should follow a strict format when being conveyed to the user's mobile device, so that it may recognize and know how to use them. They may be stored freely of formats in the repository/back-office solution, though, and be packed upon user request — as part of a package retrieval operation, for example.

The main objective of the back-office is to ease the production of new packages to the end-user, so that he may produce new content in a convenient and hassle-free fashion. In order to achieve such endeavour, a certain degree of abstraction is to be presented to the user: he should not have to know the exact format the content packages follow in order to create one.

4.5.1 Recommended location naming schema

A unique location identifier should be used whenever possible as default triggers for content retrieval on the mobile platform when using tags — QRCodes — as physical location identifiers.

As a practical example, a recommendation on a package naming schema is devised. The unique location identifier (ULI) will follow standard URI syntax, as stated by The Internet Engineering Task Force's RFC2396 [URI10].

Assuming the system is using web services to offer the users content packages, the schema should be "http" or "https" if using SSL for content encryption between repository and mobile client. Any other can be used, but as a matter of convenience, this is recommended, as general mobile users may not have knowledge of this platform when scanning a tag with this information and when accessing the UPI as a simple URI for a web-page, an explanation would be presented.

Should a system have the ability to support multiple repositories as content package sources, the location field of the ULI should match the location of the repository from where the package owning that location originates. If the supposed package was retrieved from "rep.openlbs.org", then the ULI would at this point be "http://rep.openlbs.org/".

The path fields in the ULI should reflect the existing package-location hierarchic model explained on section 4.3. The unique package name — within each repository, if that's the case — should come first, followed by the location unique identifier — within each package. If, for the same hypothetical ULI referenced earlier, we were referencing the location “RoomD007” inside the package “FEUP”, then the final ULI would look like: “http://rep.openlbs.org/FEUP/RoomD007”.

If that location were to be recognized by placing a QRCode in its vicinity, the QRCode’s content should be the ULI mentioned above.

4.6 Content adaptation

Size is obviously a concern in these kinds of scenarios. So, a very lightweight data format should be used in order to convey information to the user. Ideally, images and video should be resized according to the usual screen resolution and compressed, in a general fashion.

There is a wide array of mobile devices available to the general public today, each one with its unique features and abilities. As these differ between brands, even between models, one may not assume that a specific data format will be able to be consumed in all the equipment. Therefore, beyond optimizing content for size, there is a need to make sure this platform doesn’t monopolize all the free space on a mobile device, as an example, or include content that a certain model cannot make use of.

Here is an example of how this could be explored. Three distinct users possess three different mobile equipments:

- User A has a state of the art smartphone, capable of playing various video formats, browsing html and pdf documents, and even handle more complicated formats, such as a calendar. The device has a large, colorful display, storage capacity to spare.

- User B has a feature phone, capable of displaying some image formats and playing a limited set of video formats. Its storage capabilities are limited, and so are its processing abilities. The device features a color display, but rather small in size.

- User C has a mobile device capable of accessing the Internet. Its storage capabilities are adequate, but it features only an electronic paper display capable of displaying black and white content.

Content packages should be adapted to each one of these devices. If, say, a package contains a video, that video should suffer the necessary modifications to run optimally in the mobile device, unless specified otherwise. In this particular example case, providing the content was a high resolution video, user A would have a near original experience, seen that his mobile device has no problem handling large volumes of content and has a display that will take advantage of all the video’s resolution.

User B would get a downsized version of the video mentioned, due to the nature of his mobile device. There is no point on sending the full fledged video to his mobile device if it would fill up all its capacity and look the same when compared to a stripped down version.

User C wouldn't be able to download the video at all. As his display supports very poorly content that requires animation, there would be no point in wasting bandwidth downloading content that the user wouldn't be able to see.

As some contents are sensitive to resizing or transcoding, this would be a completely optional feature. A content bearing a high resolution image could be, for example, a map. Though user B owns a device that wouldn't be able to see all the map's details when fitting the image to its display, he might want to zoom in in some parts.

For the content to be adapted this way, an additional effort must be supplied by the back-office system. How and when that content is adapted is described in the next subsection.

4.6.1 Content adaptation methods

Content adaptation requires some consuming tasks like resizing images, or transcoding video. For that same reason, this task is left to the repository system.

The first issue to have in mind is that of the knowledge that the repository has about mobile devices. In order to produce adapted content, the repository must know what it will adapt content to, and that will require it to maintain a database containing the specifications of known mobile devices.

The full scope of that database is subject to some debate. It would be desirable to have a set of pre-compiled packages ready for download for each of the mobile devices that existed, but that would lead to many packages being left unused, leading to an unnecessary waste of space and processing power every time a new package or package version would be generated. Besides that, building such a database would require an incredible amount of work.

Another possible method of achieving content adaptation would be to generate the package as it is being requested, on the fly. That would deal with the issue of the waste of resources by the repository when generating packages. However, generating a single package for a single mobile device would be a heavy and long operation, which could result in many delays when downloading a single package. It could even result in a small unintentional Denial of Service attack on the repository if many mobile clients downloaded packages at once.

Combining the two solutions presented above, it's possible to achieve one that tries to take advantage of the best of both parts. Using a "package cache" approach, the repository could learn which mobile clients download certain packages and provide only those when generating new content, or updating. It could also check when a certain type of mobile client downloaded something for the last time, so it may discard the profile regarding that mobile equipment if it wasn't used for long.

Generation of the database containing mobile client profiles would be done on the fly, as clients connect to the repository. Each time the mobile application connected to the server it would send relevant data about the environment it is running on, so the repository may consider that information. The mobile client may include that information by piggybacking a GET request, in its Content field, or perhaps in the UserAgent field, should the access to the repository from the mobile client be on top of the HTTP protocol.

Chapter 5

Prototype

Following the research and specification of a solution for the problem this investigation addresses, a small prototype was developed to check the feasibility of the solution presented.

This prototype uses QRCodes to uniquely identify a Location, but can easily be extended to use other available geo-location data.

The prototype has two main components: an Android OS application that implements the features designed for the mobile client, and a Back-office that functions as a content generating platform and a Package repository.

5.1 Development methodology

Rapid application development was the methodology chosen to properly develop this project. Among various advantages when using this method, the following are highlighted when in this project's context:

Application Modeling
: The application modeling phase occurs, as in a traditional fashion, in the early stages of development. This modeling is done recurring to standard methods, establishing well-defined application processes and data. However, this specification is subject to change as development ensues, making it ideal for an experimentation as this project's.

Rapid Prototyping
: Android's API allows for rapid prototyping using Java's interfaces and abstract methods and XML for quick interface designing. This should prove sufficient to build mock-ups or functional prototypes for case studying.

Application Generation
: RAD promotes application code generators, so that an application scaffold can be generated on the fly, easing the process of prototype development. Although there are none available

at the time of writing for the system specified, its high level architecture allows for quick prototype and mock-up design and develop. It's also focused on re-usability, a key point on this methodology's philosophy.

Component Testing

Since the RAD process emphasizes reuse, many components are already tested, thus reducing the total time of testing. However the new components should be tested and all interfaces must be fully exercised.

5.2 Top-down architecture

This section describes the basic architecture of this prototype.

5.2.1 The two main prototype components

This prototype is divided into two main components: the back-office and the mobile application. These are designed to communicate in a client-server architecture, where the back-office plays the role of the server and the mobile application the client. An introductory note about each follows:

OpenLBS Back-office

This component serves as both the Package generator and publisher. Users may design new packages and retrieve available ones by accessing this component of the prototype. It is basically a web application exposing some web services in order for both users and mobile applications to access it.

OpenLBS Mobile Application

The mobile application is a prototype that implements part of the concepts explained in Chapter 4. It connects to the back-office in order to retrieve Content Packages, and its updates. The rest of its operation is done entirely offline.

Both components are described in detail in the course of this chapter, providing explanations on how they work and how they interact.

As far as this prototype is designed, there may be multiple clients accessing the same back-office, but a mobile client may only access one back-office at a time.

5.2.2 Information architecture

Figure 5.1 depicts the class model as its used on both the components of this prototype. Though based on the same class model, database schemas vary slightly between the OpenLBS back-office and the mobile application. Those differences will later be explained in the section regarding each part.

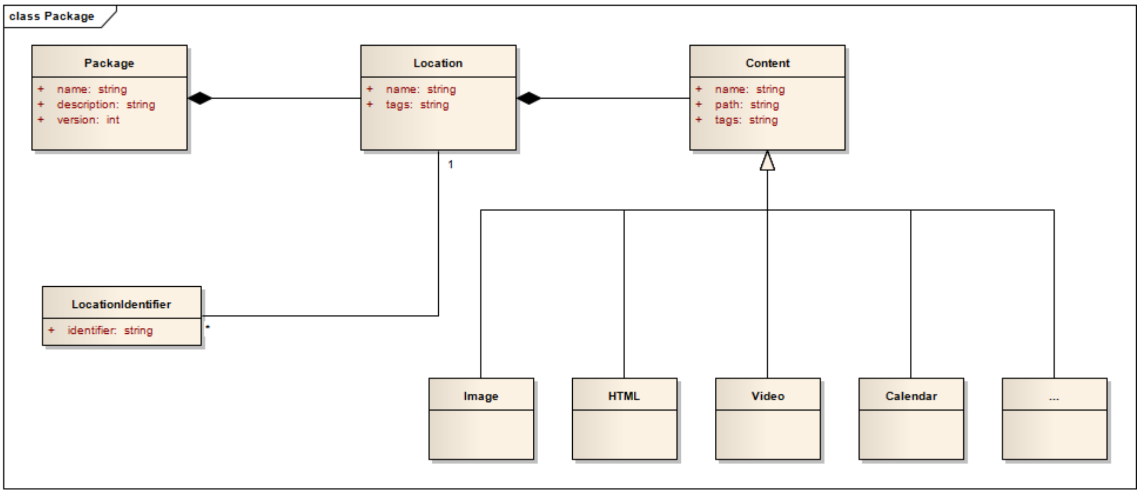

Figure 5.1: The prototype's information architecture

5.2.2.1 The Package class

A Package represents a collection of Locations, thus being related to the Location class in a composite relationship.

Each Package instance has the following attributes containing meta-information:

Name
: The name the user who generated the Package gave it. It is fully textual, and can contain only alphanumeric characters, without spaces.

Version
: The package version. This is an incremental integer that allows the application to check for updates to that same package. The rule followed in this prototype is quite simple: between two versions of the same packages, the one with the higher package version number is the most recent.

5.2.2.2 The Location class

A Location instance represents the relation between a set of contents and their geographic location. Each Location instance also references a geographic location itself, through the LocationIdentifier class.

Attributes present on the Location class describe the following:

Name
: The name given to the Location by the user who created the Package it resides in. This name can contain only alphanumerical characters, and no spaces. It also needs to be unique to each Package, in order to avoid name clashing when identifying a Location by its name using the ULI described earlier in Section 4.5.1.

Tags

A set of comma separated tags, which serve as meta-information about the geo-location itself. Their main aim is to better characterize the geo-location this Location represents. An example would be adding the tag "atm" to a Location referencing an ATM machine. These will help to find relationships between this and other geo-locations, which will enable some navigation features this application can perform, such as finding an ATM machine in the vicinity of another Location where the application's user would be at that time.

5.2.2.3 The LocationIdentifier class

There is a need to map different types of geo-location identifiers, such as the ULI — described in subsection 4.5.1 — or GPS coordinates, to a specific Location. This class implements that set of relations, by taking a single attribute to identify the Location it is related to.

This way a Location may be retrieved using not only one but several identifiers. One may use our GPS location or a ULI scanning a QRCode to retrieve the same location, which grants this application some flexibility.

There is only one attribute present in this class.

LocationIdentifier

A string representing an identifier for the related Location class. These may be GPS coordinates or a ULI.

5.2.2.4 The Content class

The Content class encompasses the set of meta-data about the Contents that are related to a certain Location. This includes a name for the content, a path pointing to the actual content in the file system, some tags to better characterize the content *per se* and its type, in order for the application to know how to handle it.

Name

A name given to the content. It may be something more descriptive than its file name because this will be the text presented when the application lists and describes a Location's Contents.

Path

The path to the actual content in the system's file storage. This is a relative path, and, in the particular case of this mobile application, the Content's file name. This is because the absolute path is calculated by joining the base directory where to store files inside the mobile device and the strict directory hierarchy where contents are stored. All this is explained in detail in the next section.

Tags

This field differs from its analogous present in the Location class in the way that it characterizes the Content and not the Location. It may be used to tell the user what kind of information is related to a certain Location, such as its history, description or perhaps suggestions related to that Location.

Type

A field that contains the mime type of the content, in order for the application to know how to handle the content. One can also use this to check if Contents on a Location are more graphically or textually intensive.

5.3 The OpenLBS Back-office

To better aid the development and publishing of new Content Packages a back-office was implemented. This back-office automates the creation of both the meta-data related to each Package and the Contents package, containing all the files to be cached.

It also serves as a Package repository, automatically publishing any Package generated using it.

5.3.1 Features

The main aim of this prototype is to focus on showcasing Package generation and publishing features. Being so, the OpenLBS back-office serves as both a Package generator and publisher, functioning as a Package repository.

This way, when using the back-office, a user may generate a Package and, after saving any changes, instantly publish it. He may also update an existing Package, altering its Contents or adding some, without affecting the previously published version.

5.3.1.1 Package generation and editing

Package generation is done through a simple web interface. Note that the Human Computer Interaction regarding that same interface is not one of the objectives established for this work, so usability testing is disregarded at this time.

The interface allows for a user to create new Packages, add Locations to those Packages, editing its meta-data such as the Location name, coordinates and tags, and add Contents to the Locations he created.

This back-office also implements a Package versioning strategy. Already published Packages may receive additional Content or Locations, or having them removed without affecting the currently published version of that Package. Changes between versions are also tracked, enabling the user to know when a certain Content was added. Removal tracking is not implemented, do to a design planning decision, though it could be added without any difficulty. The main reason behind this decision is that it escapes the main purpose of this work.

Upon generating a Location, a QRCode is automatically generated encoding a slightly modified version of the schema proposed for the Universal Location Identifier — as explained in section 4.5.1. That same QRCode is presented to the user in the web interface upon creating a Location inside a Package.

The naming schema used when uniquely identifying Locations in this prototype follows some different rules, in order to simplify it according to this prototype's needs. As this prototype supports a single repository at any given time, it becomes unnecessary to mention it in a Location name. Also, a different URL Schema was given to the Location name to ease the development of this prototype, regarding its integration with the Android Operating system.

The rules followed by the prototype's naming schema are then:

- The URI schema is now “lbs://”;
- The repository name is omitted;
- Package and location name follow as specified before.

An example Location is then named as follows, being that same information encoded in the Location's QRCode:

```
lbs://FraunhoferAICOS/Room011
```

5.3.1.2 Package publishing

The OpenLBS back-office also implements RESTful web service objects and methods, functioning as a repository for the Android application.

Though the OpenLBS back-office does not expose a API in order to remotely publish Content, it lists all the Packages in the system in a format that is machine readable, allowing for devices to parse that information without difficulty.

Since all the information is available through a web service provided by the back-office using REST, automating tasks such as fetching the available Package list, or searching for any given Package become trivial, involving only simple GET and POST requests without any attached XML data, in this particular case.

Further details on the interaction between the Android application and the back-office are given further in this chapter, after some concepts about both the back-office and the Android application are introduced.

5.3.1.3 Package search and update

In a similar fashion, performing a remote Package search using the mobile device is also possible using web services provided by the back-office.

The back-office implements full text search on Package names and their description, allowing for broader search results.

One can also search for updates, providing the server with a small list of owned Package names with their respective version. This is the only POST request that has to be made through the mobile device in this prototype, in order to send some information that will be used to show the user the updates available to the Packages he sends to the server.

5.3.2 Application Usage

In order to better illustrate the features described before, some use case scenarios are presented in this section. The most relevant operations regarding this back-office will be described using still images of the operation being performed along with its textual description.

5.3.2.1 Creating a Package with Locations and Contents

Creating a new Package and adding subsequent Locations and Contents follows a simple straight path. The aim of this interaction is to be simple and to prove that conceiving a localized Package is at the reach of a regular user.

As we may see in Figure 5.2, creating a new Package is somewhat a three layer model. One needs first to create the Package object *per se*, then proceed to create a Location, and then add Contents to that Location. After all is done, the user finishes by saving the Package, creating its first version and publishing it in the repository.

In order to give a better understanding of the process, and showcase the back-office, the series of steps needed will now be explained along with a visual depiction of each one.

The prototype's entry point is a list of all Packages available, as seen in Figure 5.3. From there, the user may choose any Package if there's a need to edit it, remove it or add a new Package. Details about editing will be explained in the next section.

Choosing the "New package" hyper link brings up the page seen in Figure 5.4. Filling the new Package's name and validating the choice pressing the "Create" button creates a new Package bearing that same name, in this case "Paris".

The user is redirected to the screen seen in Figure 5.5. A new Package is now created, but not yet published. Notice the new Package's version is still "0". Only Packages with a version greater than "0" are considered published.

Notice the Package size is also still "0". That's because no contents have been added yet. This counter only considers Contents when calculating the Package size, meta-information that is passed to the mobile device is not considered.

From that point, the user may want to create new Locations for that Package. By choosing the "New location" hyper link, he is taken to the screen depicted in Figure 5.6.

The user may choose to fill the Location's coordinates, as they are not mandatory. Notice the Location's name has no spaces. This is a limitation in this prototype's implementation, not on its design. Changing that behavior is entirely possible with the current architecture.

After the new Location is created, the user is taken to the screen shown in Figure 5.7. Notice how a QRCode has already been created to be deployed in the vicinity of the newly created Location. The OpenLBS identifier of that Location is also presented.

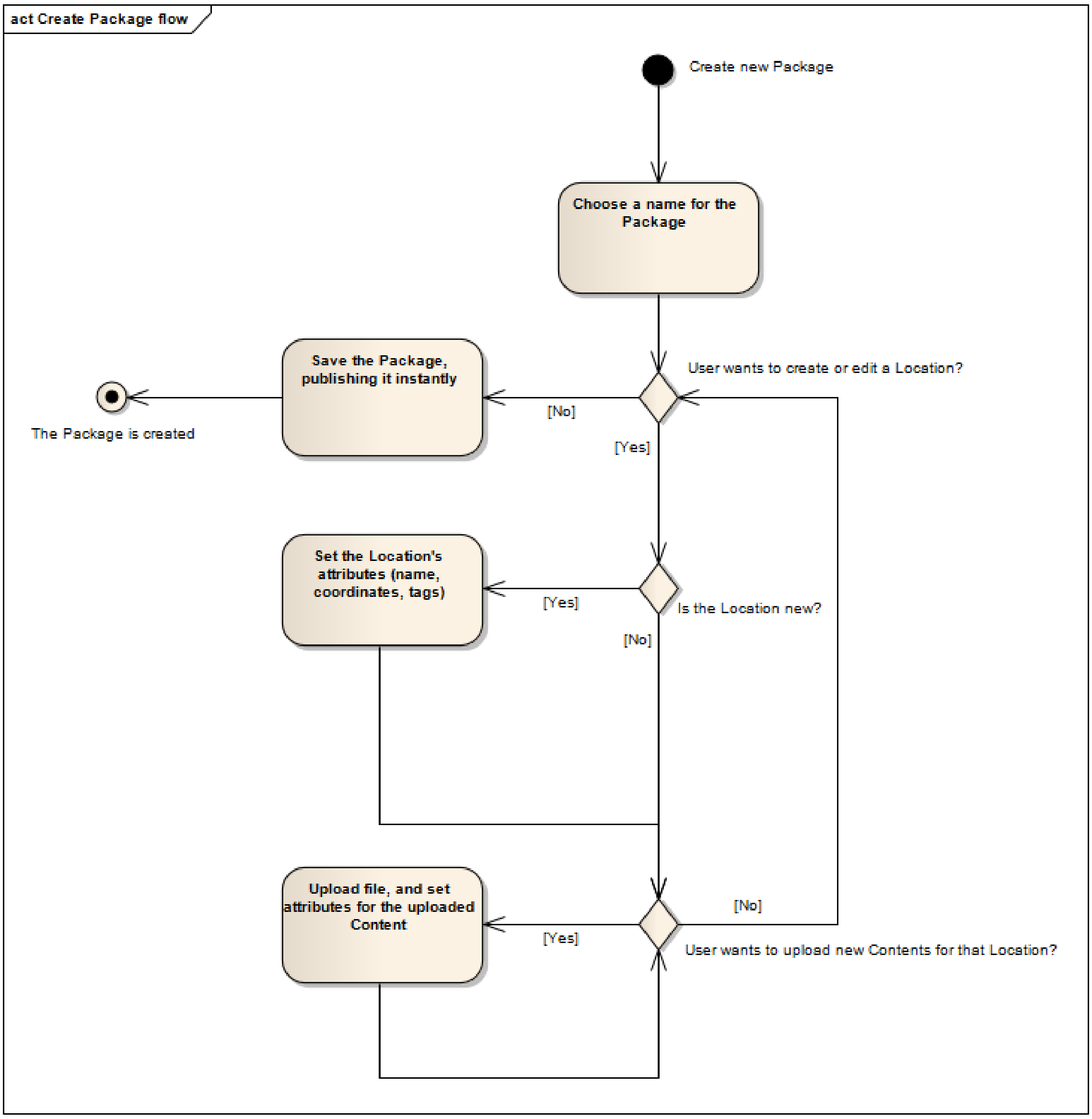

Figure 5.2: Creation of a Package

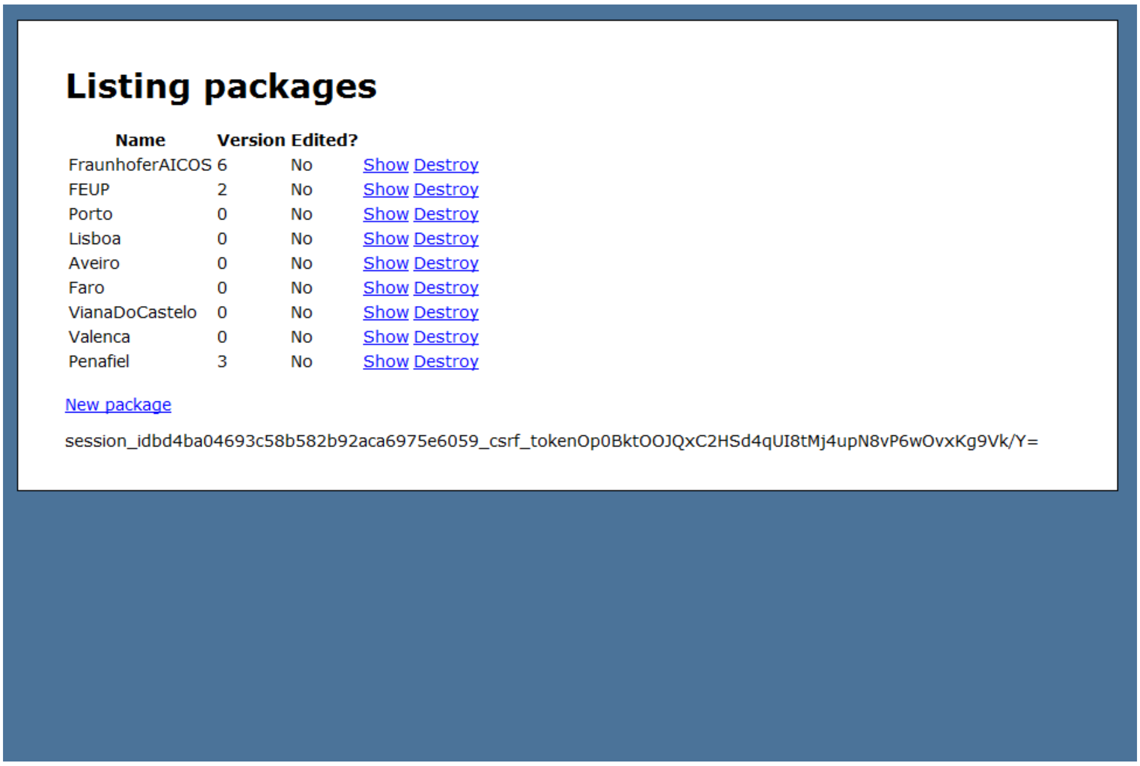

Figure 5.3: OpenLBS back-office listing all available Packages

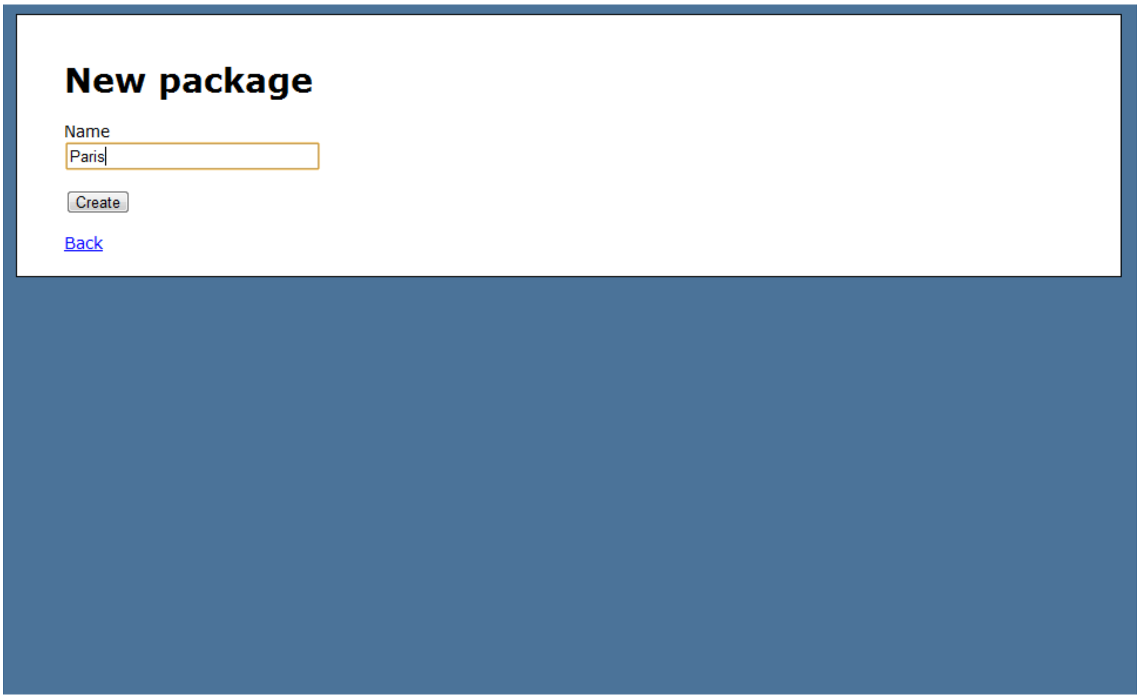

Figure 5.4: New package screen, filled with dummy data

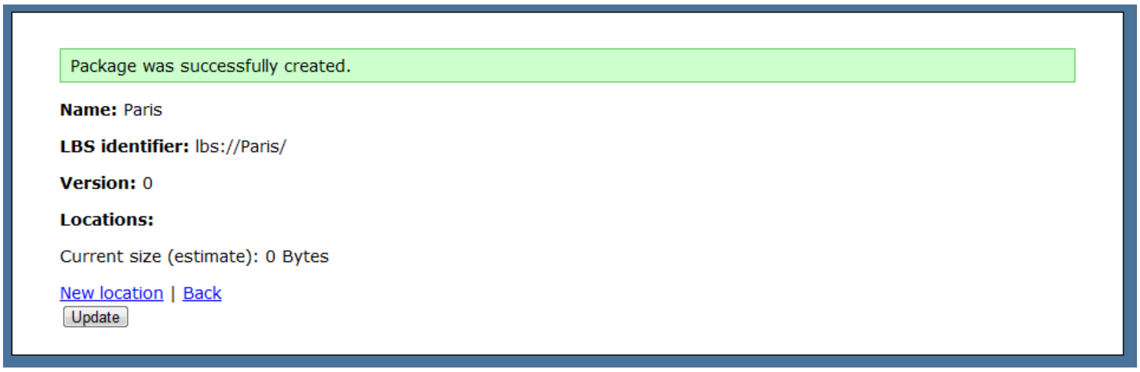

Figure 5.5: Sample Packages as it is created

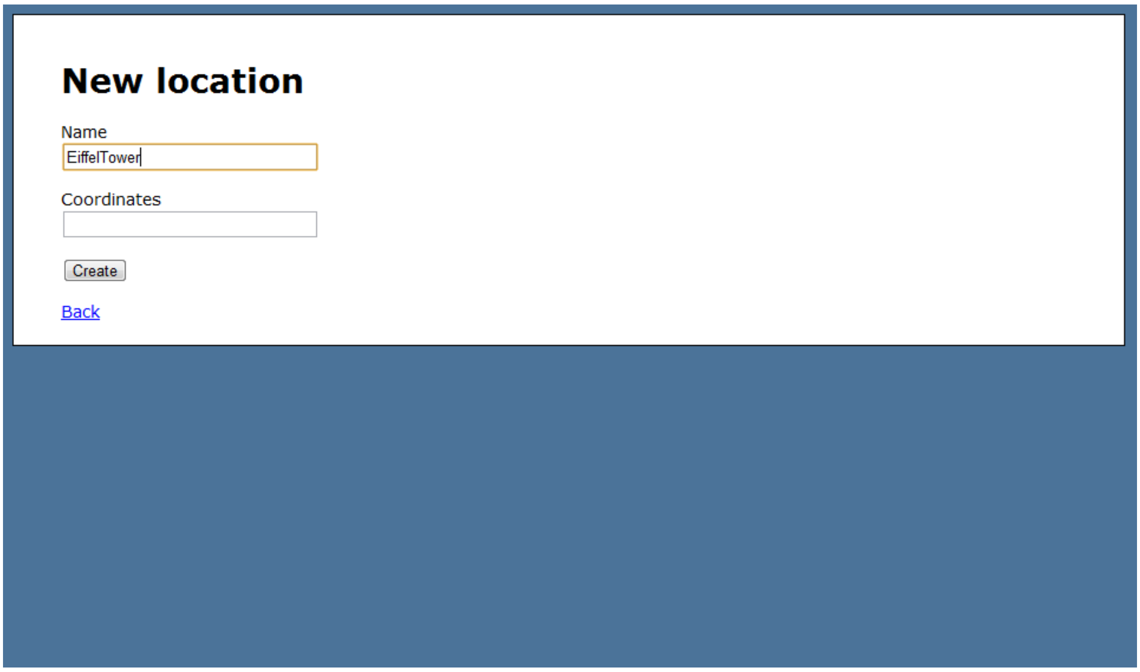

Figure 5.6: New Location dialog, already filled with some data

Figure 5.7: Location right after being created, without any Contents

Figure 5.8: New Content dialog, already filled with some data

As there are no Contents for the Location yet, the process of creating one is described now. By pressing the hyper link “New content”, the user is redirected to the screen depicted in Figure 5.8.

After the meta data fields are filled, the Content file is chosen. None of the meta data fields are mandatory. The name is automatically filled with the uploaded file’s name if empty and the tags are entirely optional.

Now that the Location has some Contents, as seen in Figure 5.9, the user can head back to the Package screen in order to save and publish it.

The Package screen as it shows at this point, depicted in Figure 5.10, shows some modifications regarding its previous state: it now estimates the package size in its next version, identifies newly added Locations and displays a “Update needed!” message.

Once the user presses the “Update” button at the bottom of the screen, the Package is updated to its first version and published in the repository. All the Contents are compressed in a single ZIP file ready for download. This can be seen in Figure 5.11.

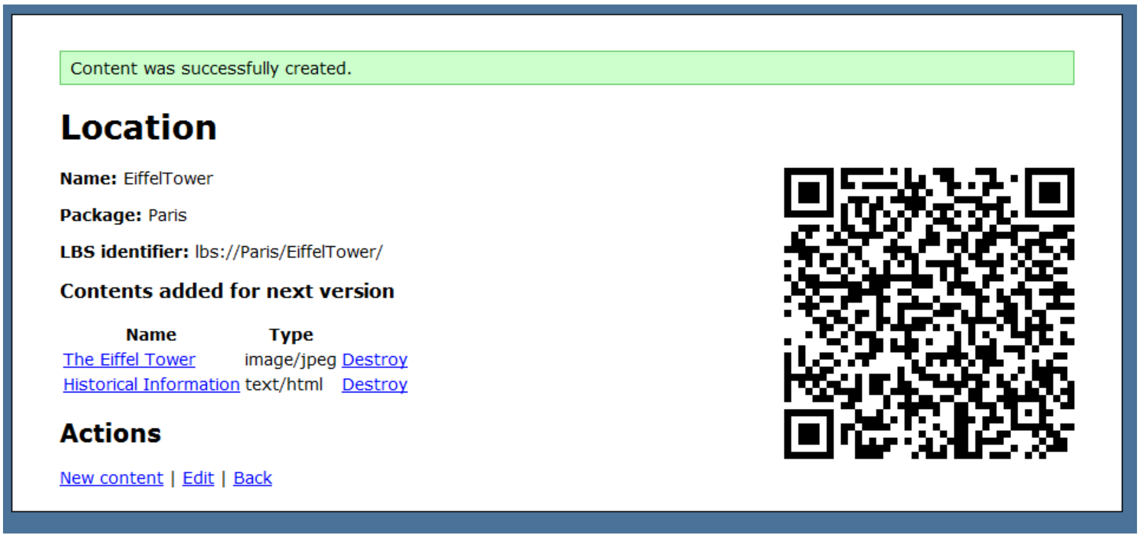

Figure 5.9: A Location with some Contents added

Figure 5.10: The sample Package, ready to be updated

Figure 5.11: The sample Package, updated to its first version

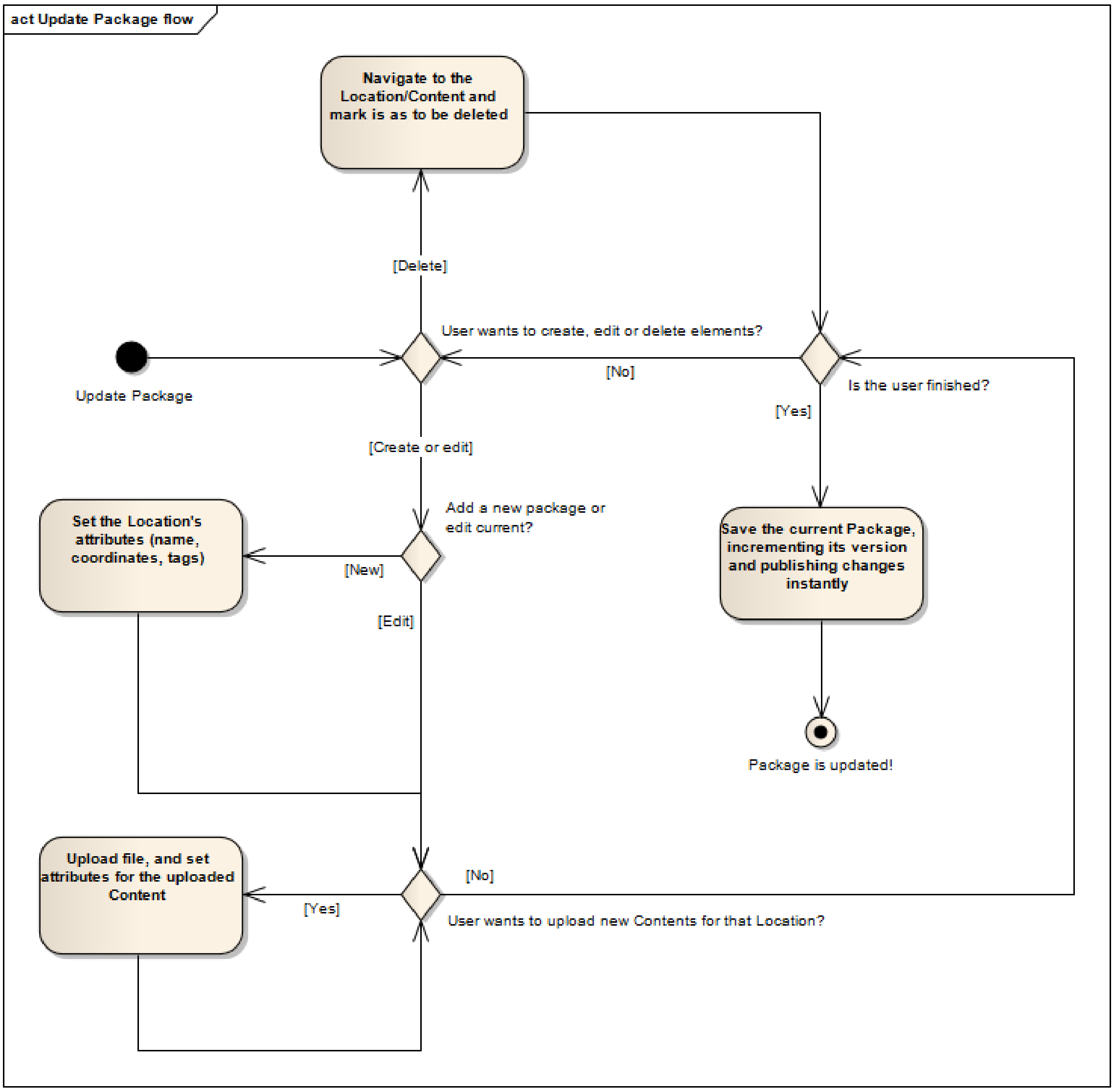

Figure 5.12: Flow diagram describing the update of a Package

5.3.2.2 Updating a Package

The process of updating a Package is quite similar to the act of creating. The user can similarly add, delete and edit Contents, all without altering the currently published Package.

Any element that is added, edited or deleted is marked as such with its alterations sheduled for the next time the user saves the Package. At that time all the changes are commited, the new set of Contents are compressed into a new ZIP file, deleting the previous' version, and the Package is published, with its version counter incremented.

A flow diagram depicting the process of updating a package using the OpenLBS back-office can be seen in Figure 5.12.

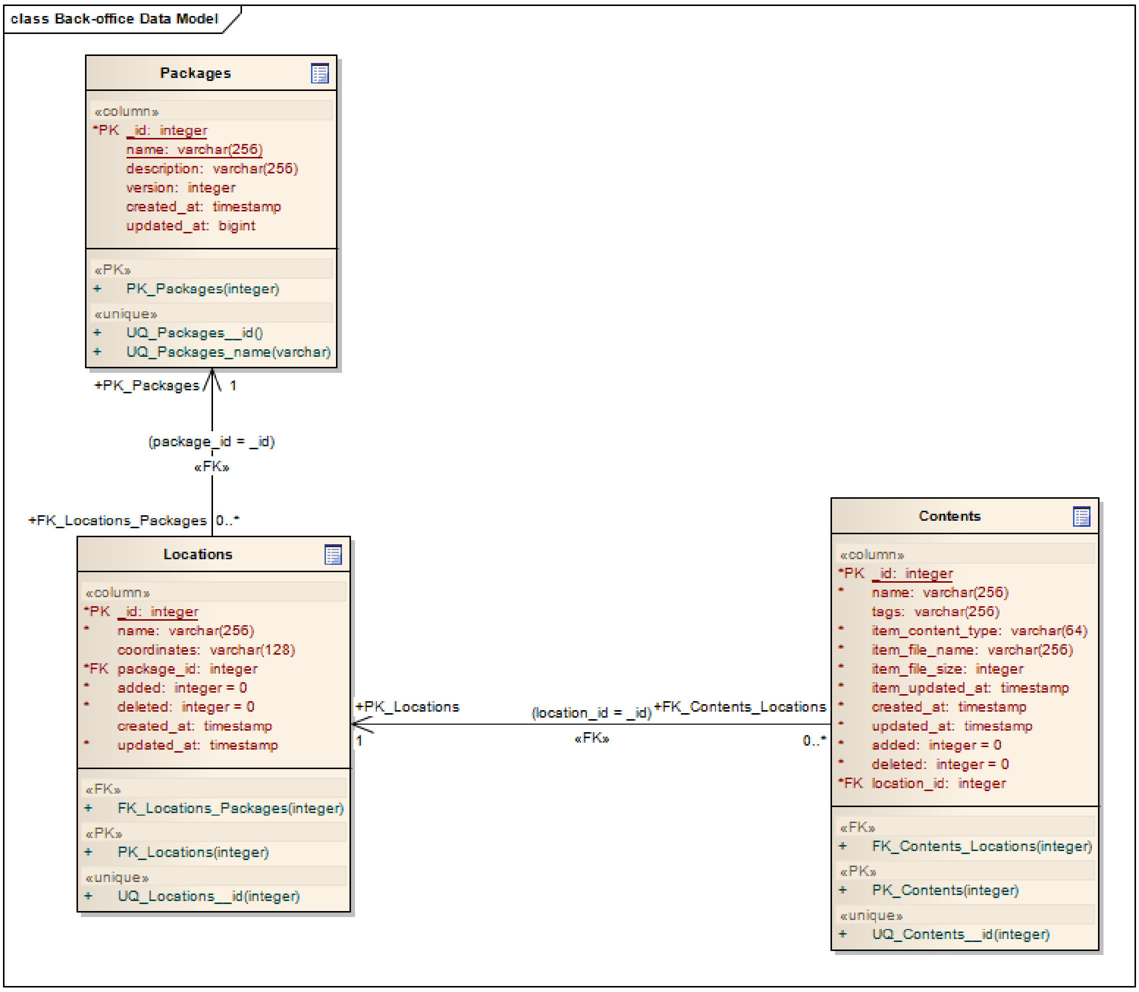

Figure 5.13: OpenLBS back-office database schema diagram

5.3.3 Database Specification

The database schema for the back-office application is based on the class diagram for the information architecture specified in Section 5.2.2, and implements a few changes to allow for some of the back-office's functionality.

The database used in this back-office application is a version of PostgreSQL, capable of performing full-text searches.

5.3.3.1 Database schema remarks

This implementations main differences concerning the initial class diagram are mostly located at the attribute level at each class, and focus on providing additional functionality retaining the meaning of each entity.

In the Package entity, the main differences are:

Added description field

This field will allow the user to provide a better definition of each Package, what will be used when searching for a Package granting better search results.

Added *created_at* and *updated_at* fields

These two fields serve only the purpose of registering when the object was first created and last updated, respectively. They exist mostly for debug purposes, but can be also used to provide additional information to the user, like when a certain Package was last updated.

In the Location entity, these are the differences worth mentioning:

The *added* and *deleted* fields

These two fields aid the Package edit operation, identifying added and deleted Locations by storing in them the Package version they are added or deleted. For example, if a certain Package is at its 17th version, and a Location is added into that Package, then that Location will have its *added* field set into 18.

The Content entity suffers the most changes as the back-office application needs certain attributes in order to manage files uploaded:

The *item_ fields**

This group of fields (*item_file_name*, *item_content_type*, *item_file_size*, *item_updated_at*) serve as an aid for the application to track files uploaded by the user.

5.4 The Android OpenLBS Platform

The Android OpenLBS Platform is the mobile application that was implemented for this prototype. It is able to use a local database to cache contents, map them to a Location and display them as appropriate. It is also able to synchronize with a remote repository and fetch new Packages, as well as update them.

In order for it to behave as such, a local database had to be implemented in order to store information about the Packages, Locations, Contents, and how they are related to each other.

Lastly, that local store will be explored and detailed as necessary. Afterwards, some actions the platform is able to perform are detailed through their flow and finally some key features about this mobile application will be explained in detail.

5.4.1 Features

This prototype mobile application focuses on showcasing a simple interaction between a user's geographic location and Content accessing.

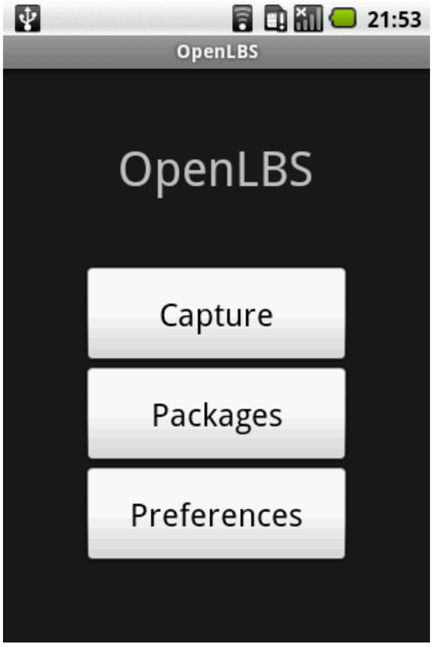

Figure 5.14: Main screen of the prototype Android application

5.4.1.1 Location Visualization

This prototype allows a user to access a Location and retrieve its Contents.

Access is done in a simplistic and meaningful way: by scanning the QRCode that symbolizes that Location. Using that process, it becomes guaranteed that the user is standing at the correct geographic location and doesn't add any complexity to the system.

5.4.1.2 Package installation and update

A user may also need to install additional Content Packages to its mobile system's application.

This prototype allows the install and updating of Packages in a straightforward manner. All the user must do is navigate through the available packages in the repository and choose the one he wishes to install.

The system will then process to unpacking the Package, installing it into the system and checking for future updates from there on.

5.4.2 Application Usage

This section describes the application usage regarding the mobile application of this prototype.

5.4.2.1 Checking available Packages

In order to check available Packages to install or update, the user must press the Packages button in the main screen of the prototype's mobile application, depicted in Figure 5.14.

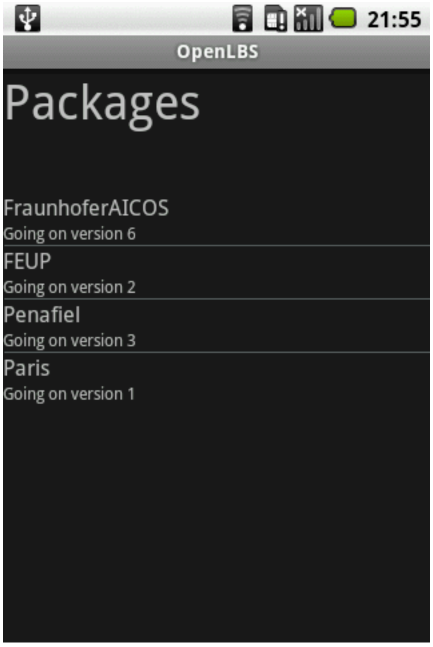

Figure 5.15: Android displaying the list of available Packages

There, he is presented with a choice of all available published Packages in the repository. The list of currently available Packages is shown in Figure 5.15.

5.4.2.2 Installing a Package

In order to illustrate the act of installing a Package, the installation of the "Paris" Package created before in the course of Section 5.3.2.1 will be performed.

Pressing the Package name takes the user to a screen that further describes it, pictured in Figure 5.16. Finally, pressing the "Install" button commands the application to install the Package, and enabling its Contents.

5.4.2.3 Accessing a Location

Package visualization should be done in a simple fashion. That way, in order to access a Location, all one needs to to is to scan the QRCode that symbolizes that same Location, as depicted on Figure 5.18.

After that step is taken, the user is presented with a list of available Contents for that Location, and proceeds to select what he wants to see. Both the contents for the Location shown in Figure 5.19 were added before in the course of section 5.3.2.1, and can be seen in Figures 5.20 and 5.21.

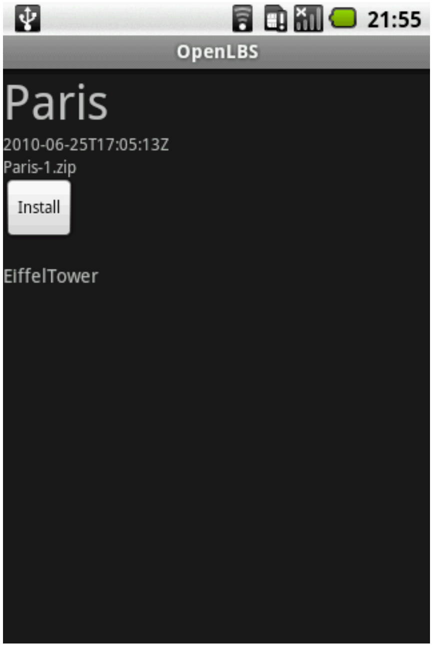

Figure 5.16: Android's informative screen describing a Package

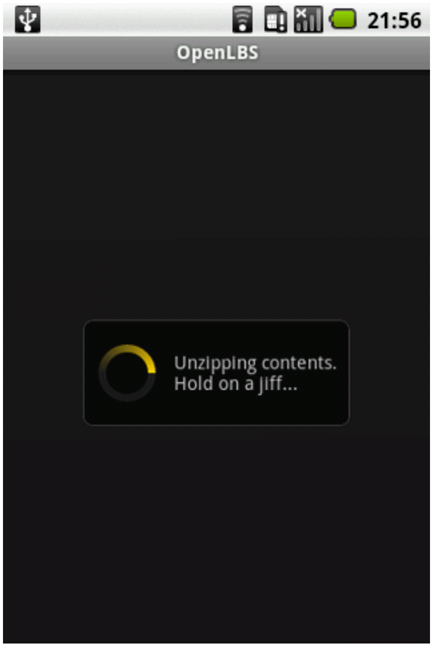

Figure 5.17: A screen of Android's application performing the install operation

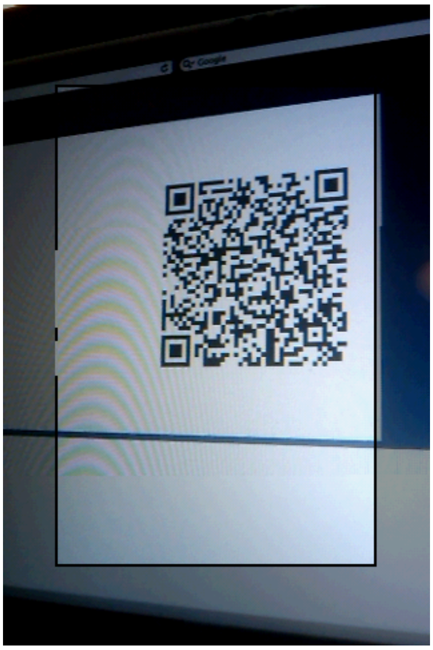

Figure 5.18: The Android application scanning a QRCode

Figure 5.19: Location screen on the Android application

Figure 5.20: Sample Content consisting of an HTML document

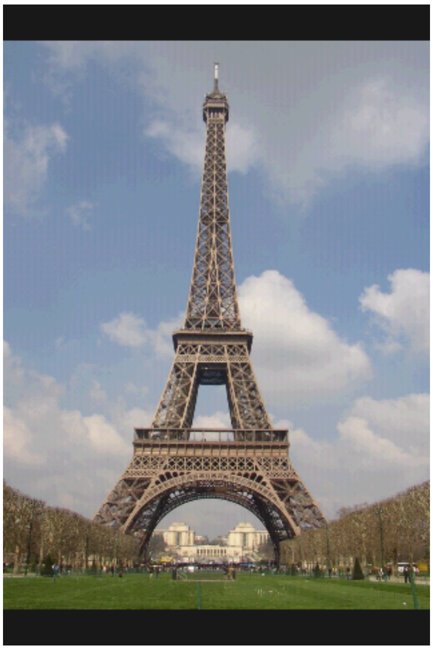

Figure 5.21: Sample Content consisting of a JPEG image

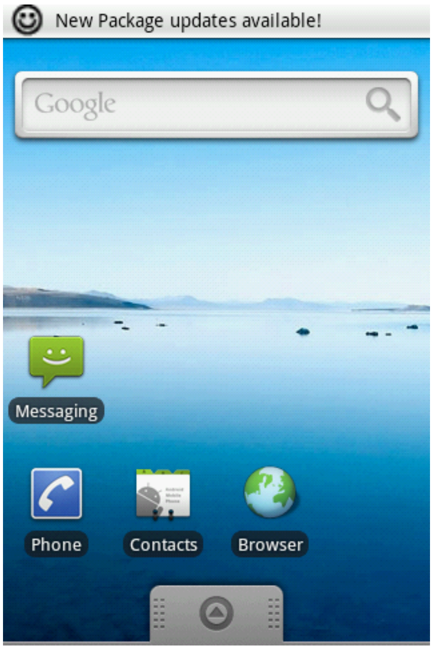

Figure 5.22: New update notification in Android's status bar

5.4.2.4 Updating installed Packages

The update process is straightforward: a user simply has to navigate to the Package list shown in Figure 5.15 and choose the Package he'd like to update or install. Packages that are already installed on the system are marked as such, and Packages that need to be updated also bear a distinctive mark.

The system also has a process that occasionally, depending on the user's preferences and whenever there is an Internet connection checks for new Package updates and notifies the user, as depicted in Figures 5.22 and 5.23.

5.4.2.5 Setting some application parameters

The user may choose to set some application parameters regarding where the prototype shall save the Content files, when should the prototype check for new Package updates, and on what type of connection should it function when checking (WIFI only, or WIFI or 3G).

If he wishes to do so, he may navigate into the Settings screen and define the settings he pleases.

5.4.3 Database

Content caching is done through two different types of local storage: a small relational database that stores meta-information about Packages, Locations and Contents; and the mobile phone's file system that stores the Content files.

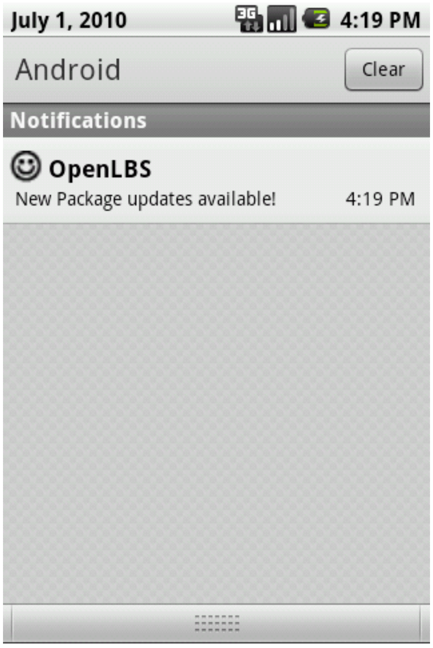

Figure 5.23: New update notification in Android's notification tab

The relational database is implemented through Android's "sqlite3" libraries present in the original SDK of Android's platform. Though simple, it allows for simple relations between entities (tables) which is well enough for this prototype's functionality.

5.4.3.1 Database Specification

The local "sqlite3" relational database will store meta-information regarding Packages, Locations and Contents. That meta-information encompasses a set of attributes such as tags for a Location or Content, names for all of the entities and, most importantly, the relations between them, such as what Contents belong to a Location or what Locations are included in a Package.

The schema used also aims to implement the class diagram mentioned in Section 5.2.2, its similar to the schema used in the back-office application. It doesn't add any entity attributes, however, due to their lack of use in the Android application.

The schema diagram is depicted in Figure 5.24.

5.4.4 Local Content Storage

In order to store content files, the Android's file system is used. This is because, generally, the file system is better optimized to deal with files than the "sqlite3" implementation.

The application follows a strict directory hierarchy in order to store the content files. It is similar to the ULI schema used in this prototype, as explained in Section 5.3.1.1.

Local storage directory hierarchy rules follow the same principle. The base directory for this structure is a predefined one that may be configured in the OpenLBS application preferences. From

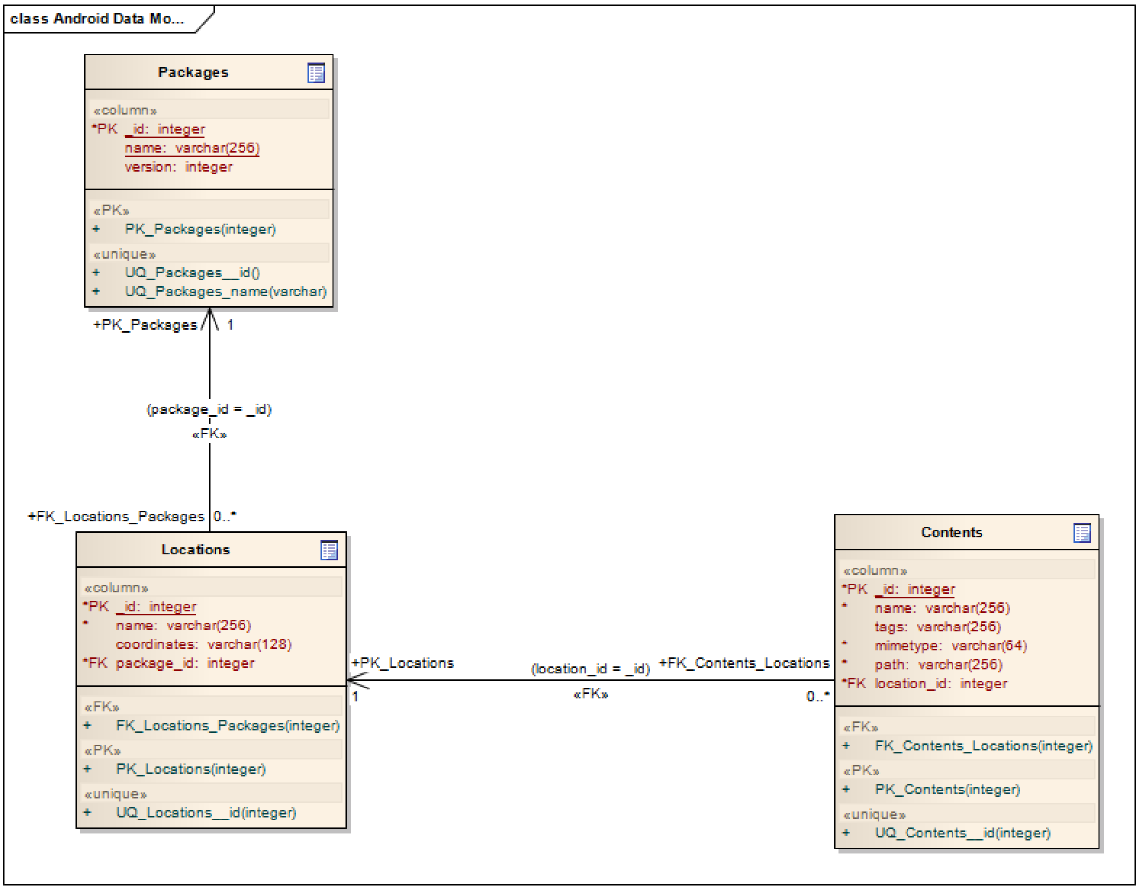

Figure 5.24: The schema diagram for the Android application

there on, the directory structure follows the same hierarchy as the Location naming: the first level of directories are named after the Location's Package and the second level after the Location's name.

The directory structure for the example Location mentioned above would then be, assuming the base predefined directory is "/sdcard/pt.fraunhofer.openlbs/content":

```
/sdcard/pt.fraunhofer.openlbs/content/FraunhoferAICOS/Room011
```

In this prototype, content files are stored as they are, which means every file contained in a Location will be stored in its original format inside its respective directory.

5.5 Component integration

This section describes in greater detail the interaction between the two main components of this prototype. It focuses on the data interchange operations and the exploits developed in order to make the mobile part of the prototype an application platform instead of just an application.

5.5.1 Information Exchange

Information exchange between both components of this prototype is done via the HTTP protocol, as both the entry points for user interaction and the exposed API of the back-office counterpart are web components.

When the Android application requests information about the Packages available on the repository, updates or even when installing a Package, all the process is done through HTTP requests.

This decision happens because there is was no need to implement complicated communication techniques at this point to exchange the kind of information that is needed. The HTTP protocol doesn't convey that much of overhead for it to be considered, and the stability and robustness of its network stack are well proven by the stress testing it has been subject to for the past twenty years.

In order to convey information to the mobile device, the OpenLBS back-office or repository uses JSON to encode it in the HTTP responses to the requests performed by the mobile client.

JSON was the format chosen between a set of formats that included XML, plain SQL and a customized format. This is due to JSON being light in size and complexity, without sacrificing portability. Though XML may be more descriptive, it is also more verbose and more complex. While exporting the back-office database and sending its pure SQL would work, the size gained from performing this operation would not be that greater compared to JSON, sacrificing portability and adding complexity to the system. The same principle applies to using a custom format.

There are two interactions using JSON in this prototype: when a Package listing is requested, either for installation or update purposes, and when a specific Package information is requested.

The last contains all meta data of a specific Package, needed for instalation. It describes all the Locations and Contents on the specific Package, and it is what the mobile application uses to populate its database.

Examples of JSON responses apended to each HTTP response both depicting the list of available Packages on the repository and the full meta data on a specific Package can be found on appendixes A and B, respectively.

As for the Content files, they are compressed in the ZIP format when the user saves a Package in the back-office, becoming ready for immediate download. They are fetched as a Package is installed in the mobile device.

5.5.2 Interaction as a content platform

This prototype, as is, can be easily integrated with other applications that may try to use offline localized content, providing they implement the necessary mechanisms to access it.

Using Android's Intent system, third party mobile applications for this mobile operating system may request a OpenLBS bar-code scan thus allowing the third party application to know the user's current localization context. OpenLBS also returns a list of all the contents in that specific location when that operation is performed.

That list of contents can also be used by the third party application, as this prototype implements an operating system level handler for the schema used when identifying OpenLBS Locations — the schema "lbs://". That way, when a third party application receives the list of contents, it may choose to access them by simply calling the predefined handler with the Content identifier.

Contents are uniquely identified by appending the Content file's name to the prototype's Location identifier, explained in section 5.3.1.1. An example Content identifier using the Eiffel Tower figure would be:

```
lbs://Paris/EiffelTower/8a06ad3781fef2d0.jpg
```

Though the identifier loses some of its human readability due to appending a randomly generated file name to the Location identifier, there shouldn't be any situation where users manipulate Content identifiers directly. Everything regarding this identifier is done in the background of this two-way application interaction and does not require human intervention.

Since a handler for that schema is defined in Android by OpenLBS, the operating system shall take whatever actions necessary to pass the request to the OpenLBS platform in order for it to return the Content *per se*.

This way, third party applications may use all functionalities this platform provides.

Chapter 6

Evaluation

As with every project, an evaluation is conducted to assess both the feasibility and purpose of the idea proposed. Some tests have been performed, covering both the size of an actual Package, the bandwidth it will need to be installed and the strain on the CPU and battery of the mobile device.

6.1 Overall evaluation

In order to test the viability of this project, a Package was generated for the premises of the Fraunhofer Portugal Association, at Campo Alegre, Porto.

The Package includes a set of Locations that describe a room or a desk. Each Location that is attached to a room has various HTML files that both describe the room itself and the people who work there. Locations attached to a desk describe textually via another HTML file the person that works in there, attaching a picture of that person. These Locations also describe the projects that the person positioned there works on, through text, pictures and sometimes video or sound. Along there are some miscellaneous Locations that represent points of interest, such as the water cooler, with instructions on how to use them and/or what they mean.

The application was successfully deployed in the scenario depicted above. It performs as it should, displaying off-line content as a Location is triggered via a QRCode scan positioned at the physical location.

The Android application was set up in an HTC Magic smart phone, capable of running the Android Operating system, version 1.5. The specifications of the device can be found in Appendix D. Those should be checked in order to verify its storage and processing capabilities.

The Package was built entirely using the back-office, and installed or updated using the interaction between both the back-office and the mobile application. All that interaction works as planned, thus making the prototype for this project fully functional.

The prototype works as specified in the workflows explained before in Chapter 5.

6.2 Package contents size

Upon full completion, the final Package Content file size is 6,403,947 bytes. A full listing of the files in the Package, including its types and sizes is presented in Appendix C.

Besides being a small value, it is quite well-suited for today's standards of mobile equipments. As of the time of this writing, mobile devices have default storage capacities that go well beyond 1 gigabyte, and most of them can may have that capacity extended using expansion cards, such as SD or MemoryStick cards.

It is also good to take into consideration that no Content adaptation techniques are being used in this prototype, so all files are downloaded unmodified into the mobile device. Should it be implemented, there would be a notorious gain in size when the majority of Contents are image or video based.

6.3 Storage space used

In order to calculate the full storage space used by this test suite one needs to add the size occupied by the application code in the mobile device to the size occupied by the "sqlite3" database and the Content files.

The application currently occupies 512 kilobytes in the phones internal memory, while the relational database fits in an extra 8 kilobytes, providing it stores only the meta-information of the test Package, that is, the information regarding the several Locations and Contents that Package has without the Content files per se.

The total sum is 520 kilobytes occupied in the phone's internal storage for the application and the Package meta-data and roughly 6.5 megabytes in the external storage, in this case an SD card, for the test Package's Content files.

6.4 Bandwidth usage

In order to download and install or update a Package, several operations regarding data exchange are to be performed. Providing the difference in bandwidth consumption between updating and installing packages is negligible, this section will analyse the install Package operation regarding its bandwidth usage.

Before any Package installation is performed, the mobile application needs to check what available Packages are there to download. In our test repository, there were 4 published packages. The downloaded response that contains that full list of Packages weighted 353 bytes, including HTTP headers for the response. In the response, the data describing each Package had an average size of 90 bytes. The response containing the listing of all Packages available in the repository can be consulted in Appendix A.

When installing a Package, its definition must be downloaded first: the Package's meta-data that is to be inserted into the mobile device's database that the server responds using JSON. The size of the response is 6,593 bytes, including HTTP headers and the full JSON response, which may be consulted in Appendix B.

Next, the Package Content files are to be downloaded. This is the ZIP file containing all the Content files, so its size is already known: 6,403,947 bytes, as stated above.

6.5 Battery and CPU usage

In order to measure battery and CPU usage, an external Android application, called PowerTutor[1] was used. PowerTutor is a light profiling application developed by University of Michigan Ph.D. students Birjodh Tiwana and Lide Zhang under the direction of Robert Dick and Zhouqing Morley Mao at the University of Michigan and Lei Yang at Google.

The two most significant operations will be analysed: checking a Location and installing a Package. The figures to be presented are actual screenshots of the PowerTutor Android application. Each presents four graphics depicting the energy use of four mobile device key parts. From top down: the CPU, the Wifi interface, the LCD and the GPS receiver. Higher values mean more electrical power is being drained by that specific part.

The right side of the figures presents a summary containing the average power consumption during the time PowerTutor ran on the application.

6.5.1 Base benchmark results

To evaluate CPU and Battery usage, one needs to have a set of known values in order to perform a comparison.

This section exposes two applications that will serve as comparison points. For the CPU intensive application a popular game is used, Light Racer[2]. As it is a real time game, it is bound to draw a great deal of CPU power.

As we may see in Figure 6.1, Light Racer tops CPU usage throughout its course. The average amount of energy drawn by the CPU while the application is running is 367.40mW.

The other application is Google Talk, Google's instant messaging service. As this is an application that is expected to be always running while the phone is turned on, its energy consumption must not be very high. We may see exactly that in our results when testing that particular application, seen in Figure 6.2. The mean CPU energy usage is about 189.69mW throughout the application's use.

[1] See: http://powertutor.org/
[2] See: http://www.bestandroidapps.net/2010/03/light-racer-3d-app-review/

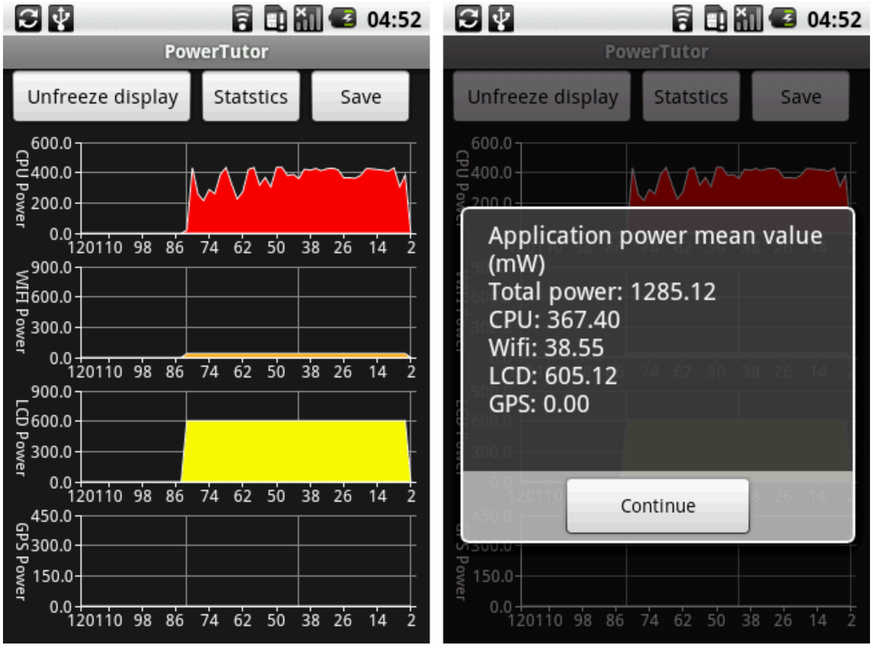

Figure 6.1: Light Racer's energy profile

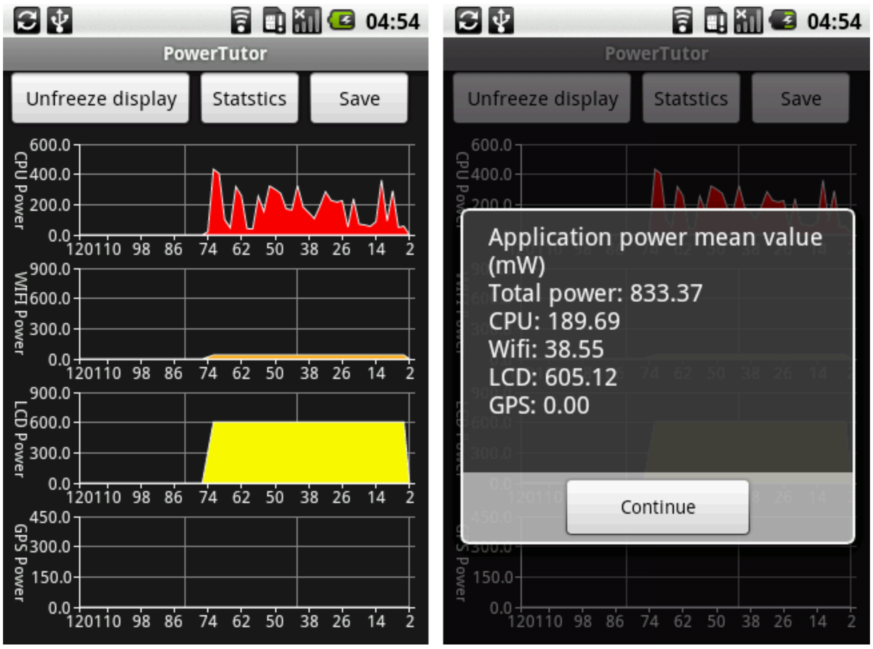

Figure 6.2: Google Talk's energy profile

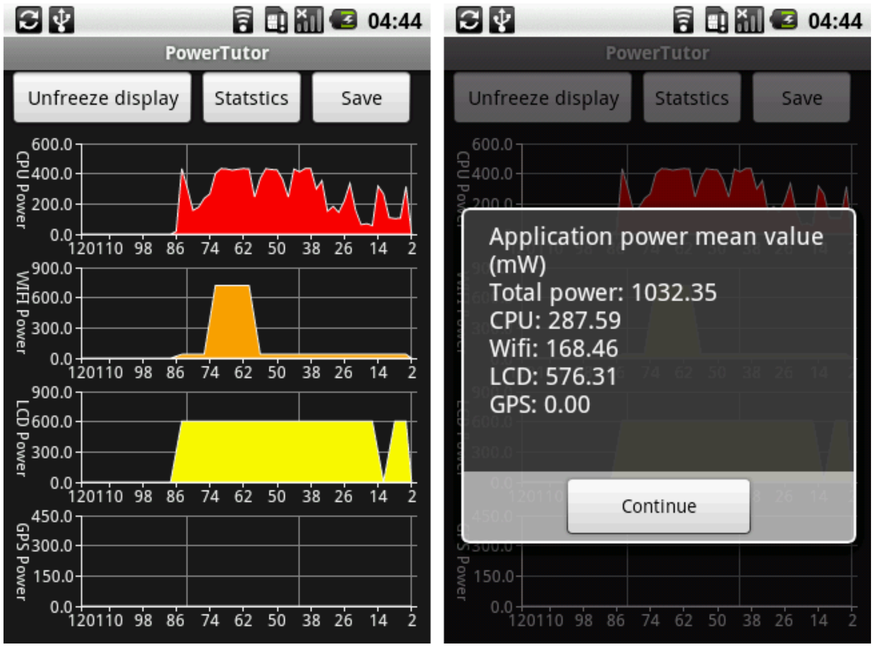

Figure 6.3: Package installation energy profile

6.5.2 Installing a Package

The operation of a Package install encompasses all the process from fetching the Package list from the repository to adding information to the mobile device's internal database and decompressing Content files into their respective directories. The results of such process are depicted in Figure 6.3.

As it may be seen from the results, WIFI power is used during the first step of the operation, during the phase of fetching the list of available Packages, the Package meta-data and the Contents compressed file. The rest of the time is devoted to decompress the Contents and insert the meta-information into the database.

At a first glance, these results seem coherent with the nature of the operations performed. However, one may notice the CPU is being intensively used when fetching information from the Internet. That may be subject to study later on to determine if it is an operating system issue or badly performing code in that section.

Overall power consumption seems appropriate, though. Fetching a Package costed an average of 287.59mW of CPU power, plus 168.46mW used by the WIFI interface and some stunning 576.31mW used by the LCD, resulting in a total of 1032.35mW of average power consumption. Compared with the results of the two applications selected as base cases, in Section 6.5.1, one may observe that this process is neither too much or too little CPU bound. As this is an operation that is deemed not frequent, and the battery usage is not exhaustive, one may conclude these are satisfactory results.

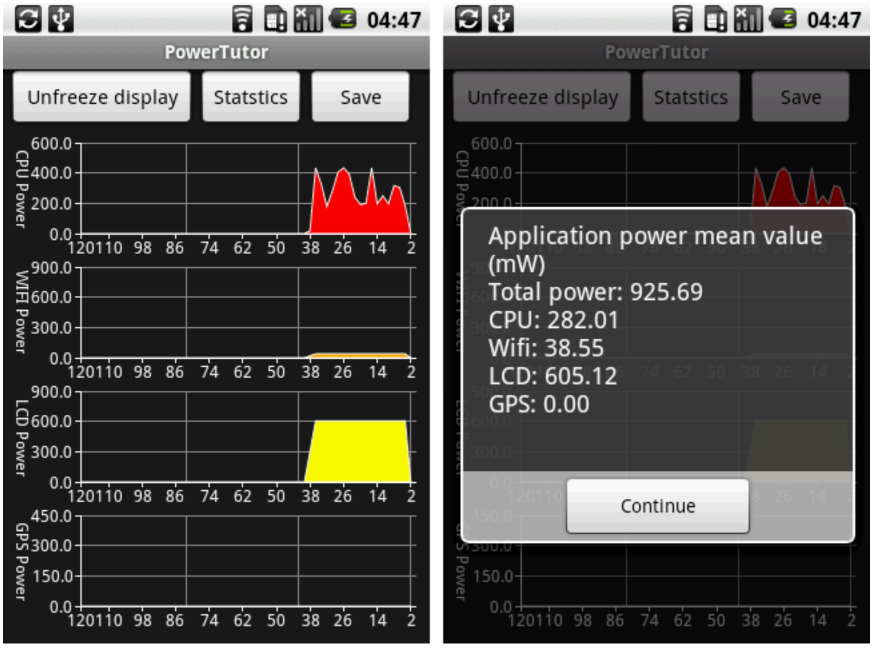

Figure 6.4: Energy profile when consulting a Location's contents

6.5.3 Accessing a Location

The process of accessing a Location for its Contents starts by scanning a bar-code using the mobile device's camera. Then it searches the local database for that Package and then displays the Content listing.

For this test a bar-code was scanned, resulting in a successful Location fetch. From there three Contents were observed without a great deal of delay — practically only the time to close the visualization of a Content to hop on to the next: two HTML documents and a JPEG image. The power profile of that operation may be seen in Figure 6.4.

As it may be observed, there are several spikes that match some stages of the operation performed. The first spike indicated the start of the application, the second long spike the process of acquiring the bar-code through the mobile device's camera and fetching the Location information from the database. Finally, the last three spikes result from checking two HTML documents first, and then a JPEG image.

While one would expect for this process to be not that intensive on the CPU and the battery, that doesn't quite happen. That is also because all this process was performed without any breaks when opening Content files one after another. As this is not the normal case scenario — a normal user would stop to check the Contents instead of immediately closing them — this test's purpose is only to illustrate the total battery drain this process would provoke. As it was done quite quickly, that influenced the average calculation.

As any keen observer would have noticed, the Wifi interface is drawing battery power on these latest tests. That is not because it is being used to fetch Contents, as it is observable that there is a big difference of energy use when the Wifi is actually pulling information, as seen in Figure 6.3. This is only a residual power consumption due to the fact that the Wifi interface was on during these tests, albeit not actually connected to anything.

Nevertheless, results turned out to be satisfactory. With an average of 925.69mW drained this process is, resembling the Package installation process, neither too much or too little CPU bound.

Chapter 7

Conclusions

This final chapter closes this document by concluding the thoughts about this project. Final considerations are weaved into the everlasting loom of time, and committed arguments proving both the feasibility and purpose of this project are sewn as patchwork into the to be omniscient mantle of science.

The dilemma that this work addresses doesn't fall within the purviews of the conundrums of philosophy, as this is a typical engineering problem. This document proposes to solve a practical problem, with a simple, efficient and proven practical solution. As James Michener once said: "Scientists dream about doing great things. Engineers do them".

Afterwards, some considerations about possible future work are described.

7.1 Final thoughts

As this project reaches its end, one may draw several conclusions from its completion.

Firstly, having identified a clear fault regarding connectivity and location based services, a conceptual solution was proposed that solves that issue at a low cost for the end users, both at the monetary and the convenience level.

Researching about this subject revealed that this kind of work is yet an untouched field. All the work around Context-awareness seem to revolve around gathering and interpretation of contextual information disregarding possible limitations on the environment, such as the ones presented as the initial problem of this project.

After a conceptual approach, a prototype was developed in order to test the feasibility of the solution provided. Although not all its conceptual elements were implemented, the ones that were became well enough to prove that this concept perfectly works as intended, as the evaluation performed on the prototype shows.

The platform specified as it is may be used by another applications or even another developers to build similar systems that will allow end users to benefit from the solution presented. Due to the simple design conceptualized, an application using these ideas, or even a fully fledged OpenLBS implementation may be constructed quickly.

Results gathered when testing this prototype show very satisfactory values regarding bandwidth, CPU and battery usage. Also, the system behaved quite well during that same testing, displaying both stableness and functionality. Along with some refinements, particularly in the area of human computer interaction, a version ready for the end user could be achieved in little time.

This project was an overall success. For a problem that was presented there was a solution and a proof of concept that proved the feasibility of that same solution. Better yet, this approach to this problem may indeed aid people in overcoming current Internet connection limitations in the short to long term.

7.2 Future work

As with many projects, there is an array of tasks and ideas worth considering that didn't fit our timespan to implement. What follows is a compilation of those planned for future work on this subject.

Interaction between Locations

The platform could allow interaction between Locations, in the way that users could see what surrounds the Location they are visiting at the moment.

Should a Location be identified by, or contain information about its GPS coordinates, finding nearby Locations is rather easy. The platform could that way suggest nearby points of interest, or allow the user to search for them using the information tags provide to characterize Locations.

Content adaptation

Content adaptation, as explained in Section 4.6, isn't implemented in this project's prototype. It would be rather interesting to assess the impact this measure would have on this project.

Incremental updates

At the time of this writing, updates are done not incrementally. This means that Contents that the client already possesses are downloaded again if the user wishes to update that specific Package.

Implementing this idea would bring some overhead to the communication between the mobile client and the repository. That is due to the extra amount of information the mobile client would have to send the server either specifying the version of the Package he wishes to update or the list of Contents he has already in his possession. However, the gain in downloaded size regarding Contents would most probably justify the extra communication and repository processing time.

Package suggestions

Should a user download a specific Package, other Packages may be suggested that are somewhat related to the later. For example: if a user downloads a Package containing information about the Louvre, the Package Paris may be a relevant suggestion.

One step beyond this would be creating Package dependencies. However, in spite of its advantages regarding convenience to the user at the time of choosing the correct Packages, it would also bring nuisances when updating Packages, unexpected Package additions in order to satisfy dependencies resulting in increased download sizes and such, and problems due to poor Package maintainability. It remains under consideration if this step is actually a good idea.

Repository information caching

Every time a user wishes to check out the available Packages at a given repository, he must connect to the Internet.

While that seems logical in a pragmatic point of view — if a user wishes to download a Package, he should be connected — it becomes a limitation if the system becomes intelligent enough to tell the user he is missing a certain Package when he visits a Location he doesn't have on his mobile device.

For this kind of use, a local cache of the repository's Package listing and their respective Locations, along with some extra information, such as size and tags would be desirable.

Off-line navigation

Adapting this platform to accept a map as a Package-wide Content and fitting it with the according routing algorithms would turn this platform into an open navigational system, which would allow users to create their own maps and upload them in order to guide other users through a conference, or even a small part of a town.

Along with projects such as Open Street Map[1], this could bring a potentially broader interest on this platform by the general public.

Reusing Content files

[1] See: http://www.openstreetmap.org/

If a Content file repeats over several Locations inside a Package, it might be a good idea to reference it over all the Locations it is used on instead of repeating it throughout all Locations, as it is done now.

Encrypting Content files

Should a user when producing a Package would like to enforce the presence of the users that will use that Package in the Locations defined in order to read their Contents — prevent them from accessing the Content files through a file manager for their mobile device — he could encrypt the Content files and append the decryption key into the Universal Location Identifier for a given Location.

The result ULI that would be encoded in a QRCode, for example, could be something like:

```
lbs://FraunhoferAICOS/Room011Desk03?key=HD7fhnS6sdnaS&dhf5bs==
```

This way, a user would have to be in the vicinity of that Location in order to decrypt its Contents.

Adoption of a peer-to-peer architecture

The conceptual solution may also suffer some improvement in some areas, such as abandoning the client-server model for a peer to peer one that would bring benefits when acquiring content from another mobile device at close range via a cost-free wireless technology, such as Bluetooth.

Two way synchronization

Other possible thought would be 2 way synchronization of Contextual data and its sharing between peers, which would allow users to leave notes, as an example.

7.3 Conclusions

Ultimately, this project serves as a great starting point for this kind of off-line caching interaction. Accounting for the end of this stage, we are left with a clear set of concepts, a working prototype and a pack of ideas to apply on it, in order to experiment new thoughts and ideas, which will surely aid in the pursuit of truly ubiquitous information exchange.

References

[ADB+99] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London UK, 1999. Springer-Verlag.

[Amb10] Scott W. Ambler. Mapping objects to relational databases: O/R mapping in detail. http://www.agiledata.org/essays/mappingObjects.html, 2010. Accessed 25th June 2010.

[BC04] Gregory Biegel and Vinny Cahill. A framework for developing mobile, context-aware applications. In *PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, page 361, Washington, DC, USA, 2004. IEEE Computer Society.

[BDR07] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, 2007.

[BHO+09] Marc Benhaim, Cédric Houlette, Lutfi Ilteris Oney, David Buckley, Doreen Dentes, Mark Van Eeghem, Raman Chhima, Silvério Paixão, Michaela Hähn, Wang Yi, Naoko Mori, Jean-Claude Muller, Michel Ottiker, Nora Kaci, Hitesh Brahma, Nevenka Elvin, John Pearce, Frank Sharkey, and Jim Willmott. Introduction to gs1 datamatrix. Technical report, March 2009.

[DAW98] Anind K. Dey, Gregory D. Abowd, and Andrew Wood. Cyberdesk: a framework for providing self-integrating context-aware services. In *IUI '98: Proceedings of the 3rd international conference on Intelligent user interfaces*, pages 47–54, New York, NY, USA, 1998. ACM.

[Dey98] Anind K. Dey. Context-aware computing: The cyberdesk project. In *AAAI 1998 Spring Symposium on Intelligent Environments*, pages 51–54, Palo Alto, 1998. AAAI Press.

[GLA10a] More ways to share your Google Latitude location. http://googleblog.blogspot.com/2009/05/more-ways-to-share-your-google-latitude.html, 2010. Accessed 27th June 2010.

[GLA10b] See where your friends are with Google Latitude. http://googleblog.blogspot.com/2009/02/see-where-your-friends-are-with-google.html, 2010. Accessed 27th June 2010.

[GOO10] Google's Coorporate Information: Philosophy. http://www.google.com/corporate/tenthings.html, 2010. Accessed 27th June 2010.

[HI06] Karen Henricksen and Jadwiga Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2(1):37 – 64, 2006.

[HSP+03] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-awareness on mobile devices — the hydrogen approach. In *HICSS '03: Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03) - Track 9*, page 292.1, Washington, DC, USA, 2003. IEEE Computer Society.

[JFF98] David Franklin Joshua, David Franklin, and Joshua Flachsbart. All gadget and no representation makes jack a dull environment. In *In AAAI Spring Symposium on Intelligent Environments. AAAI TR*, pages 155–160, 1998.

[JSO10a] JSON. http://www.json.org/, 2010. Accessed 27th June 2010.

[JSO10b] JSON: The fat-free alternative to XML. http://www.json.org/xml.html, 2010. Accessed 27th June 2010.

[KMK+03] Panu Korpipaa, Jani Mantyjarvi, Juha Kela, Heikki Keranen, and Esko-Juhani Malm. Managing context information in mobile devices. *IEEE Pervasive Computing*, 2(3):42–51, 2003.

[KT05] H. Kato and K.T. Tan. 2d barcodes for mobile phones. In *Mobile Technology, Applications and Systems, 2005 2nd International Conference on*, pages 8 pp.–8, Nov. 2005.

[KT07] Hiroko Kato and Keng T. Tan. Pervasive 2d barcodes for camera phone applications. *IEEE Pervasive Computing*, 6:76–85, 2007.

[MWBS09] Alessandro Mulloni, Daniel Wagner, Istvan Barakonyi, and Dieter Schmalstieg. Indoor positioning and navigation with camera phones. *IEEE Pervasive Computing*, 8(2):22–31, 2009.

[Pas98] J. Pascoe. Adding generic contextual capabilities to wearable computers. In *Wearable Computers, 1998. Digest of Papers. Second International Symposium on*, pages 92–99, Oct 1998.

[PBA+03] Ariel Pashtan, Remy Blattler, Andi, Andi Heusser, and Peter Scheuermann. Catis: A context-aware tourist information system. 2003.

[Rai10] Ruby on rails: Applications. http://rubyonrails.org/applications, 2010. Accessed 25th June 2010.

[RPM97] N. Ryan, J. Pascoe, and D. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications and Quantitative Methods in Archaeology (CAA97), Oxford*, 1997.

[RTH09] Sam Ruby, Dave Thomas, and David Heinemeier Hansson. *Agile Web Development with Rails*. Pragmatic Programmers, third edition, March 2009.

[SDA99] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. The context toolkit: aiding the development of context-enabled applications. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 434–441, New York, NY, USA, 1999. ACM.

[SP04] Thomas Strang and Claudia L. Popien. A context modeling survey, September 2004.

[ST94] Bill Schilit and Marvin Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8:22–32, 1994.

[URI10] Uniform Resource Identifiers (URI): Generic Syntax. `http://www.ietf.org/rfc/rfc2396.txt`, 2010. Accessed 27th June 2010.

[VMG+01] K. Virrantaus, J. Markkula, A. Garmash, V. Terziyan, J. Veijalainen, A. Katanosov, and H. Tirri. Developing gis-supported location-based services. In *Web Information Systems Engineering, 2001. Proceedings of the Second International Conference on Web Information Systems*, volume 2, pages 66–75 vol.2, Dec 2001.

[W3C10] W3C. Extensible markup language (XML). `http://www.w3.org/XML/`, 2010. Accessed 27th June 2010.

REFERENCES

Appendix A

Sample JSON: Available Package listing


```
[
    {
        "package": {
            "name": "FraunhoferAICOS",
            "updated_at": "2010-06-01T18:23:02Z",
            "id": 3,
            "version": 6
        }
    },
    {
        "package": {
            "name": "FEUP",
            "updated_at": "2010-05-25T19:25:15Z",
            "id": 4,
            "version": 2
        }
    },
    {
        "package": {
            "name": "Penafiel",
            "updated_at": "2010-05-25T20:09:42Z",
            "id": 11,
            "version": 3
        }
    },
    {
        "package": {
            "name": "Paris",
            "updated_at": "2010-06-25T17:05:13Z",
            "id": 12,
            "version": 1
        }
    }
]
```

Sample JSON: Available Package listing

Appendix B

Sample JSON: FraunhoferAICOS Package information


```
{
    "package": {
        "name": "FraunhoferAICOS",
        "version": 8,
        "updated_at": "2010-07-06T02:51:15Z",
        "content_file_name": "FraunhoferAICOS-8.zip",
        "content_file_size": 6403947,
        "locations": [
            {
                "name": "Room011",
                "coordinates": "",
                "tags": "",
                "contents": [
                    {
                        "name": "foliveira.html",
                        "tags": "",
                        "mimetype": "text/html",
                        "path": "719f9330aa2b0c91.html"
                    },
                    {
                        "name": "ccardoso.html",
                        "tags": "",
                        "mimetype": "text/html",
                        "path": "e8617c4e1c3d686c.html"
                    },
                    {
                        "name": "hpeixoto.html",
                        "tags": "",
                        "mimetype": "text/html",
                        "path": "7b698a9e0e144cfc.html"
                    },
                    {
                        "name": "jsantos.html",
                        "tags": "",
                        "mimetype": "text/html",
                        "path": "4823aac710408c85.html"
                    },
                    {
                        "name": "msilva.html",
                        "tags": "",
                        "mimetype": "text/html",
```

```
                "path": "3f0a1ea316d79a83.html"
            },
            {
                "name": "pcoelho.html",
                "tags": "",
                "mimetype": "text/html",
                "path": "4c5907f38c7f2612.html"
            },
            {
                "name": "pteixeira.html",
                "tags": "",
                "mimetype": "text/html",
                "path": "4b0ead941d53bdf1.html"
            },
            {
                "name": "xico.html",
                "tags": "",
                "mimetype": "text/html",
                "path": "97e7bd46ff329b7b.html"
            },
            {
                "name": "gaivotas",
                "tags": "teste",
                "mimetype": "image/jpeg",
                "path": "e4f13752e0e72425.jpg"
            },
            {
                "name": "Photo: Room011",
                "tags": "",
                "mimetype": "image/jpeg",
                "path": "1653d055cd1c26ed.jpg"
            }
        ]
    },
    {
        "name": "Room010",
        "coordinates": "",
        "tags": "",
        "contents": [
            {
                "name": "fabrantes.html",
                "tags": "",
                "mimetype": "text/html",
                "path": "d6df09810b5c03d6.html"
            },
            {
                "name": "psaleiro.html",
                "tags": "",
                "mimetype": "text/html",
                "path": "6952d9a93833c26b.html"
            },
            {
                "name": "Photo: Pedro Saleiro",
                "tags": "",
                "mimetype": "image/jpeg",
                "path": "51f79d8d7c11ebb3.jpg"
            }
        ]
    },
    {
```

```
        "name": "Room011Desk01",
        "coordinates": "",
        "tags": "",
        "contents": [
            {
                "name": "",
                "tags": "",
                "mimetype": "text/html",
                "path": "hpeixoto.html"
            },
            {
                "name": "",
                "tags": "",
                "mimetype": "image/jpeg",
                "path": "photo.jpg"
            },
            {
                "name": "",
                "tags": "",
                "mimetype": "image/png",
                "path": "project.png"
            }
        ]
    },
    {
        "name": "Room011Desk03",
        "coordinates": "",
        "tags": "",
        "contents": [
            {
                "name": "",
                "tags": "",
                "mimetype": "image/png",
                "path": "omgrecursion.png"
            },
            {
                "name": "",
                "tags": "",
                "mimetype": "text/html",
                "path": "pcoelho.html"
            },
            {
                "name": "Photo: Me",
                "tags": "",
                "mimetype": "image/jpeg",
                "path": "347be423cb5de2b5.jpg"
            }
        ]
    },
    {
        "name": "Room010Desk01",
        "coordinates": "",
        "tags": "",
        "contents": [
            {
                "name": "",
                "tags": "",
                "mimetype": "image/png",
                "path": "photo.png"
            },
```

```
            {
                "name": "",
                "tags": "",
                "mimetype": "text/html",
                "path": "filipe_abrantes.html"
            }
        ]
    },
    {
        "name": "Room011Desk04",
        "coordinates": "",
        "tags": "",
        "contents": [
            {
                "name": "",
                "tags": "",
                "mimetype": "text/html",
                "path": "ecaalyx.html"
            },
            {
                "name": "",
                "tags": "",
                "mimetype": "text/html",
                "path": "jsantos.html"
            },
            {
                "name": "",
                "tags": "",
                "mimetype": "image/gif",
                "path": "logo.gif"
            },
            {
                "name": "",
                "tags": "",
                "mimetype": "image/jpeg",
                "path": "me3b.jpg"
            }
        ]
    },
    {
        "name": "Room011Desk09",
        "coordinates": "",
        "tags": "",
        "contents": [
            {
                "name": "",
                "tags": "",
                "mimetype": "image/jpeg",
                "path": "anymails.jpg"
            },
            {
                "name": "",
                "tags": "",
                "mimetype": "text/html",
                "path": "ccardoso.html"
            },
            {
                "name": "",
                "tags": "",
                "mimetype": "image/jpeg",
```

```
                "path": "celso.jpg"
            }
        ]
    },
    {
        "name": "Room011Desk10",
        "coordinates": "",
        "tags": "",
        "contents": [
            {
                "name": "msilva.html",
                "tags": "",
                "mimetype": "text/html",
                "path": "msilva.html"
            }
        ]
    },
    {
        "name": "Room011Desk11",
        "coordinates": "",
        "tags": "",
        "contents": [
            {
                "name": "About: Chico",
                "tags": "",
                "mimetype": "text/html",
                "path": "160528bce3ebfbe1.html"
            },
            {
                "name": "Chico's photo",
                "tags": "",
                "mimetype": "image/jpeg",
                "path": "17142a565bf71952.jpg"
            }
        ]
    },
    {
        "name": "Room009",
        "coordinates": "",
        "tags": "",
        "contents": [
            {
                "name": "Room009",
                "tags": "",
                "mimetype": "text/html",
                "path": "5601de3f75af8b1d.html"
            },
            {
                "name": "Ricardo Duarte",
                "tags": "",
                "mimetype": "text/html",
                "path": "14cf1b3c9a2a8fab.html"
            },
            {
                "name": "Ricardo Melo",
                "tags": "",
                "mimetype": "text/html",
                "path": "d915cec20d94170a.html"
            },
            {
```

```
            "name": "Cl\u00e1udia Peixoto",
            "tags": "",
            "mimetype": "text/html",
            "path": "a7d911ffe050a2f2.html"
        },
        {
            "name": "Fraunhofer AICOS",
            "tags": "",
            "mimetype": "image/png",
            "path": "e9ff09bcd25dc133.png"
        },
        {
            "name": "Photo: Ricardo Duarte",
            "tags": "",
            "mimetype": "image/jpeg",
            "path": "2ecd9ba2e16ee073.jpg"
        },
        {
            "name": "Photo: Ricardo Melo",
            "tags": "",
            "mimetype": "image/jpeg",
            "path": "7ec12f34c36c6efd.jpg"
        },
        {
            "name": "Photo: Dami\u00e3o",
            "tags": "",
            "mimetype": "image/jpeg",
            "path": "28f06359aa8b558b.jpg"
        },
        {
            "name": "Photo: Room009",
            "tags": "",
            "mimetype": "image/jpeg",
            "path": "8ed05d1b2660ed20.jpg"
        }
    ]
},
{
    "name": "Room008",
    "coordinates": "",
    "tags": "",
    "contents": [
        {
            "name": "Room008",
            "tags": "",
            "mimetype": "text/html",
            "path": "379aa5768e127114.html"
        },
        {
            "name": "Fraunhofer AICOS",
            "tags": "",
            "mimetype": "image/png",
            "path": "184a1c195bf25864.png"
        }
    ]
},
{
    "name": "Room007",
    "coordinates": "",
    "tags": "",
```

```
        "contents": [
            {
                "name": "Room007",
                "tags": "",
                "mimetype": "text/html",
                "path": "602cd14558475473.html"
            },
            {
                "name": "Fraunhofer AICOS",
                "tags": "",
                "mimetype": "image/png",
                "path": "f219dbe7786925ec.png"
            },
            {
                "name": "Photo: Room007",
                "tags": "",
                "mimetype": "image/jpeg",
                "path": "8c41512f7c4c834e.jpg"
            }
        ]
    },
    {
        "name": "Room002",
        "coordinates": "",
        "tags": "",
        "contents": [
            {
                "name": "Room002",
                "tags": "",
                "mimetype": "text/html",
                "path": "eca85860cbe2b07c.html"
            },
            {
                "name": "Fraunhofer AICOS",
                "tags": "",
                "mimetype": "image/png",
                "path": "952f485b923e9d0b.png"
            },
            {
                "name": "About: Naviporto",
                "tags": "",
                "mimetype": "text/html",
                "path": "e16d3aed8dd2c2bd.html"
            },
            {
                "name": "Photo: Naviporto Posters",
                "tags": "",
                "mimetype": "image/jpeg",
                "path": "7c958f40d079a82e.jpg"
            },
            {
                "name": "Photo: Naviporto Booth",
                "tags": "",
                "mimetype": "image/jpeg",
                "path": "3cd2de3501e1c979.jpg"
            },
            {
                "name": "Photo: Room002",
                "tags": "",
                "mimetype": "image/jpeg",
```

```
            "path": "5f9a52ecd129d9cf.jpg"
        }
    ]
},
{
    "name": "Room010Desk03",
    "coordinates": "",
    "tags": "",
    "contents": [

    ]
},
{
    "name": "Room010Desk04",
    "coordinates": "",
    "tags": "",
    "contents": [

    ]
},
{
    "name": "Room005",
    "coordinates": "",
    "tags": "",
    "contents": [
        {
            "name": "Photo: Room005",
            "tags": "",
            "mimetype": "image/jpeg",
            "path": "fb933d81c7bd8a5f.jpg"
        },
        {
            "name": "Photo: Carlos Resende",
            "tags": "",
            "mimetype": "image/jpeg",
            "path": "705bf4291ea92953.jpg"
        }
    ]
},
{
    "name": "Room003",
    "coordinates": "",
    "tags": "",
    "contents": [
        {
            "name": "Photo: Room003",
            "tags": "",
            "mimetype": "image/jpeg",
            "path": "1ade8fcbd381d92d.jpg"
        }
    ]
},
{
    "name": "Corridor",
    "coordinates": "",
    "tags": "",
    "contents": [
        {
            "name": "Photo: Entry",
            "tags": "",
```

```
                        "mimetype": "image/jpeg",
                        "path": "caf5561199ef1657.jpg"
                    },
                    {
                        "name": "Photo: Alcohol Dispenser",
                        "tags": "",
                        "mimetype": "image/jpeg",
                        "path": "d991aa1a78553ad3.jpg"
                    }
                ]
            }
        ]
    }
}
```

Sample JSON: FraunhoferAICOS Package information

Appendix C

Content listing for the FraunhoferAICOS Package

Table C.1: Content listing for the FraunhoferAICOS Package

Content name	File name	Mime type	Size (bytes)
Untitled	photo.png	image/png	209030
Untitled	filipe_abrantes.html	text/html	358
Untitled	hpeixoto.html	text/html	146
Untitled	photo.jpg	image/jpeg	37732
Untitled	project.png	image/png	317705
Untitled	omgrecursion.png	image/png	9350
Untitled	pcoelho.html	text/html	175
Untitled	ecaalyx.html	text/html	1107
Untitled	jsantos.html	text/html	242
Untitled	logo.gif	image/gif	6578
Untitled	me3b.jpg	image/jpeg	18339
Untitled	anymails.jpg	image/jpeg	22063
Untitled	ccardoso.html	text/html	175
Untitled	celso.jpg	image/jpeg	1443638
msilva.html	msilva.html	text/html	280
fabrantes.html	d6df09810b5c03d6.html	text/html	358
psaleiro.html	6952d9a93833c26b.html	text/html	260
foliveira.html	719f9330aa2b0c91.html	text/html	179
ccardoso.html	e8617c4e1c3d686c.html	text/html	175
hpeixoto.html	7b698a9e0e144cfc.html	text/html	146
jsantos.html	4823aac710408c85.html	text/html	242
msilva.html	3f0a1ea316d79a83.html	text/html	280
pcoelho.html	4c5907f38c7f2612.html	text/html	175
pteixeira.html	4b0ead941d53bdf1.html	text/html	154
About: Chico	160528bce3ebfbe1.html	text/html	292
Chico's photo	17142a565bf71952.jpg	image/jpeg	2220
xico.html	97e7bd46ff329b7b.html	text/html	292
gaivotas	e4f13752e0e72425.jpg	image/jpeg	1054359
Room009	5601de3f75af8b1d.html	text/html	261
Ricardo Duarte	14cf1b3c9a2a8fab.html	text/html	274
Ricardo Melo	d915cec20d94170a.html	text/html	262
Cláudia Peixoto	a7d911ffe050a2f2.html	text/html	273
Room008	379aa5768e127114.html	text/html	241
Room007	602cd14558475473.html	text/html	835
Room002	eca85860cbe2b07c.html	text/html	834
Fraunhofer AICOS	e9ff09bcd25dc133.png	image/png	2786
Fraunhofer AICOS	184a1c195bf25864.png	image/png	2786

Fraunhofer AICOS	f219dbe7786925ec.png	image/png	2786
Fraunhofer AICOS	952f485b923e9d0b.png	image/png	2786
About: Naviporto	e16d3aed8dd2c2bd.html	text/html	837
Photo: Room005	fb933d81c7bd8a5f.jpg	image/jpeg	219673
Photo: Ricardo Duarte	2ecd9ba2e16ee073.jpg	image/jpeg	242192
Photo: Ricardo Melo	7ec12f34c36c6efd.jpg	image/jpeg	210332
Photo: Damião	28f06359aa8b558b.jpg	image/jpeg	213622
Photo: Room009	8ed05d1b2660ed20.jpg	image/jpeg	215852
Photo: Room011	1653d055cd1c26ed.jpg	image/jpeg	302832
Photo: Pedro Saleiro	51f79d8d7c11ebb3.jpg	image/jpeg	217153
Photo: Carlos Resende	705bf4291ea92953.jpg	image/jpeg	217537
Photo: Me	347be423cb5de2b5.jpg	image/jpeg	21484
Photo: Naviporto Posters	7c958f40d079a82e.jpg	image/jpeg	289325
Photo: Naviporto Booth	3cd2de3501e1c979.jpg	image/jpeg	248291
Photo: Room002	5f9a52ecd129d9cf.jpg	image/jpeg	255344
Photo: Room003	1ade8fcbd381d92d.jpg	image/jpeg	203137
Photo: Room007	8c41512f7c4c834e.jpg	image/jpeg	240389
Photo: Entry	caf5561199ef1657.jpg	image/jpeg	228479
Photo: Alcohol Dispenser	d991aa1a78553ad3.jpg	image/jpeg	243715

Appendix D

HTC Magic Specifications

Table D.1: HTC Magic Technical Specifications

Processor	Qualcomm® MSM7200A™, 528 MHz
Operating System	Android
ROM Memory	512 MB
RAM Memory	288 MB
Dimensions	113 x 55.56 x 13.65 mm (4.45 x 2.19 x 0.54 inches)
Weight	116 grams (4.09 ounces) with battery
Display	3.2-inch TFT-LCD flat touch-sensitive screen with 320x480 HVGA resolution
Network	• HSDPA/WCDMA: 900/2100 MHz • Up to 2 Mbps up-link and 7.2 Mbps down-link speeds • Quad-band GSM/GPRS/EDGE: 850/900/1800/1900 MHz
Device Control	Trackball with Enter button
GPS	Internal GPS antenna
Connectivity	• Bluetooth® 2.0 with Enhanced Data Rate and A2DP for wireless stereo headsets Wi-Fi®: IEEE 802.11 b/g • HTC ExtUSB™ (11-pin mini-USB 2.0 and audio jack in one)
Camera	3.2 megapixel color camera with auto focus
Audio supported formats	AAC, AAC+, AMR-NB, MP3, WMA, WAV, AAC-LC, MIDI, OGG Video, MP4, 3GP
Battery	• Rechargeable Lithium-ion battery • Capacity: 1340 mAh
Expansion Slot	microSD™ memory card (SD 2.0 compatible)
AC Adapter	Voltage range/frequency: 100 240V AC, 47/63 Hz
Special Features	G-sensor, Digital Compass