Application of a Context Model in Context-Aware Mobile Government Services

César E. Ariza Avila

Dissertação submetida à Universidade do Minho para obtenção do grau de Mestre em Sistemas de Informação elaborada sob a orientação da
Doutora Helena Rodrigues

Departamento de Sistemas de Informação
Escola de Engenharia
Universidade do Minho
Guimarães, Outubro de 2006

Acknowledgements

I would like to express my gratitude to all people that directly or indirectly contribute to the execution of this work.

Special thanks to my supervisor Doutora Helena Rodrigues for creating the environment for doing this research.

Special thanks also and again to Helena Rodrigues and Jason Pascoe for the revisions, suggestions and supporting in order to finish this dissertation.

Special thanks to Helder Pinto and the researchers of the Ubicomp Laboratory at University of Minho.

I would like to thank, to the Information Systems Department for creating the environment needed to this research.

Aplicação de um Modelo de Contexto em Serviços Moveis Baseados no Contexto para o governo electrónico

Resumo

As novas tecnologias móveis oferecem um elevado potencial para a interacção e a participação dos cidadãos com as autoridades locais. Estas tecnologias são exploradas nesta dissertação, tendo como objectivo promover e melhorar esta interacção, suportando e criando formas novas de comunicação e tentando tomar vantagem da mobilidade tanto quanto possível.

As aplicações baseadas no contexto utilizam o contexto para facilitar e melhorar a experiência do utilizador Estas aplicações, combinadas com as novas tecnologias móveis, podem simplificar os processos de comunicação com o governo.

Esta dissertação apresenta as áreas chaves a considerar no desenvolvimento de uma aplicação móvel baseada no contexto. Inicialmente são apresentados os principais requisitos identificados para tais aplicações. Contexto é descrito então como um modelo do ambiente circunvizinho, e um modelo de objectos é apresentado a fim de demonstrar esta visão. Para a implementação do modelo de objectos foi utilizado RDF, que permite a representação de conhecimento. Um conjunto de aplicações centrais foi desenvolvido a fim de suportar o modelo de contexto e a criação de aplicações baseados no contexto. As aplicações foram testadas no município de Vila Nova de Cerveira, com a contribuição de um conjunto de cidadãos. Por último, os cenários de teste, assim como os resultados de usabilidade são apresentados.

Application of a Context Model in Context-Aware Mobile Government Services

Abstract

New mobile technologies offer a wide potential for interaction and participation between citizens and local authorities. These technologies are explored and exploited in this thesis with the aim of promoting and improving this interaction by supporting and creating novel ways of communication and taking as much advantage of mobility as much as possible.

Context Aware Applications utilise context to facilitate and improve the user experience. These applications, combined with the new mobile technologies used by citizens, can simplify their process of communication with government bodies.

This dissertation presents the key areas for consideration in developing a Context Aware Mobile Government Application. Initially we present the central requirements we identified for such applications. Our understanding of context is then described as a model of the surrounding environment, and an object model in particular is presented in order to realize this understanding. The object model was implemented using RDF, which allows the representation of knowledge, and a set of core applications were developed in order to support the context model and the creation of new context aware applications. These were tested in a trial implementation at the municipality of Vila Nova de Cerveira, which we present along with some user studies that we conducted with the help of some citizens of the municipality.

Contents

List of figures

List of tables

1 Introduction

New mobile technologies offer a wide potential for interaction and participation between citizens and local authorities. Those technologies will be explored and exploited in order to promote and improve this interaction, supporting and creating novel ways of communication and taking as much advantage of mobility as possible [1].

A good example of citizen interaction is a spontaneous participation in the community, in which citizens may have to provide, or be asked for, information about their current situation (i.e. a traffic jam, a crash, a fire, a parade, a commendation, etc). The challenge is to allow the citizen to react immediately, in the right place and at the right time. Such spontaneous participation inside a community can be seen as a new application area.

Local governments [1] like to provide communication channels to the citizens in order to improve the relationship between citizens and government. Those channels can be electronic services such as web portals, mobile services or, face to face services. In any case the citizen must know which service or channel can use and when those services are available. For instance, if a citizen wants to interact with the government in order to inform them of an anomaly or ask for information, the citizen must find the appropriate person or department to communicate with. Also the communication must be done in the appropriate time (e.g. office hours). In many cases this could discourage the citizen from contacting a government service. In the case of mobile services, the service selection can be improved if citizens are provided with a tool that helps them to discover the right services they need when the citizens want to make a spontaneous participation. This tool could take the form of a set of applications that discover or help to discover services on behalf of the user by using contextual information.

Spontaneous participation from citizens in a community faces two somewhat contradictory requirements. Firstly, it is imperative to acquire as much information as possible from situations. This information does not only include data submitted by citizens (i.e. text, pictures or video sent via SMS and MMS), but must also include information about the context, for instance the time and location of the participation. If this information can be represented in a rich and coherent structure then we expect that there will arise many useful new services that could exploit it. Secondly, a spontaneous

participation needs simple and easy-to-use interaction mechanisms to attract and retain as many users as possible[2].

A central issue in providing these kind of services is to accurately ascertain the context of the users based on their situation. For example in a complaint-scenario, the users location is useful in determining the topic of the complaint. But there are difficulties in getting accurate location info. For example the user can send a complaint via SMS but the location granularity provided by mobile operators is only available to the level of a mobile cell. In this situation other kind of context information are also necessary and useful, such as the type of place where the user is located (i.e. a garden, a park, a hospital, a rural zone, etc). And there are similar challenges to ascertain and represent these types too. An equally important issue is how to model and represent the context of both the user and the services provided, which includes how to describe the user, how to describe the nearby objects, how to describe the user situation and how to describe the service context (i.e. the context in which a particular local service is useful).

Finally, when the user context is known, it is necessary to find a match between the user context and the service context in order to provide an appropriate service to the user.

1.1 The USE-ME.GOV project

The USability-drivEn open platform for the MobilE GOVernment consortium (USEME.GOV project) [3] aims to develop a platform to enable the easy development and deployment of local government applications/services that citizens can access via their mobile devices (e.g. smartphones or PDAs).

The project vision is centred on the provision of appropriate services (i.e. context-dependent services) to the citizen directly in the time and place they are needed or useful [2]. One component of the USE-ME.GOV platform concept was a context model to support context-aware behaviours of the various services through the provision of a model of the surrounding environment [4].

Municipalities (or other government bodies) may have many services to offer their citizens. Consider for example, if the citizen wants to report an anomaly. In this case the citizen must know the department to interact with and the specific information to provide. It is also likely that the information given by the citizen must be processed and delivered to the correct responsible party. In the case of reporting an anomaly, for example, to a

damaged bench in a garden, the information must be delivered to the responsible department of gardens. This plethora of services, procedures and situations may be overwhelming for the citizen.

However, it is possible to help the Citizen and even the Municipality to select the correct service. In the garden scenario, if the citizen wants to report an anomaly immediately they could use their smart phone to access the anomaly report services that the municipality provide for that. Furthermore, by using the context information of the citizen (e.g. user location) it is possible to suppose that in the case of the garden complaint situation, that the user wants to inform the local government something related the garden. So using context information the services that will appear in the smart phone of the citizen will be only those related to the citizen's context, such as a service service to report an anomaly for the garden.

Using contextual information benefits the citizens as well the Municipality or government. Some of the benefits derived from the use of contextual information can be: speeding up the problem resolution, simplifying the process for citizens and governments, encouraging the participation of the citizens, making easy the citizen to access to the services [1].

One of the goals of the USEME.GOV platform is to support the service discovery on behalf of the user. This service discovery is based on a component that utilises the contextual information of the user and services.

This dissertation forms part of the work developed in the USE-ME.GOV project, particular with respect to the components in charge of context information.

1.2 Context Awareness Concepts

1.2.1 Context in aware computing

According to the Merriam-Webster dictionary [5] the term "context" has two meanings:

1. *the parts of a disclosure that surround a word or passage and can throw light on its meaning.*
2. *the interrelated conditions in which something exists or occurs.*

The context concept is widely used in grammar. In some cases it is possible to know the meaning of a word by using its context. By analogy it is possible to know some facts about other entities by knowing their context. For instance if someone wearing overalls it is possible that he or she may be working. Context in this document is related to the context used in context-awareness computing.

Many definitions of context have been proposed in context-aware computing research. A widely accepted definition proposed by *Dey* and *Abowd* is:

> *"Context: any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computational and physical objects."*[6].

However the definition above is very general and covers a lot of information. For this reason several types of context (sometimes known as context dimensions or categories) have been defined, such as a time context, location context, activity context, physical context, etc [2,3]. These concepts help to clarify the nature of context but they do not really address the issue of how to represent it.

There already exists several forms of representation for different types of context. For example, the location context of entities (such as objects, persons, or services) can be well expressed using GML (Geography Markup Language)[4,5] among other standards. A higher-level model is still required though in order to bring these different pieces together and to offer a unified model that may support all types of entities and contexts. The semantic web offers a means of doing so through the use of ontologies to model and represent context and related concepts [6], and also utilises previously acquired knowledge expressed as ontologies [7].

Based on these definitions, the general needs of a context aware system are:

i. context acquisition: how to obtain the contextual information,
ii. context representation: how to organize and store the context information;
iii. context use: how to use the context information in an appropriate manner.

One of the objectives of this work is to present a solution to the three needs presented above in a specific application scenario. The specific scenario is the participation of the

citizen in the community by interacting with the government. This work will also present the three basic needs about, in the form of requirements in the requirements chapter.

1.2.2 Context categorization and acquisition

There is not a unified classification of context. Many authors define those categories depending on the use of the context information. Shilit et. al. [7] defined three categories of context: user context (user's profile, user's location, social situation, etc), computing context (net connectivity, communication costs, com. bandwidth), and physical context (lighting, noise levels). Dey and Abowd [6] define four main types of context: location, identity, activity, and time.

In this dissertation the term context dimension is used as context type or context category, referring to the kind of contextual information. For instance, the user feelings, the user activity, the user location, etc. are all kinds of context dimensions.

Contextual information can be acquired in several forms. For instance the altitude can be acquired by using an altimeter or a GPS, or even by heating water and calculating the altitude by measuring its boiling point with a mercury thermometer.

Context acquisition refers to the form the contexts is acquired. Mostéfaoui et. al. [8] defined three ways of context acquisition:

- *Sensed context*: This kind of information is acquired by means of physical sensors, such as for temperature and pressure.
- *Derived context:* Is the information computed on the fly, such as time and date.
- *Context explicitly provided:* For instance the user preferences when they are explicitly provided to the application.

The above mentioned categories of context and context acquisition help us to realize how the context can be acquired in this work. Cities do not have an infrastructure for context sensing and acquisition. Smart spaces, instead, are provided with sensors that acquire contextual information such as temperature, noise levels, location, light intensity, etc. Also the people that use those spaces in some way have artefacts that make possible to know their position. Those artefacts can be cameras or electronic tags such RFID, or even Bluetooth devices. Cities are not smart spaces. If some sensing devices exist in cities they are often not accessible or usable. For instance, if the time is

considered as a context dimension, certainly the watch in the tower of the church can be hardly useful. Also Citizens lack of devices needed to contextualize in a city. However, in order to contextualize citizens or entities in a city, one of the possibility is to use (or reuse) infrastructure not intended for contextualization.

The way citizens can be contextualized is unpredictable. Nowadays citizen environments have restrictions such as the lack of sensors and the lack of accuracy in the sensed context dimensions. Assuming the citizens have as a minimum a mobile phone, in some way the position of the citizen (even with a really bad accuracy) can be revealed. Citizens with more sophisticated devices – with embedded GPS devices- can be located with a better accuracy. However, location is only one of the context dimensions, there are other dimensions, such as activity, citizen feelings, etc. If it is not possible to acquire all the required context dimensions there remains the option to ask to the user for their context.

1.3 Context aware systems

Context aware applications are those that uses context, more specifically Shilit [7] defined context aware systems as:

> *"A context-aware system is one that can determine and react to the current physical and computing context of mobile users and devices, by altering the information presented to users or commands issued by and on behalf of those users."*

Dey and Abowd[6] defined a context aware system as:

> *"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."*

Research into frameworks and models to support context-awareness in applications has tended to concentrate on particular niches.

For example, the GUIDE project [9] developed a guide for tourists in the city of Lancaster, in which, based on the user preferences and location, the application will create an appropriate guide tour. GUIDE was developed using an application centric

approach with any context modelling component embedded inside the application. The user preferences are obtained directly from the user and the location information is obtained by receiving location messages transmitted from strategically positioned Wi-Fi (IEEE 802.11) base stations. The physical device used was a touch-screen sub-notebook (Fujitsu TeamPad 7600 with 850g), with the software limited to this specific device.

The Context toolkit (CTK) [10] dealt primarily with handling and conversion of sensor data. The CTK proposes a network of widgets, some widgets dealing directly with sensors. The widgets communicate the contextual information to an intermediate layer using a publish/subscribe mechanism. In this approach the contextual information is categorized using a taxonomy. One of the advantages of the CTK is the separation to the applications about the context acquisition tasks and interpretation.

Two broader approaches to modelling context are SOUPA (Standard Ontology for Ubiquitous and Pervasive Applications) [11] and CoOL (Context Ontology Language to enable Contextual Interoperability) [12]. SOUPA is designed to model and support pervasive computing applications in the form of a set of ontologies that are expressed using the Web Ontology Language (OWL). SOUPA is composed of a set of core ontologies and extension ontologies. The core set includes ontologies to represent persons, events, actions, space, time and the proposed extension ontologies include location, devices, schedule, etc. These extension can be added to further, and this possibility to extend SOUPA is one of its advantages. In fact the core ontologies reuse some pre existing made ontologies such as the Friend-Of-A-Friend ontology (FOAF), DAMLTime and the entry sub-ontology of time and the spatial OpenCyc. There are developed applications that use SOUPA such as CoBrA [13]. CoBrA is a broker-centric agent architecture for supporting context-aware systems in smart spaces.

CoOL employs semantic web techniques also. CoOL uses OWL-S and its principal aim is the interoperability between web services. CoOL focuses in the description of the “semantics” (semantics refers to how to invoke the service, data types of the parameters, and data types of the received answers) of the services by using OWL-S. CoOL also allows to describe the context of use of the services by using aspects. Those descriptions are used to discover services and, once discovered, to invoke that services. The discovery process is performed by matching a given context (represented as aspects) with the service context description.

To summarize, GUIDE uses an applicational centric architecture and was developed for a specific device. The CTK is a good approach because it separates and hides the context acquisition from the applications, and encapsulates each context dimension in a software component (widget). SOUPA proposes the use of ontologies to model context and is used in smart spaces application. CoOL focuses on web services description and service discovering. All those approaches has its niche and contribute important ideas such as the formal representation of context, service discovery and representation, and the separation between context acquisition and the context use. These important ideas will be reflected in the development of this work.

1.4 Objectives of this dissertation

The main objective of this dissertation is to explore the manner in which to design and develop a Context-Aware mobile government application. The specific goals are:

- To analyze the context requirements for context aware.
- To apply one approach of representing and modelling context. Context representation can be further divided into (a) context about objects (persons and physical objects) and (b) context about services that will be offered to these objects.
- To implement a mechanism to discover what services are useful for which users depending on their context.
- To deploy and to analyze the results of this deployment.

The structure of this dissertation is as follows: the first chapter contains the introduction; the second chapter presents some concerns and challenges about building context aware applications; the third chapter presents the context object model as a way of modelling context; the fourth chapter presents one computational artefact that manages the context object model, the fifth chapter describes the creation of an context aware application and finally conclusions are presented in the last chapter.

2 Context management applications requirements

2.1 Introduction

In design of a context aware mobile government application (CAMGOV) several approaches can be used to identify its requirements. There are methodologies for software requirements specification [14], however in the area of context aware computing those guidelines are not suitable directly in a design and development process; as stated in [10] and from the experience gained in the USE-ME.GOV project. For instance in a retail business application, there are only well known operations and data to model, basically products and data related with that business such as providers, clients, inventory, sales, etc. It is not hard to affirm that all of the retail commerce sector has a similar data model; for this kind of traditional application conventional methodology is used for requirements specification and modelling.

Unlike the retail business applications, there is not a methodology for requirements specification and design of context aware applications. These applications are still under research and therefore there is not a unified view of what is a context aware application and what is the contextual information, which makes difficult the creation of this kind of application. Moreover, the data in these kind of applications can vary greatly and it is therefore very difficult to use conventional structured data types for this kind of information. Proof of that is the great number of approaches used for modelling context [15], and, for this reason a non empirical methodology is used to find out the context requirements for context aware applications.

By analysing constraints it is possible to find requirements; moreover the nature of the application implies some tacit constraints too. The identified constraints of a context aware mobile government application, in brief, are: the application must be able to acquire context information, process context information and allow using context information. Furthermore, it is necessary the representation of the context information. From another perspective, requirements can be divided into two groups, one related with context information and another related with the application that manages the context. Based in the main constraints above and in other research work, the following is the list of the key requirements for a CAMGOV.

2.2 Context Information Requirements

The following sections describe the requirements related about the contextual information.

2.2.1 Model for context Information

***Requirement**: A model for context information is necessary.*

In order to use computer technology with the aim to mechanize some process it is necessary to have a model of the players involved in the process. For instance in a selling process the invoices or receipt is filled. The information found in those documents is normally related to the client and the products. This form can be filled in by hand. In the case a computer fills and prints the form, the client and the product information can modelled in form of entities and relations and those entities and relations stored as tables in a database. Similarly we need to model the context information in order to have a common understanding of how the context is used in the management system.

A good model makes smoother the representation of the real word by using data structures, for instance a model to represent location must have the possibility to represent symbolic location information such as the “Eiffel Tower” and non symbolic location information such as coordinates (i.e. latitude, longitude and altitude).

There are many approaches to model context information [15] and generally the model is very related with the way the context is represented and stored. Earlier approaches propose the use of dynamic tags associated with the objects [16], more recent works such as Dey’s Context Toolkit [10] use a taxonomy for context categorization and define attributes of objects merely answering questions like “who”, “where”, “what”, “when” and not defining formal model for context. The context toolkit also allows components (widgets) to create derived context information, this derived context information is not well supported by the taxonomy model and is specific for the application where the context is used.

The Context Ontology Language (CoOL) [12] is another approach for modeling context; CoOL model is very good at describing services because the ontology is based in OWL-S (formery DAML-S). OWL-S [17] is a language for describing web services in an unambiguous form. Chen et al. [11] and Wang et al. [18] propose the use of ontologies for context modelling, both models are very rich for modelling the world, and present a formal representation of context; in both approaches the prototypes that support their

systems are based in relatively small spaces (confined spaces) such as a kitchen or a laboratory-room, those places provided with several sensors as sources of context information. The differences in the scope of application between the two mentioned ontology approaches and the CAMGOV are the quantity of sensors used to acquire context, in [11] and [18] the number of sensors is greater than compared to the number of sensors in a city.
Additionally the specific places they consider, such as a kitchen, seems to have less activities (i.e. cooking, blending, washing) per person than in a open space as a city. However the formalism of the model approaches used in [11] and [18] (by using OWL to represent objects and services) serves as example about the importance to create a model for context in a CAMGOV.

2.2.2 Representing and storing context Information

***Requirement**: It is necessary to represent the model for context information using adequate data structures.*

Once the model for context is defined, the next step is to define data structures to represent this information. For example, location could be modelled as symbolic and non symbolic information. That could represented in files that use the GML[19] schema to represent location.

In order to use context information, it is necessary to find out how to represent and store (give persistency) the context information model. Representing and storing is necessary to keep historical information and for the easy querying of the data. The importance of how the model is represented is the feasibility to map *entities* of the model into *data-entities* of the representation, for instance in a library, books and readers have their representation as entities in a relational model; fields such as title, author and price are expected for books; and for the users name, address; borrowing is a relation between books and readers. The possibility the storing-artefact brings to access the information is a key characteristic in the selection of this artefact. For instance in a library; the entity relational model is stored in a relational database, *books* and *users* may have a table and may be the "*borrowing-table*" exist or not.

The form to represent and store context information is dependent of the structure of the model. For example, it is possible to use a database or XML files to store the same contextual objects. If the context is represented using ER (Entity Relation) models as

cited in [20], ER databases seem to be the best way to store and query context representation, because exists the "*know how*" about querying ER-databases. However in [21] the authors model context using OWL and for representation translate the OWL documents to F-Logic documents in order to improve use effectiveness due to the third party backend components that are used.

There are expected characteristics of the selected way to represent and store the context information, such as was mentioned before, and including the feasibility to query data. For instance the easy way the data is managed using a SQL capable database makes it unthinkable not to use a databases in traditional applications.

2.3 Application Context Requirements

Beyond the model representation, it is also necessary to have an application to manage the model. This application must make usable the information represented by the model to other applications. The management system basically needs to enable other applications to create, to modify, to query and to update the stored context information. This access to the context information must be as simply as possible. There are two potential client of the management application: creators of context information and the consumers of context information. Next are presented requirements for the management system included those concerning to consumers and producers.

2.3.1 Ability to communicate with other applications

***Requirement**: The management application must allow other applications to communicate with it in a standard way.*

The management application must be able to communicate with other applications using open standards (e.g. Web Services, CORBA, Java RMI) in order to allow the use of the contextual information. As mentioned, there are two kinds of application that interacts with the system: consumers and producers; Figure 1 shows a very general view of those roles.

Figure 1. Very general roles in a context aware architecture

The producers of contextual information must be able to update the information maintained in the system; there are many sensors that can update context information.

> **Scenario 1.Tourist in a city**
>
> Let's imagine a tourist with a mobile phone in a city. The position of the tourist is acquired using the mobile network. Then when the tourist is in a garden and because the mobile phone is equipped with a GPS receiver, the localization was improved by using the GPS. Then the tourist enters a shopping centre, and their location is acquired inside the shopping centre thanks to the mobile phones Wi-Fi capabilities.

This scenario presents three ways to acquire the location of one person with a mobile device. In all the cases there must exist an infrastructure to detect the user and to communicate this information. For instance the shopping centre must have the sensors and Hot-Spots for the wireless network, and the software to process the information in order to detect and provide (communicate) the tourist's location. Due to the wide variety of sensors or context producers it is important that the management application allows communication with these different kinds of contextual providers, and with the applications that use the contextual information, in an appropriate way.

Several ways can be used to communicate context information between the management application and the providers. The most common approach is by defining an API with the possible functions to invoke and use an open standard communication protocol. Irrespective of the technology of deployment, the API must have functions to create, to modify, to query and to update context information. There are examples of communication approaches. For instance in [10], the author propose a network of widgets to acquire and communicate the context, where the communication is done by transmuting XML messages over sockets.

2.3.2 Ability to accept new forms of context

***Requirement**: The management application must allow to add new forms of context information.*

Scenario 1 presents a hypothetical situation of one tourist. In the scenario, it is possible to acquire the location of the tourist by using the network cell, GPS devices and wireless networks. It is possible that the tourist's device does not have those capabilities or that it has other ones such as Bluetooth or tags RFID or the facility to measure tourist's heart

beat and temperature. If a new form of context are necessary for a consumer applications; it is essential to allow these new forms of context to be accepted by the management application. For example, the tourist's temperature must be able to be modelled and used in the management application just as any other new forms of context information.

Allowing new forms of context information ensures the management application is generic, extensible and reusable. If the management application does not have this characteristic, the result will be a specific solution/application for a specific scenario.

The typical example is the application with location as context information, but we can also consider an example where an application requires the user feelings ("state of the spirit") or other kind of contextual information. The management application, must be able to integrate such new contextual information in the system.

To integrate a new contextual dimension in the context model management application is used to define the concrete structure of the new type of context. Some providers of the new contextual dimension must be created and configured to run together the management application in order to feed the model with this data for this type of context.

2.3.3 Ability to add value to the context Information

***Requirement**: It is necessary to add value to the context information*.

Context information like the location of one user in form of coordinates (i.e. long.: -8, lat.: 44) doesn't have much sense in the daily life of people or to some applications. Value can be added to a pair of coordinates by converting those numbers into an understandable format or higher level abstraction such as "you are in Braga", "it is raining there" or "the weather conditions of the place you want to go are bad". GIS's are able to convert geographical information (e.g. symbolic to coordinates and vice versa); however context is more than geographical information.

In addition to allow the context model to be queried the management system must provide such an added value to the information. "Added value" can be read as new context information by knowledge inference or new context by transformation. Certainly, the management application can not convert or transform the information received from sensors into contextual information, it is necessary to add other layer to make this conversion. For example, lets imagine a consumer interested in location and who

understands location as city names. Two o three numbers as latitude and longitude do not mean anything to them; by adding meaning as latitude, longitude and altitude is possible to know if the point belongs to the city, or if there are happy people around. The management application must help to do such conversions.

Other important types of added value are context inferring or information inferring. Inferring is natural and intuitive for humans. For example, people know that when a car is moving in a street, normally someone is driving the car. If a program needs to known the location of the driver, it is possible to know this information if the system knows the relations between car and the driver (i.e. if the car is moving, the driver is driving and both are at the same place) and can be infered that the driver has the same location as the car.

2.3.4 Ability to make complexities in context information transparent to users and applications

Requirement*: It is necessary to hide the complexity of the context from applications by supporting them in using context*.

In the tourist scenario, the location of the tourist can be discovered by Bluetooth beacons, Wi-fi hot spots, RFID readers, GPS devices, mobile operator networks and even manually given by the tourist thenselves. Developing a Consumer application that understands all those formats of location context is a real challenge. Those complexities of context must be hidden to the consumer application. The consumer application should not need to know how the management application got the location, this level of transparency must be added by the management application.

2.4 Conclusion

There is no existing software engineering architecture suitable for the design of a CAMGOV application; however some of the key requirements presented above are overlapped with the typical application requirements of scalability, synchrony, extensibility, openness and reusability. Other requirements such as QoS, privacy and security must be provided by the technologies used to develop the CAMGOV. However issues such as privacy are not mentioned at all here because is considered to belong to other area.

Many approaches do not differentiate clearly the way to model, represent and store context information. Modelling context (the creation of the model) is from the world of ideas or a theoretical task, storing the model is a computational task, and the representation is a mixture of the two worlds. Formal definitions (or languages) to store, represent and model context will give a better approach to the problem. Context information can't be manipulated as fully structured data, it is necessary to use a set of methodologies for semi-structured data manipulation.

3 Context Model

3.1 Introduction

In any computational development, it is necessary to model the topic of interest by abstractions [4]. Modelling context or modelling the surrounding environment is a challenging task and the field is still in under continuous development. As mentioned in the survey by Strang et. al. [15], there are diverse proposed models for context for a diverse set of applications. The divergence of the approaches is evident because in the validation or proof of concept it is difficult to see two different models used for the same scenario. Moreover, some models seem to be designed for specific kind of scenarios. Strang [15] states that the ontology based models and object models are good approaches to modelling context. In [11, 22] Chen also proposes the use of ontologies to model context. SOUPA is a Standard Ontology for Ubiquitous and Pervasive Applications[11]. SOUPA is composed by a set of ontologies including ontologies for persons, time and space. Gu et. al. [23] present another approach (CONON) that uses ontologies. CONON is intended for use in Smart Spaces where ontologies are used to model objects and rules.

Usually, the first context dimension that comes to mind (and often used in applications) is location and in a CAMGOV location is certainly an important dimension. Location is one of the context dimensions that allows to determine the context of the entities when combined with other information. For instance if the location of one person is the “Shopping centre”, it is possible to assume the person is there to buy something if the profession of the person is known (i.e. they are not a seller). Also it is possible to know the surrounding environment (e.g. nearby objects such as stores, people, facilities, etc).

The initial attempt in the development of a CAMGOV in the USEME.GOV project was to adapt previous research results. Since location was perceived to be the main context dimension, the initial literature review was targeted on location representation in aim to reuse and create a bridge (a common understanding) between GIS (Geographical Information Systems) and context-aware applications. There are a lot of models for location representation from a basic GML (Geographical Markup Language) [24] to an adequate representation of geographical locations by special data types (ADT Abstract Data Types) developed by the OpenLS initiative [24]. Those are very useful to communicate and represent information extracted from a GIS server.

Some locations can be easily expressed with GML, for instance a point on the earth can be defined with two coordinates, altitude and the coordinate system used. Imagine a citizen with a GPS device, it is quite simple to know the point where the citizen is. Now imagine a tourist attraction such as “The Alhambra castle”. It is a complex location but can be represented using ADT coordinates; there are ways to know the spatial relation between the citizen and The Alhambra using a GIS. For instance, the spatial relation “the citizen is near by The Alhambra” (represented by distance between the user and the castle) can be obtained from a GIS. In some places inside the castle it is possible to know the citizen “is in” the Alhambra (i.e. by using a GPS device). However, frequent locations such as “I’m in my car” or “I’m in a bus” are quite difficult to represent using ADTs. GIS and ADTs are not enough to represent this kind of context location information but are useful to represent traditional location information. There are examples of architectures for mobile applications using commercial products where the location is almost the only context-dimension [25]. However, for a CAMGOV location is not the only important context dimension, and for this reason, a higher and richer level of abstraction to express context dimensions and context is necessary.

In addition to the user location and location of real word objects, it is important to be able to represent virtual entities such as software in the form of services. We can consider there are three kinds of software components in context aware systems: the core software of the context aware system, the software that captures context information (e.g. context providers) and the software that will be discovered by using context information (e.g. user services). The last two kinds of services must be modelled. For example consider a service for happy people that should only be presented to users when they are happy. The model must be able to model both the services that ascertain the mood of the user and also the services that can be used by those happy people. There are technologies that enable the description and invocation of services. WSDL[26], UDDI [27] and OWL-S [28] are technologies that can be used to describe what web services can provide and how to use them. Furthermore, UDDI and OWL-S enable the description of the semantics of the services.

3.2 The selected model

The necessity expressed above and in the requirements chapter is supplied by the object model introduced in [4]. The Object model allows the representation of the

surrounding environment and the relationships of this environment with services and applications.

The main element of the object model [4] is the base object, whit most of the other entities represented using this construct, even relationships and properties. The Figure 2 presents the four big roles of this model. On top, the Base Object can be used as an anonymous object. In spite of the Figure 2 shown a typical class diagram, there is not a strong concept of classes in the model due the multiple role objects can play.

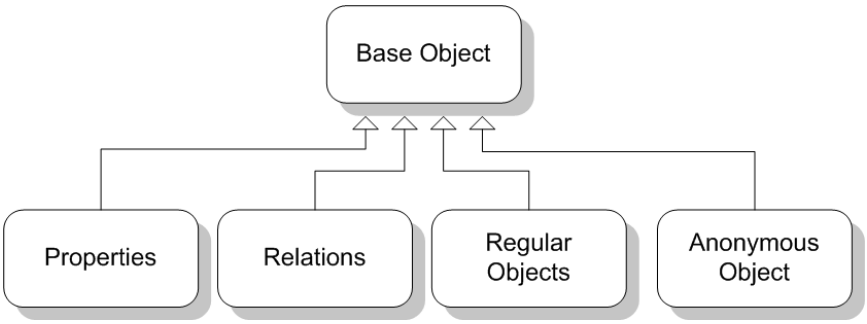

Figure 2. The basic objects

The location for material objects can be defined as container-contained relationship where the existence of the object implies the existence of the container; for instance, users can be in containers such as homes, houses, cars, offices or an unknown container. The anonymous objects help to model those unknown containers and many other anonymous objects without defining a class (see Figure 3).

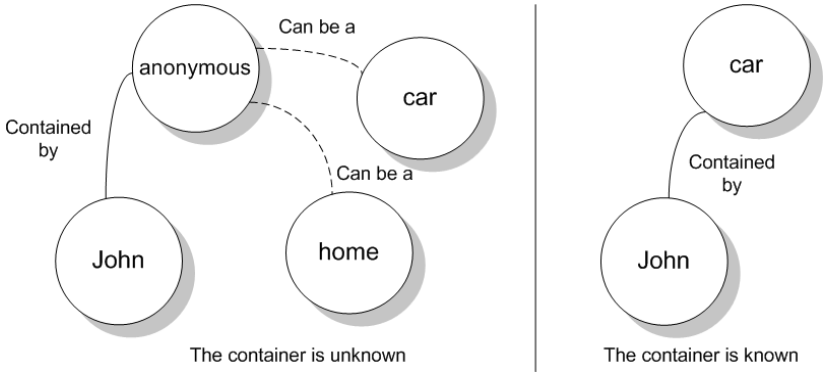

Figure 3. Example of anonymous objects

One of the advantages of the anonymous objects is that those objects can be manipulated without knowing their nature. For instance consider some application that needs to know if John is close to one specific traffic light. Supposing John is with his PDA, and there exists a relationship between the PDA and John; and the PDA can run

an application that communicates the location of the PDA, then it is possible to create an algorithm that infers the containment relationship of John with his container by using the is-with relationship between John and the PDA. If the container of John is the same as the container of the traffic light, then further information of this container is not needed and it remains as anonymous object.

The context of one object (i.e. a person) can be seen as the relationship between this object and the other objects, including the properties of the object. As part of the object model there is computational support such as algorithms for querying the information.

The last part of this chapter describes how the context is represented by the object model using objects and relationships. At the end of the chapter is presented how the model is stored.

3.3 Using the Object Model

The model for context in a context-aware application must be able to describe the target "surrounding environment" that the application needs. In order to exemplify how to identify the relevant elements in the "surrounding environment" we present here two scenarios featuring a citizen in a city. Those scenarios are intended to show how an application to submit anomalies to the government authorities will work:

> **Scenario** 1. *"John was driving down the street and suddenly sees a malfunctioning traffic light and he wants to inform to the public authority about the situation. He stops the car and with his PDA (Bluetooth capable) opens an application that helps him to inform to the public authority about the situation. Because he is currently in his car, and his car has a GPS device with Bluetooth interface, the application passes the information about his location."*
>
> **Scenario** 2. *"John was driving down the street and suddenly sess a malfunctioning traffic light, that he wants to inform to the public authority about. He activates by vocal commands his PDA (Bluetooth capable) opening an application that helps him to inform to the public authority about the situation. Because he is currently in his car, and his car has a GPS device with Bluetooth interface, the application uses the information about his location."*

This scenario describes a situation where it is easy to identify physical objects. The first objects to identify are the nouns: John, street, semaphore, car, PDA, GPS. It is possible also to identify the relationships between those objects such as:

- *John has a PDA*
- *John drives the car*
- *The car has a GPS*
- *The car is in the street*
- *The semaphore is in the street*

By comparing the two scenarios, other kinds of information can be supposed such as “the application works when the car is stopped in case the user has to manipulate the application manually”. Otherwise if the application accepts vocal commands “he can drive the car as well using the application”. Some objects are important only in some situations, for instance the application (with manual input) used by the driver to report the situation does not have sense when the car is moving. These kind of rules should be verified in the model by the context aware applications. The properties of the objects (e.g. the car’s speed, voice commands capabilities) can be used for this purpose.

There is information that humans are aware of but that is not formally expressed. In the scenario the GPS, the car, the PDA and John are in the same place, this kind of information can be inferred by programs if adequate information is represented in the model. It is possible to consider if John is-with the PDA, and create a program that infers both are in the same place.

Finally it is possible to infer other information such as, if John is driving, that the car has some speed. The speed of the car can be considered as a property of the car and is used to validate if an application can be launched.

3.3.1 Modeling Objects

An important question is what is an object? For instance, in the Solar System it is important to model planets as objects, asteroids, stars, satellites, comets and the gravitational relations between each planet or celestial object. Also, in an chemical reaction it can be interesting to model molecules, the atoms, and the relationships at an “atomically scale”, including the behaviour of electrons. In the two examples the size level are totally different and probably electrons in the chemical model are objects, but in

the planetary model atoms do not exists as objects or rather are not interesting as objects, even though planets are composed by atoms. One of the advantages of the model is that anything can be considered an object, and defining “what is an object” depends on the desired granularity, relevance and the designer’s-point-of-view.

In a normal ER representation the user location can be a property, such as the “user house address”. Then it is possible to have a entity “user” with the fields “address” and a relation with an entity “city”. Those entities will be reflected in a physical database model as two tables, one for users and another with cities. Both tables must have a fixed number of columns. A simple change in the location of the user can be untraceable in a static model; for instance the user location can be their home and moments after the location can be their car. In a static model such as an ER model, could be difficult to change the field “address” for a field that represents a “car”. In an object model the change can be easily made in the containment relation of the user. Another aspect is the dynamicity of the properties. In a static model there are a fixed number of properties. For instance the “house address” in a static model can be a string, but a house has a more deep meaning that a string. House by itself can be a complex object, with rooms, colours, floors, spaces, sensors, etc. Some entities that in other models can be seen as properties, in the object model can be viewed as objects or relations. This flexibility of the model is clearly one of its advantages.

3.3.2 Modelling Relationship Objects

Relation is another important concept for modelling context; basically the “surrounding environment” can be described by the objects and relations between objects. To test the model we can consider the following scenario:

> **Scenario** 3. *In the city of Pamplona there is a square, in the square there is a corner, in the corner there is a house, in the house there is a room, and in the room there is a bird.*

Relationships can be more than a link between two objects. Scenario 3 has several objects and relationships between those objects. For example regarding the relation between Pamplona and the square, it is possible to affirm “The Square is in Pamplona” or “Inside Pamplona is a square”. In such cases the relationship has an “inverse” relationship, as is the case with Is-In which has an inverse of “contained-by”. The object model allows to “attach” properties to the relationships, and regards the relationships are objects.

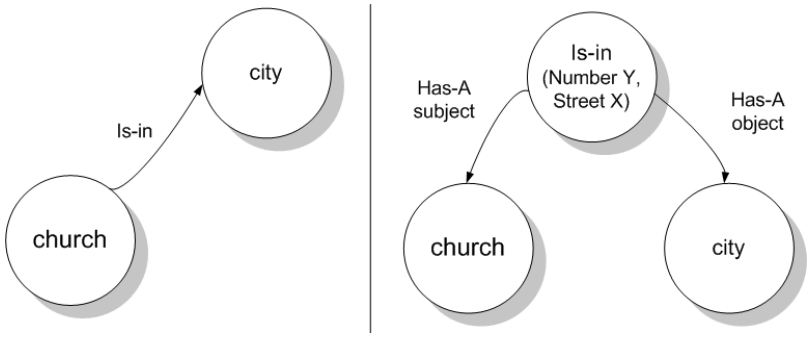

Figure 4. Simple and complex Relationships

As well as properties like inverse and refection, relationships can have more information. The Figure 4 shows the containment relation between a church and a city. More than a simply relationship of containment; it is possible to add geographical information such as street name and number, or GPS coordinates, to the relationship. In this case the relationship will be not a simple link; it will be a complex object with information inside. In order to describe the containment relationship of a region (such as a mobile cell coverage area) or a city, the information in the relationship can be a special data type describing the boundaries of such an area.

The Selection Algorithms and the representation (using RDF [29]) are presented in the following, and provide the computational support required. Both elements are an integral part of the approach proposed in [4].

3.3.3 Selection algorithms

A query mechanism must be created in order to extract the information about objects and relationships. By examining relationships it is possible to get partial information about context. Let's imagine a tourist searching for restaurants: the location information of the tourist can be used to search for which restaurants are nearby. But for this task it is necessary to create a set of algorithms to find this kind of information (nearby things). In the case of the restaurant, the algorithm will look for near-by objects (restaurants) to the tourist. Previously will be necessary a definition of what is near-by (e.g. 600 meters).

Other kinds of algorithm can be defined depending on the needs of the context aware application. For example, an Is-In algorithm can be useful to find out if object **A** is in object **B.** Derived from the is-in algorithm, further algorithms can be created to explore containment relationships. For instance, to find which objects are contained in another object, which is the object that contain the current object, and, which are the objects in the same container. The last algorithm (which are the objects in the same container), can

in some way be used to resolve the restaurant scenario, to find which are the restaurants that are nearby the location of the tourist.

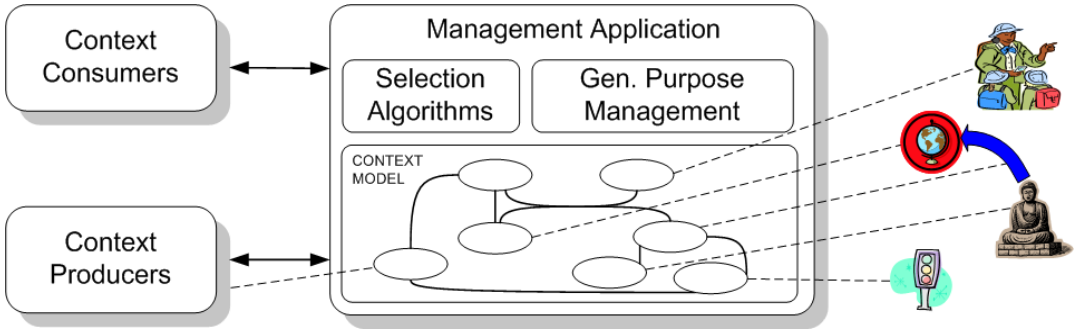

Figure 5. General view of the of the context aware architecture

The algorithms for querying the model looking for information inside derived from the relations are called Selection Algorithms. Selection Algorithms are an integral part of the Management Application (Figure 5) that is necessary to use the context object model. The consumers must know which selection algorithm use. Also it is possible for each consumer to create its own algorithm. More details about Selection Algorithm can be found in the section 4.2.2.

Complementing the figure presented in the chapter 2; the Figure 5 adds new elements: the context model, the selection algorithms and the general purpose management application (that includes the API). The context model is the data structures that contain all the modelled objects. The general purpose management application provides functionalities for creation, updating and querying the context model.

3.3.4 Context model representation

Once the theoretical model is designed, it is necessary to represent that model by using computational artefacts. There are several ways to represent objects, and the typical case is to use a database to store objects properties. Nevertheless in new Java application development is quite common to find applications using an ORM (Object Relational Mapper) to give persistency to the objects (instances of the classes). For example in an application with the users, addresses and cities, the Hibernate [30] method makes transparent the use of the database when classes are instantiated (by embedding automatically some code in the classes).

The representation for context is not simple as than in traditional applications; there are no commercial context databases. Context representation normally is not approached as an isolated issue in context aware applications. It is very common to see approaches including complex architectures or frameworks for context management alongside with client applications and services. For instance in COoL [21] is used an ontology for service modelling, where the representation is made by OWL documents but those documents were converted in F-Logic [31] in order to improve reasoning process. Similarly the GAIA project [32] uses XML in a DALM format to store contextual information, and CORBA for communications.

The object model [4] is based mainly upon objects and relationships. For instance John is-in the car, or John is-with their mobile. These examples are clearly statements and statements have three parts: object, subject and predicate. There are notations for statements, such as functional notation "*Is-In (John, car)"* or triplets notation *"("John", "car", "is-in")".* For this kind of structured information it is necessary a flexible technology with an appropriate computational support. The choice was the Resource Description Framework (RDF). Later in this chapter is present why RDF was selected.

The Resource Description Framework (RDF) [29] is a general-purpose language for representing information in the Web. RDF has three basic elements subject, predicate and object. RDF was created to describe web resources, and then was extended to express relationships between entities. RDF suits very well for the object context model because it allows to create relations between objects. The basic structure in RDF is the triplet and can be represented as a graphs as showed in Figure 6.

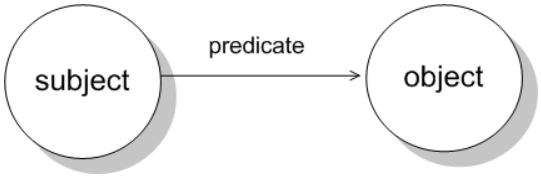

Figure 6. RDF Model

In Figure 6 the subject and the object are represented by a node and the predicate is represented by an arrow. Let us consider the following statement:

http://www.example.org/index.html has a **creator** whose value is **John Smith**.

This statement decomposed into the three elements:

- the *subject* is the URL `http://www.example.org/index.html`
- the *predicate* is the word "creator"
- the *object* is the phrase "John Smith"

Notice the subject is a URL; as stated before, RDF is used to describe web resources. The first adaption to be made in the context model is to create a URL for each object to be represented in the model. In the sentence “John is-in the car”, the URL representing John can be something like:

`http://purl.oclc.org/NET/contextobjects#Jonh`

This mapping from real objects to a URL must be done by the application that creates the object John. The following fragment of RDF/XML can be used to represent the expression John is in the car.

```
<rdf:Description rdf:about="http://purl.oclc.org/NET/contextobjects#Jonh">
       <use:is_in>car</use:is_in>
</rdf:Description>
```

The graph representation is as follows:

Figure 7. Simple RDF Graph

However the only object is John, the other objects (i.e. the *Is-In* relationship and the car) in example above are not represented in the context model, in spite of the statements in RDF to represent relations. Relationships in the context model are richer objects and must be represented as objects in a richer way. Additionally, the car in the example is only represented as a *literal* (value).

The form to represent a relationship in the context model is more complex. Continuing with the example that ”John is in the car”, consider the following fragment of RDF/XML:

```
<rdf:Description rdf:about="http://usemegov.org/contextaggr/isin322">
   <use:subject rdf:resource="http://purl.org/NET/contextobjects/John"/>
   <use:object rdf:resource="http://purl.org/NET/contextobjects/Car"/>
   <use:type
rdf:resource="http://purl.org/NET/contextobjects/IsInRelationship"/>
</rdf:Description>
```

And the corresponding graph:

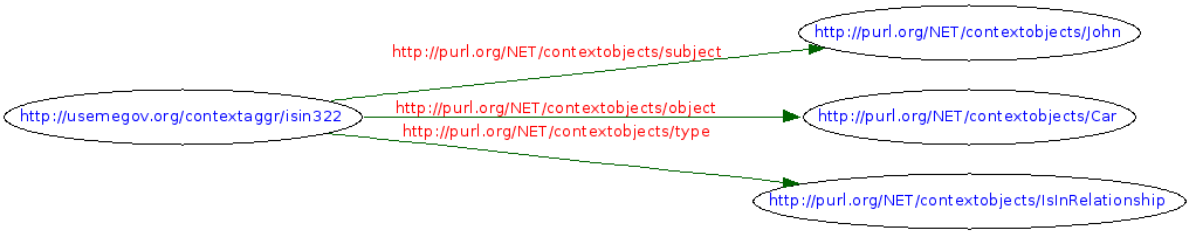

Figure 8. RDF graph representing a Relationship.

The presented RDF above show how is represented a Relationship in the model, it has two properties, the *subject* and the *object,* that have the same meaning shown in the elements of the Figure 6. Notice that the relationship can have more information than the subject and the object. For instance if it is a containment relationship it could also define an address in a city where one of the properties can be the street and house number.

Using RDF it is possible to represent other kind of objects such as services. Properties of the services, such as access point or kind of service, can also, be represented. A possible example of a complaint service can be the following:

```
<rdf:Description rdf:about="http://usemegov.org/contextaggr/objects/42">
        <use:type rdf:resource="#reportComplaint"/>
        <use:location rdf:resource="http://193.137.8.61:8080/usemegov/services/complaintresiduos"/>
        <use:point
rdf:resource="http://193.137.8.61:8081/complaintresiduos/services/ComplaintServiceAVService"/>
        <use:name>Servi&amp;amp;co de recolha de residuos solidos</use:name>
        <use:id>42</use:id>
        <use:serves rdf:resource="#Papeleira"/>
        <use:serves rdf:resource="#Contentor"/>
</rdf:Description>
```

Notice that RDF not intended to be human readable. In this example some relationships were simplified by XML namespaces. For instance the use:serves element in the above excerpt is presented as a property. This element represents a relationship between the service and the objects that is possible to complain about. Other relationships are also simplified. The model allows to create properties and relationships for all the objects in a way specific for each scenario of use, where the properties and relationships may vary according with the specific requirements of the scenario. It is also possible to create a base model for each scenario. The base model contains a set of core objects that do not change over a long period of time. For example, a bridge, a monument, a service and the relationships among those objects.

3.4 Conclusions

The Object Model provides a model for context information. As showed it is possible to apply the model to different scenarios and providing the computational support for the model is a feasible task. When realized it will bring the functionalities required of a CAMGOV.

The selection of RDF was based on the potential of utilizing tools as Jena and Protégé[33], and can leverage the momentum of the Semantic Web movement that is supported mainly by RDF. Another concrete reason to select RDF was the computational support. In order to use the representation it is necessary to have a set of tools to query, create, update and delete RDF. There are tools ready to work with RDF such as Jena and Protégé [33]. Jena is set of Java classes that allow manipulating RDF at a higher level. Jena also includes a rule-based inference engine [34]. Protégé is a very versatile tool and helps to model knowledge. Since RDF is not intended to be human readable; the models for specific scenarios can be initially created in Protégé for further conversion to RDF.

One of the important advantages of RDF is that can be queried by using its own query language. The RDF query language [35] (SPARQL) allows the creation of query sentences very similar to SQL. Jena supports SPARQL that, combined with the rule-based inference engine creates a powerful environment to develop applications over RDF.

4 Implementation

This chapter presents the design and implementation of the context model and the proposed supporting architecture for object model. First the overall architecture is presented, followed by each component in more detail.

The implementation was made using Java, and as the application container was used Apache Tomcat. The selection of Java and Tomcat was made due to the intended web support of the context aware applications.

4.1 Context management architecture overview

The architecture (see Figure 9) includes the clients of the context information (Context Aware applications), the producers of the context information (Feeders), the managers of the context aware applications (Management application) that make transparent the complexities of the Selection Algorithms and the Context Management to the applications. Additionally, external services are showed in case the context aware application redirect the users to the services they need.

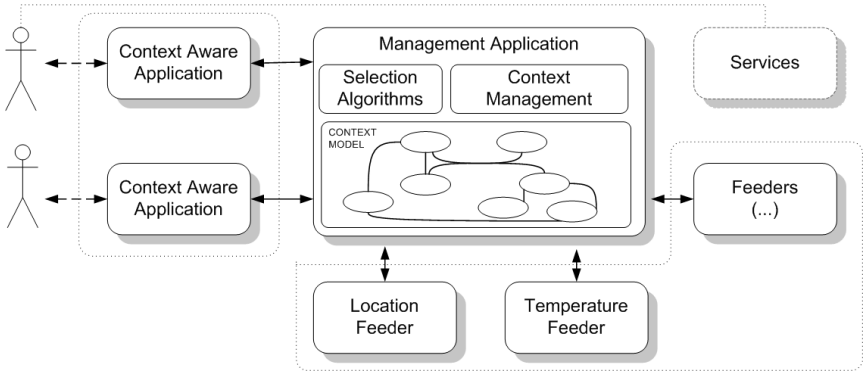

Figure 9. Context Management Architecture

Below is a brief introduction to the main components of the architecture depicted in Figure 9.

- **Context Model:** The context model holds the objects of interest to its client application/services. All the objects are represented in RDF. There are several kinds of represented objects such as real objects (e.g. users, streets, cities, etc) and the objects representing relationships between real objects (e.g. is-in relationships, is-with relationships). Virtual objects such as context feeders (i.e. location, time, temperature, humor, etc) and user services are also represented.

- **Selection Algorithms**: The selection algorithms are invoked by the context management on behalf of the context aware applications. The selection algorithms are plug-in methods that query the model. An example of a selection algorithm is the Is-In algorithm. The Is-In algorithm searches the is-in relationship

between objects and returns the selected criteria. For example which object is-in another object, or which objects are contained in object X. Other algorithms can be developed, plugged-in and invoked by the applications.

- **Feeders:** Feeders provide contextual information. Feeders can be seen as providers of information about one dimension. For example, a location feeder will provide the location of objects. A mobile operator can provide information about the cell where the mobile phone is. The feeder will be the service that provides the location of the phone. Other example can be a GPS feeder that sends the sensed location to the model. Feeders must be able to push information to the context model and to be polled, and must expose an API for information interchange.

- **Context Management:** This component exposes the API for manipulation of the context model. It has functions for contextual information creation and modification, as well as pulling information from the feeders and for algorithm execution. As the context model is based in RDF, Jena[34] provides an upper layer to manipulate the query. Jena allows queries to the model using SPARQL[35], which is intended to query RDF documents.

- **Context Aware Applications:** The context aware application uses the context for a purpose, often this purpose is to provide the right information in the right time and place. Information is provided by services, so in some way the context aware applications must attempt to matchmake the users needs and situation with services that supply those needs to the user; this matchmaking process utilizing contextual information.

Besides the elements showed in the Figure 9 there are other elements not visible, such as how the context aware application interacts with the management application. Those interactions were named Application Interaction Models. One Application Interaction Model will be presented in depth in the deployment chapter.

- **Application Interaction model:** This is the form in which the application interacts with the management application and with the users. Applications can behave in different ways; for instance the typical Location Based Services (LBS) scenario is a tourist that wants to find restaurants around him or her and opens their application to receive the list of restaurants. The interaction of this kind of application could be: 1) ascertain for the tourist location, 2) query the restaurants around that location. 3) Return the list of restaurant locations. This is just one possible interaction model. Such interaction models must be specified using the API of the management application.

The Deployment chapter presents one Context Aware Context Application and explains its Interaction model.

4.2 Management Application and Context Model Implementation

The Context Management Application is strongly tied to the Context Model. This section presents the implementation of both components in order to understand better how they are interrelated and how they work together.

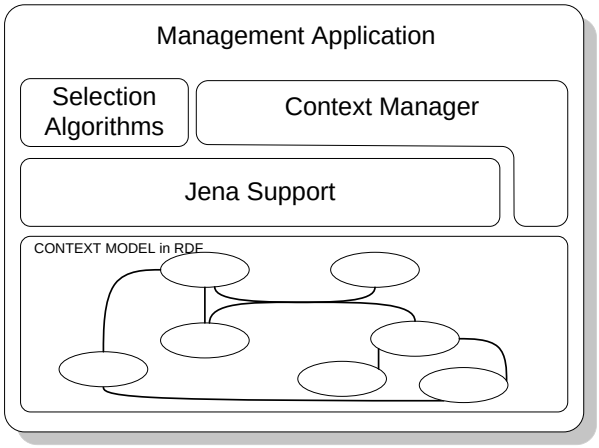

Figure 10. The Management Application overview

The object context model is represented using RDF. The natural way to store RDF information is in XML documents in files. For this reason it is necessary to use an intermediate layer to manage those documents (see Figure 10). The Jena framework provides a set of classes to manipulate RDF documents, which we employ as the intermediate layer between the RDF documents and the Management Application. All the management application was developed using Java, which is another reason for the selection of Jena (Jena was developed using Java).

The management implementation must provide functionalities for manipulation of objects. The Jena framework has the `Model` class `(com.hp.hpl.jena.rdf.moldel.Model)` for RDF manipulation. The Jena `Model` class encapsulates RDF models, where the RDF model is a set of Statements. The `Model` class provides methods for creating resources, properties, and literals, and the Statements which link them together. It also provides methods for adding statements, removing them from, a model, for querying a model, and finally, it provides a of set operations for combining models.

4.2.1 Representing Objects[1]

There are three kinds of objects: regular objects, relationships and properties. For example, an object representing a person can be seen in the RDF extract below.

```
<rdf:Description rdf:about="http://useme.dsi.uminho.pt/instances/1">
       <use:id>1</use:id>
       <use:type rdf:resource="http://purl.org/NET/coreObjects/Person"/>
       <use:name rdf:resource="http://purl.org/NET/instances/7"/>
       <use:age rdf:resource="http://purl.org/NET/instances/8"/>
       <use:dateOfBirth rdf:resource="http://purl.org/NET/3"/>
</rdf:Description>
```

Note that the object has an identifier and type attribute. The identifier attribute is mandatory and is used as a unique reference to this object within the model (the same number can be seen in the URI value of the object's `rdf:Description` tag; in the first line of the extract above).

The `type` attribute (an XML attribute) is optional, but if present it is used to specify the type of the object. Note that the type is just a value to indicate the nature of the object but not necessarily its structure. We may expect certain types of objects to have certain attributes, but these must be understood as typical attributes and not mandatory ones. Even if two objects of the same type contain completely different attributes it is still useful to know their type for reasoning purposes.

Further attributes of an object take the form of RDF resources that are fully described elsewhere, such as the name, age and date of birth of the person. This is conceptually equivalent to a Has-A relationship to another object. Following a string of Has-A relationships we will often (though not always) arrive at an object that represents a basic type such as a Number or String, in which a literal value can be described via the `value` attribute. The following fragment of code represents the age of John.

```
<rdf:Description rdf:about="http://purl.org/NET/objects/7343432">
   <use:age rdf:resource="http://purl.org/NET/objects/55324234"/>
</rdf:Description>

<rdf:Description rdf:about="http://purl.org/NET/objects/55324234">
   <use:value rdf:datatype="&xsd;int" >8</use:value>
</rdf:Description>
```

The Has-A relationship with literals is implemented by adding a `value` element to the RDF representation of the object.

[1] The words attribute and property have the same meaning in this text.

The Is-In relationship is a typical containment relationship. The relationship means that one object is inside another object. We can also read the relationship in the other direction, i.e. that one object contains another. In other words the inverse of the Is-In relationship is the Contais relationship. The RDF extract for the Is-In relationship object is shown below.

```
<rdf:Description rdf:about="http://purl.org/NET/objects/34288427">
   <use:subject rdf:resource="http://purl.org/NET/objects/249547"/>
   <use:object rdf:resource="http://purl.org/NET/objects/51546912"/>
   <use:type rdf:resource="#IsInRelationship"/>
</rdf:Description>
```

Note how relationships are represented just like any other kind of object, with an ID and a type. Here the `IsInRelationship` simply contains two attributes that link to the object and subject of the relationship, but further attributes could be added to specify the nature of the relationship in more detail (e.g. the number of the room or floor that specified where exactly in the library the person is).

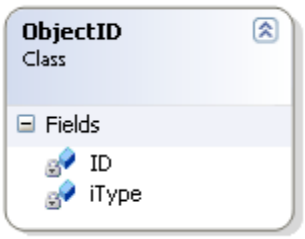

Figure 11. The ObjectID Type

In Java we implemented a class to represent the object. This class must contain the ID (identification) of the object which is specified by the URL of the `about` attribute. Also the type of the object must be included in the Java class. As each object has an indeterminate number and variety of attributes, the object only specifies the type and the ID.

There is another Java class for representing objects to external clients: the `ContextModelObject` class. This class enables querying the attributes on object. The attributes correspond themselves are represented by Has-A relationships. There are two forms of attributes literals, and objects. Literals have a direct value; for instance the age Has-A numeric value (e.g. 33). Object attributes represents complex properties. For instance, in the relationship John Has-A house, the house is a complex object identified by URL.

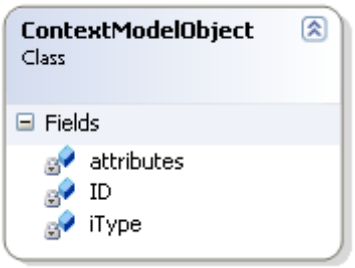

Figure 12. The `ContexModelObject` class

The attributes are stored in the field `attributes` of the ContentextModelObject class. The field `attributes` is a key-value pair structure. The key corresponds to the name of the attribute and the value to the attribute value when it is a literal attribute, or the URL of another object when the attribute is a complex property.

4.2.2 Selection Algorithms

The Selection Algorithms allow querying the model by using the relationships between objects. An example of a query is: which objects are contained in a city? The selection algorithm will return the objects directly within the object and also those that father nested within those. For instance, it may return a garden and the objects inside the garden, such as fountains, flower beds, trash cans, benches, etc. Selection algorithms were intended to be pluggable pieces of software, and for this reason can be developed and customized depending the needing of the applications. The applications must also know the name of the selection algorithm to invoke because the form those algorithms are invoked is by its name.

Selection algorithms mainly use the relationship information in the model for object retrieval. There are many similarities in selection algorithms because the model only has binary relationships. There are two objects involved in a relationship, e.g., the objects “John” and the “church”, and the relationship “Is-in”. Relationships also can be reflexive relationships or irreflexive relationships. The is-in relationship is irreflexive; it is not possible to say John is-in a church and at the same time “The church is-in John”. The is-with relationship is reflexive, it is possible to say John is-with the PDA and vice versa. However, the is-in relationship has an inverse relationship: the “contains” relationship. The binary nature of the relationships and the characteristics mentioned above influence the way the selection algorithms are invoked.

Relationships has three parts: subject, object and predicate. In the following statement “John is-with the PDA” the predicate is the relationship (is-with), the subject is John and the object is the PDA. For the binary nature of the relationships and its characteristics there are few ways to invoke the selection algorithms by varying the parameters to be passed.

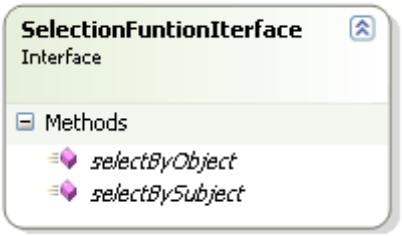

Figure 13. Selection function interface

Selection algorithms must implement the SelectionFunctionInterface. The methods to implement are shown in the Table 1.

Table 1. Selection functions interface description

Func.	Interface	Description
8	ContextModelObject[] **runSelectByObject**(ObjectID objecteid, String ObjectTypeName)	Retrieve objects of the `ObjectTypeName` type. In case of is-in algorithm will search the objects that contain the `objecteid` (provided by Selection Functions classes). The object type is optional.
9	ContextModelObject[] **runselectBySubjects**(ObjectID objecteid, String ObjectTypeName)	Retrieve objects of the `ObjectTypeName` type. In case of is-in algorithm will search all objects contained by the same container as the `objecteid` The object type to retrieve is optional.

4.2.3 The Context Manager

The Context Manager is a component that encapsulates the Context Model. The Context Manager provides the majority of the functionalities to use the Context Model by exposing functions for object creation, object querying, Selection Algorithms execution and context feeding. Figure 14 shows the interface of the Context Manager.

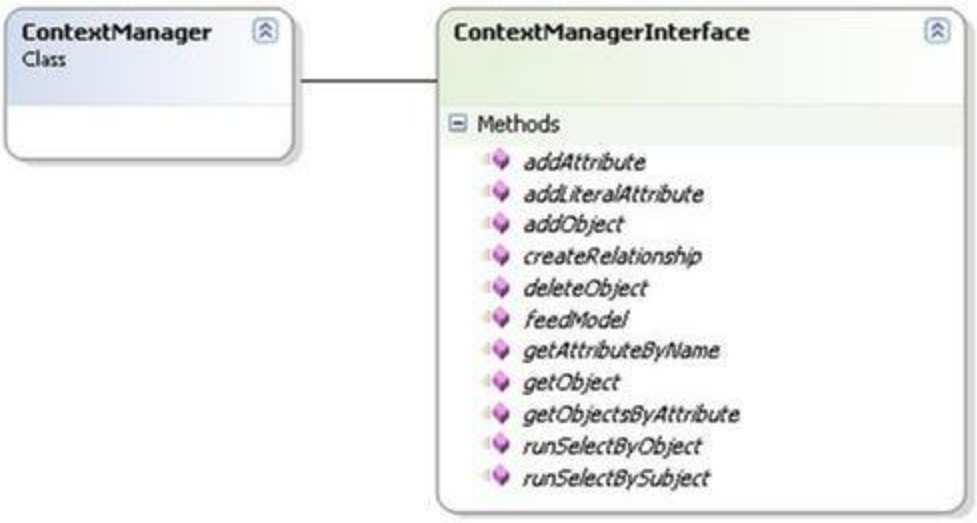

Figure 14. The Context Manager exposed functionalities (public methods)

The Context Manager was developed as a Java Class and can be accessed by using web services or by directly invocation of the class. The functionalities exposed to other applications are presented below.

4.2.3.1 Functionalities related with object creation

Basic functionalities must allow the creation and manipulation of objects. As shown above in "Representing Objects" there are objects with properties, and relationships-objects with properties. When a relationship is created, is very useful to pass the type of the relationship and the two related objects.

In order to facilitate the creation of objects, relationships and their properties, the Context Manager must expose functions for this purpose. The Table 2 shows the API that the Management Application must expose for object creation related functionalities.

Table 2. Object Creation related API

Func.	Interface	Description
1	ObjectID **addObject** (ObjectType object, String objectName) public ObjectID **addObject** (ObjectID baseObject, String objectName) ObjectID **addObject** (ContextModelObject object, String objectName);	Allows the client to add objects in the form of a blank shell or copy the type of one object by passing an `ObjectID`. The identity of the object within the model is returned. Note that the `objectName` parameter allows a description of the object to be given, but it is not required (an empty string can be passed) and it is not necessarily unique.
2	ObjectID **createRelationship** (ObjectID relationshipTemplateObjectID, ObjectID objectObjectID, ObjectID subjectObjectID)	Creates a relationship object between two objects in the model.

3	Boolean **addAttribute** (ObjectID objecteid, String attributeName, ObjectType attribute) Boolean **addAttribute** (ObjectID objecteid, String attributeName, ObjectID attributeObjectID)	Allow the client to add an attribute to an object by either (a) passing a whole object or (b) passing the ID of an object that is already within the model. Note that if an attribute with the same name already exists the function will fail (an attribute name is unique within the scope of the individual object).
4	Boolean **addLiteralAttribute** (ObjectID objectID, String attributeName, String attribute);	Allow the client to add a literal attribute to an object.
5	Boolean **deleteObject** (ObjectID object)	Delete the object passed as parameter.

4.2.3.2 Object querying

The basic querying of the Context manager component is supported by two functions.

Func.	Interface	Description
6	ObjectID[] **getObjectsByAttribute** (String attributeName, ObjectType attribute)	Search the model for objects that have an attribute of a given name that is equal is to the object specified by the client
7	ObjectID **getObject** (ObjectID objecteid)	Allow the client to check if an object exists and/or to retrieve it given its unique ID within the model.
8	ObjectID **getAttributeByName**(ObjectID objectID, String attributeName)	Allow to query for properties of objects. Returns the ObjectID of the property that have the name attributeName.

4.2.4 Invoking Selection Algorithms

Two generic forms to invoke selection algorithms were implemented and also two generic selection algorithms. These characteristics of the algorithms were derived from the binary nature of the relationships, and for the reflexive or irreflexive property of the relationship. The functions for invoking selection algorithms are presented in the Table 3.

Table 3. Algorithm invocation related API

Func.	Interface	Description
8	ContextModelObject[] **runSelectByObject**(ObjectID objecteid, String algorithm, [String ObjectTypeName])	Search with the desired algorithm for objects. In case of the is-in algorithm it will search for objects that contain the `objecteid` as the object of an is-in relationship. The object type is optional, but specified will filter by object type.
9	ContextModelObject[] **runselectBySubjects**(ObjectID objecteid, String algorithm, [String ObjectTypeName])	Search with the desired algorithm for objects. In case of the is-in algorithm it

		will search all objects contained by the same container as the `objecteid`. The object type is optional, but specified will filter by object type.

4.2.4.1 Feeders Invocation

To be useful the model must be kept up-to-date. One of the methods to kept the model updated is by requesting information (polling) from the context sources (such as sensors) when it is required. For example, asking for the location of mobile user, in a cell network, only when the system needs that location.

The software components responsible for providing context information are the feeders (see Figure 9). Feeders such as a Temperature feeder or Location feeder must be implemented and represented in the object model. The Context Manager will invoke those feeders (on behalf of the applications) in order to request context information. The Context Manager internally will find an appropriate feeder depending on the information the application needs. Applications are able to ask the Context Manager to invoke the feeders by the interface described in Table 4. The implementation of feeders will be explained later in this chapter.

Table 4. Feeding requets invocation

Func.	Interface	Description
5	ObjectID **feedModel**(ObjectID targetObject, ObjectID attribueType, String attributeName)	The Context manager will use the internal findFeeders function to find all the feeders that are able to supply such an `attributeType`. The resulting set of candidate feeders will then be filtered on the basis of their needs being met from the attributes of the target object. If more than one candidate feeder remains after this filtering process then the feeder manager will simply select the first one. However, in future versions of the model it will perform a more intelligent selection of a feeder based on qualities such as the feeder's accuracy, cost-of-use, etc.

4.3 Feeders

Feeders can be seen as pieces of software that provide context information. Normally feeders are associated with sensors such as thermometers, GPS devices, decibel

meters, anemometers etc. A feeder can push context information into the model or it can be polled by the Context Manager. The information can be reflected in the model as a relationship or as a property of one object. For example, a feeder could provide information on the location of Mobile phone Operator (MO) subscribers, and, when required, the feeder could add/update information about devices (mobile phones) and their locations. In this particular example, the feeder would update the relationship of the mobile with the cell where it is detected; in principle it will update the is-in relationship. Any feeder can update the model by using the general purposed interface provided for that (the Context Manager Interface).

The feeders must implement a web service interface to be polled. The interface is shown in the following table (Table 5).

Table 5. Feeders Interface

Func.	Interface	Description
1	ObjectID **feedModel**(ObjectID objecteid, AttributeName attributeName, ServiceURL ContextManagerURL)	When the Context Manager is requested to feed the object model it passes the request to the feeder. The feeder must provide information about the object identified in the model with the value of the parameter `objecteid`, the `attributeName` corresponding to the attribute to be feeded (`attributeName` is optional). This information must be provided to the service pointed by the `ContextManagerURL` parameter.

Feeders must know how to update the information in the model. Feeders can also be used as wrappers of existing applications such as GIS, or services that provide weather conditions. Moreover feeders can provide context information from several sources, for instance, it is possible to create a feeder that using information given by a GPS device, transforms the coordinate information into symbolic information by using a GIS.

5 Deployment and Validation

5.1 Introduction

One of the typical utilization scenarios of context information is to discover services using the user context. In order to test the proposed Context Model and architecture a context-aware complaint application was developed for use from a citizen's smart phone. In order to clarify the deployment of the Context Model Architecture the following scenario is presented to identify the pre-requisites for set up and use.

> **Scenario** 4. *"John was walking in a garden and sees a damaged fountain. He wants to inform the public authority about the situation. So, on his smartphone he opens an application that helps him to report the situation to the public authority."*

The application uses the Context Manager functionalities to help citizens find complaint services using their current context. But, firstly the context model must be populated with objects representing the citizen, their smart phone, the relationships between them, and feeders set-up to automatically update the model (e.g. to ascertain the operator cell location of the smart phone). After this initial setup procedure the application is able to query the Object Manager to, for example, ask about objects related to the user in the real world and to ask about "unreal" objects such as services. These queries are used to perform an automatic identification, or honing down, of objects that the user may like to complain about in their surrounding environment, followed by the identification and connection to the appropriate end-service to handle their complaint. In addition to these physical objects we also represented the local government services as objects within the model, and linked them to their related objects. For example, a parks & gardens complaint service may be linked to the various green areas of the city, whereas the highways complaint service may be linked to the physical road infrastructure that is also represented within the model [4].

The following sections will present the way the model is populated and how the complaint application works alongside the Context Manager, describing its particular application interaction model.

5.2 Populating the Model

The complaint application can be developed by a government body, and in or trial we worked with the municipality of Vila Nova de Cerveira. The model was populated with the municipality public equipment objects and the various inter-relationships between them.

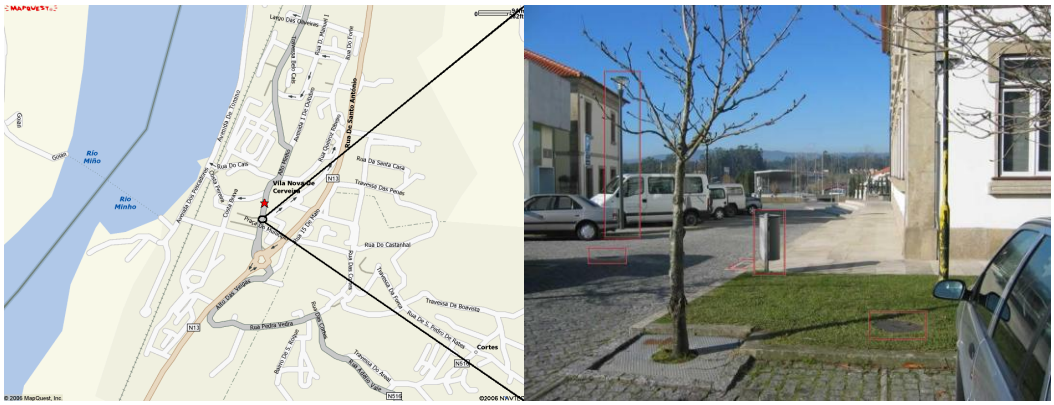

Figure 15. Vila Nova de Cerveira

Some public equipment such as roads, avenues, streets, squares, plazas, gardens, points of interest (POIs), and car parks are identifiable on maps and were added to the model. Other public equipment was indentified in the field and then later located on a map (see Figure 15).

Smaller public equipment such as public lights, trash cans, and drain covers were grouped and created as a single object in the model. For instance all the drain covers in the "*Praça Rui Diniz*" were grouped and named as "*Drain covers at Praça Rui Diniz*". Objects were grouped because in the small screen of a smart phone it is very difficult to present long lists of objects to select. Also, a "Public equipment" object was created for abstracting all the public equipment objects, and also one object for each kind of public equipment, i.e. one "Drain cover" object for abstracting all the drain covers (see Figure 16).

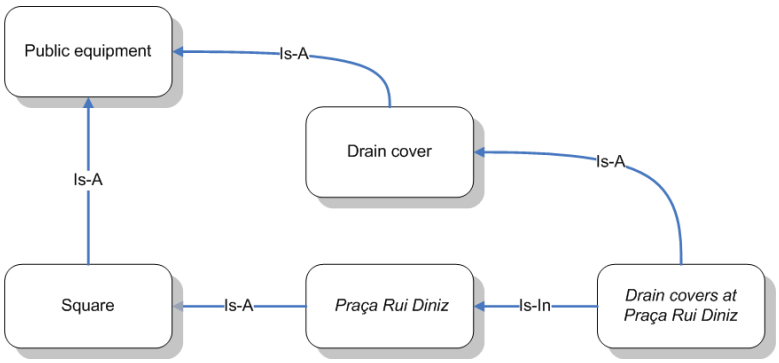

Figure 16. Object modeling

The relationship Is-A connects the abstract object to a real instance of it, , e.g. "*Drain covers at Praça Rui Diniz" Is-A "Drain cover".* Some relationships were simplified in the model. For instance, the theoretical Is-A relationship was modeled as RDF property (a literal), and the relationship Has-A were also modeled as RDF property. That modification allows the use of XML namespaces representing schemas instead of having more RDF lines, to define this so helping to reduce the size of the model files and making for a more computing efficient model.

The location dimension of the user was initially planned to be obtained from the mobile operator; but some issues (please see the section 5.2.2 Feeder implementation issues) made this impossible to use as the source of the context information. To solve this problem an alternative source of location information was used.

A run-once initialization program was made to create all the objects in the model, including relationships. The program used the Context Manager to populate the model.

There are other objects such as feeders and the Municipality complaint services that we have not mentioned here in describing the initial setup of the Context Model. These objects will be covered later in the Feeder and Service sections.

5.2.1 Feeders

One feeder that provides location was represented in the model. The feeder has two relations, one that describes the kind of relationship the feeder can provide, and other descries the "Access Point" of the feeder.

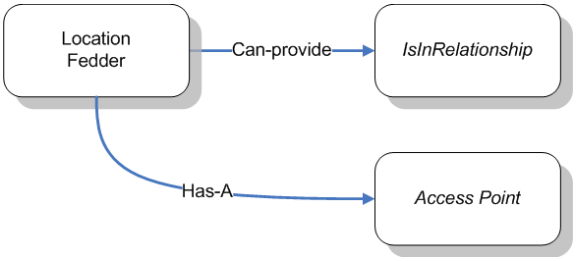

Figure 17. Location feeder representation

One feeder was created to provide location information. It provides the symbolic location of the mobile phones. These symbolic locations corresponded to areas in VNC, which were created as objects in the model. The public equipment objects were linked to these areas by creating relationships between them. The main relationships created were Is-In relationships, with some objects considered primarily as containers for other objects, such as streets, gardens and car parks. As the Is-In relationship is transitive, many other objects were included in these relationships. Relationships between the areas and the complaint services and the areas they correspond to were also created.

5.2.2 Feeder implementation issues

The Vila Nova de Cerveira (VNC) municipality does not have its own existing infrastructure with sensors that provide context information to the feeders. The external alternatives were to use the information provided by the MOs or to use smartphones with GPS devices. There are four detectable signals from MOs in VNC, three Portuguese operators at this time (TMN, Vodafone ad Optimus), and one Spanish (Telefonica). The MOs are able to provide some kind of position information, by processing the received signals from the mobile phone, it is possible to measuring the time the signals take to arrive to the antennas, or by measuring the power used for the phone to transmit the signal. For this procedure the MO must install in their antennas and reception equipment special software and hardware. Moreover the MO must develop software interfaces that allow third parts to query the location of mobile devices. The OSA Parlay[36] consortium propose an open API for this purpose. Finally, it is possible to query the mobile operator about the cell location of one mobile phone. However none of these possibilities were available in VNC from any operator.

An alternative approach is to transmit via GPRS the current cell identification from the smartphone (using some special software); with this cell identification it is possible to calculate the position of the mobile phone with a lower accuracy. In order to explore this possibility an attempt to identify MO broadcasting cells was made. Two cells were

indentified from the selected MO (Optimus). However the geographical large coverage of the cells combined with the small size of the VNC made it impossible to use this information to distinguish just two areas within the town.

Therefore we selected a solution that used smart phones equipped with GPS, where an application running in the smart phone transmitted the last area where the smart phone had been detected (by reading the coordinates from the GPS device). In the other side, the localization feeder provides the location of the mobile device when the Context Manager asked for it.

5.2.3 Complaint Services

There are four departments responsible for public equipment in the VNC Council: Water, Solid Wastes, Ways & Gardens and Lighting. For each one of these divisions a complaint service was created. The complaint services were represented too in the model as the same way as feeders.

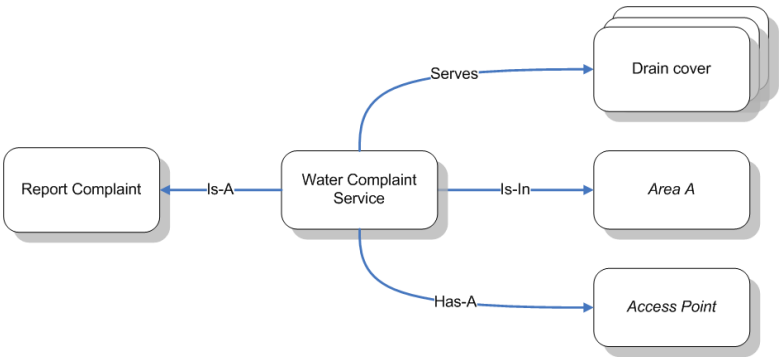

Figure 18. Complaint service representation

The Water Complaint Service has several relationships (see Figure 18). In the same way as the feeders, each complaint service has an access point that is the URL where the user is redirected after the service is discovered. The type of the service is given by the relationship Is-A Report Complaint. This categorization helps the Complain Application to discover the service. The “Serves” relationship means that this service can be used to report anomalies of the kind of objects “Drain Covers” as shown Figure 18. The Water Complaint service also has “Serves” relationships with other kind of objects represented in the model that are also related with water, such as hydrants and grating covers.

The developed complaint services have a user interface accessible via web. The user is able to view and add to the information about the complaint using this interface. The

complaint services receive input from the user, and also context information from the Context Aware application. This information is further processed by the back office in the municipality.

5.3 Context Aware Complaint Application

The Context Aware Complaint Application was developed for discovering appropriate complaint services and presenting them to the users. The complaint application has a web user interface and runs in an application container. The communication with the Context Manager is made through direct invocation (in Java) . The Context Manager is also running in the same application container. The application container used is Apache Tomcat [37].

From the point of view of the user, the application works as follows: When the user wants to report an anomaly, they run the micro browser in their smart phone and open an URL with the address of the service (it is possible that the link to this address was previously stored as a bookmark to make access quicker). A menu with a list of available applications is the presented in the browser and the user selects the link that points to the Context Aware Complaint Application. It is supposed that this complaint application is accessible in a portal of government services available to the user (see Figure 19 a). When the user selects the the Context aware complaint application it will present a menu to drive the selection of the kind of objects to report an anomaly (Figure 19 b). Then the complaint application, based on the context of the user, will present the individual objects that can present anomalies in that area (Figure 19 c). Finally the Complaint application will redirect the user to the discovered complaint service, and the user will interact with it in order to report the anomaly (Figure 19 c). Note, it is also possible to pass context information to the complaint service for further processing (e.g. the location of the user).

Figure 19. Menu driven interface showed to the user

From the user point of view it is a quite simple application, but from the Context Manager and from the Context Aware Complaint Application point of view the interactions are more complex. Those interactions are described in the next section.

5.3.1 Interaction Model

The interaction model is the way the Context Aware Complaint Application (CA) interacts with the model and with the user. It is possible to use any number of different interaction models depending on the needs of the Context Aware Application.
In summary, the CA briefly perform the following process: When the user is connected to the CA in their first HTTP-GET message, the context aware application creates the relation between the smartphone and the user. The CA ask to the Context Manager to find the location of the user, and the location feeder is asked to feed the model with the user's smartphone location. Next the CA asks the context manager for the objects near to the user (those objects are representations of both virtual and real objects). Finally the first interface (which is the response of the first HTTP-GET) is presented to the user (Figure 19 b). Finally, the CA presents the service that the user needs through a menu driven filtering strategy.

Working with micro browsers as user interface has some limitations. In general, micro browsers do not support the full implementation of technologies found in normal web browsers. For instance, technologies such as Java Script, AJAX, etc. are not available or are very limited, in micro browsers. For this reason the communication with the CA is

limited to responses (from the CA) initiated by interactions from the user. However, the advantage in using a web user interface is that many smart phones have a micro browser and the application container (in our case Apache Tomcat) manages the issues related with user session, user authentication (if necessary) and multi-user access.

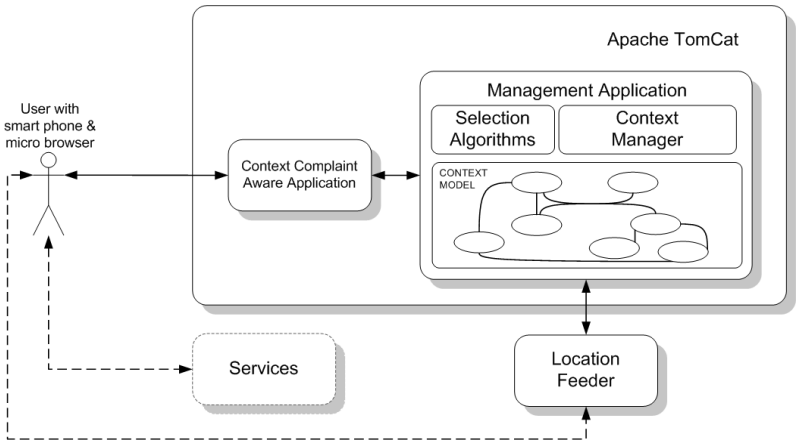

Figure 20. The general picture of the environment

The general picture of the environment is shown in the Figure 20. The interactions of the CA with the user and the feeder are presented in complete detail in the sequence diagram of the Figure 21.

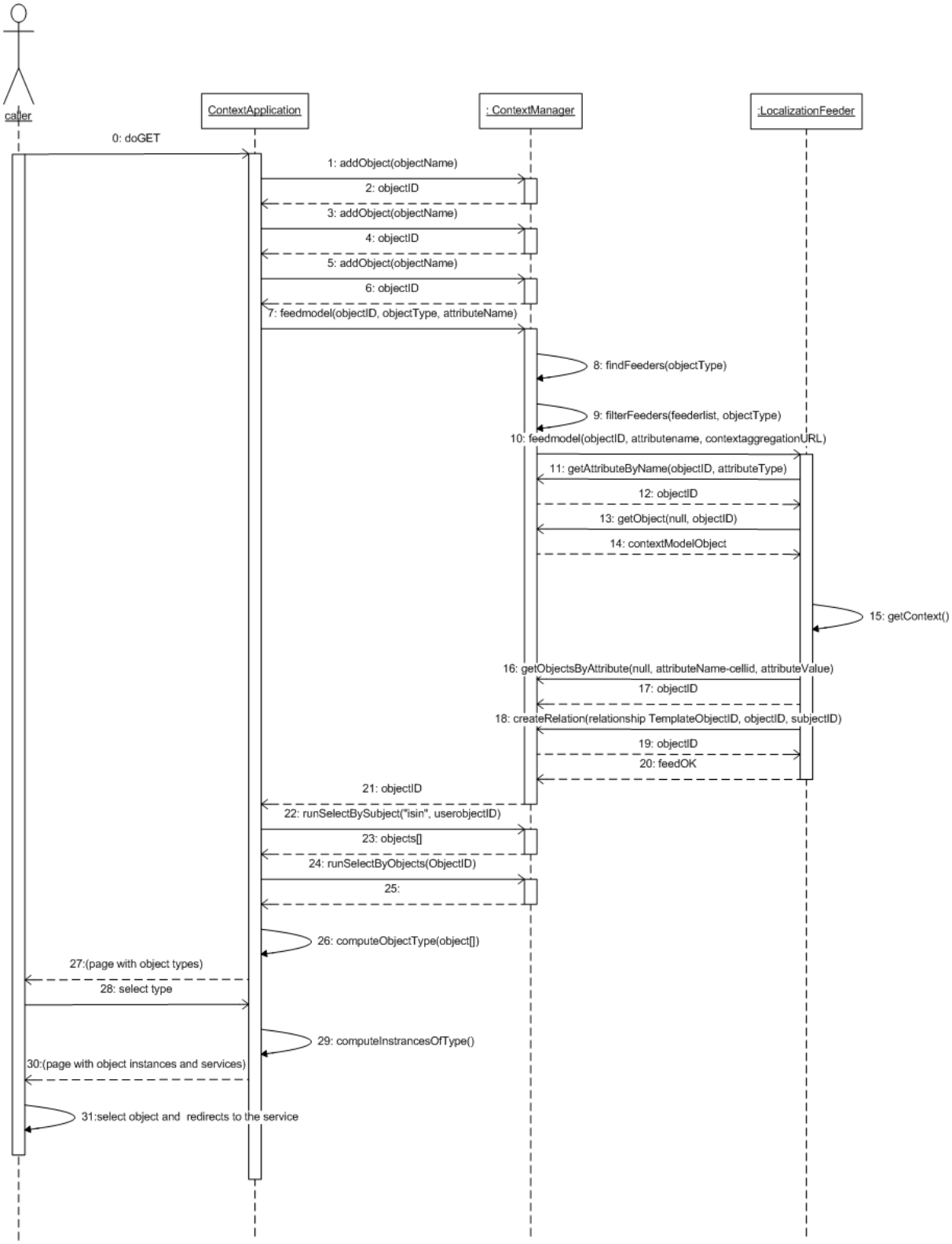

Figure 21. Complaint Application interaction model

Scenario: The CA receives a request from the user's mobile phone. The application discovers the appropriate complaint service.

0. The browser sends a get request to the CA (Figure 19 a).
1. The CA obtains a reference to the Context Manager and adds a “user” object to the object model.
2. The Context Manager returns the “user” object ID.
3. The CA adds a “mobile terminal” object to the object model.
4. The Context Manager returns the “mobile phone” object ID.
5. The CA adds a “with” relationship object with its object attribute set to the user object ID and with its subject attribute set to mobile phone object ID.
6. The Context Manager returns the “with “relationship object ID.
7. The CA requests the Context Manager to feed the mobile phone object with an attribute named “Is-In” and of the type “Is-In relationship”. This corresponds to requesting the mobile phone’s position.
8. The Context Manager finds all feeders that can feed an object with an Is-In relationship.
9. This set of candidate feeders are filtered so that only feeders whose needs are satisfied by the object´s attributes are considered.
10. The Context Manager requests the feeders to feed the mobile phone object with the Is-In relationship, and to name the attribute with the specified name.
11. The feeder requests the Context Manager for the TerminalID attribute of the mobile phone.
12. The TerminalID attribute is returned.
13. The feeder requests the object with the objectID attribute (the mobile phone).
14. The object (the mobile phone) is returned.
15. The feeder obtains the mobile phone´s location from the mobile operator or for the GPS system.
16. The feeder finds the object in the Context Manager that represents the cell id or the area returned from the mobile operator or the GPS system.
17. The id of the cell object is returned.
18. An Is-In relationship is created between the mobile phone and the cell or area.
19. The object ID of the new relationship is returned.
20. The feeder returns to the Context Manager the objectID of the new object
21. The Context Manager returns to the application the objectID of the new object.
22. The application requests from the Context Manager the user location context.
23. The Context Manager returns the user location contexts.
24. The CA requests from the Context Manager the objects that are located in user location context.
25. The Context Manager returns the list of objects co-located with the user.

26. The application computes the types of objects co-located with the user.
27. The application returns to the user the list of types (Figure 19 b)
28. The user selects the complaint target object type
29. The application computes the objects of selected type.
30. The application computes the Complaint services associated to objects of select type (Figure 19 c).
31. The user selects Report complaint service associated to respective object (as a hyperlink)
32. The user is redirected to the complaint service by selection the desired object (Figure 19 d).

One of the advantages of the separation of the Context Manager and the Context aware applications is the possibility to create new Interactions Models. Also it is possible to make small changes in the Interaction Models (for instance in model the presented Figure 1) and obtain a new behavior without modifying the Context Manager and the applications that provide contextual information (feeders).

5.4 User Testing

In order to test the Context Manager, it is necessary to test it with all other applications showed in the Figure 20. Those applications are the Context Aware Application (e.g. Complaint Application), the context feeder (e.g. localization service) and services to be discovered (e.g. Complaint Services).

A test was conducted in Vila Nova de Cerveira to know the users thoughts about the complaint application. Since only some components present a user interface (i.e. the Context Aware Application and Complaint Services), the user only sees theses interfaces and will therefore only produce direct feedback about these applications. However for the correct functioning of those applications and user interfaces, the Context Manager must be working well.

5.4.1 Test scope

In the test three tasks were selected for the users to perfom, as follows:

- Make a complaint
- Check Complaint Status
- Cancel Complaint

The first task, Make a complaint, needs all the steps that Figure 21 shows. The other two tasks (Check Complaint Status and Cancel Complaint) can be performed without those steps. The user interface is presented in a web micro browser, where also the interface is composed mainly by text options; and where each option is an hyperlink to the next step. The Figure 19 shows the user interface style.

Ten random testers were selected to use the application. The users were divided in three groups, two groups of 3 people and one of 4 people. Then each group made a walk in different areas of the city (VNC) and were asked individually to use the application (performing the three tasks).

Of the 10 users participating, 4 users had previous experience of using mobile applications and 6 users had no previous experience in the use of mobile applications. Eight were man, two were female.

One of the key questions to asked to the test users was if they understood the menu driven approach derived from the Interaction Model presented in the section 5.3.1. For this purpose evaluated if the user recognized the elements of the interface (Recognition Exercise). Another important point to respond was to check if the users could complete each one of the three tasks (Performance Exercise). Finally some impressions and feedback were collected from the users. In particular we wished to understood if they noticed and understood the use of context-awareness in the application.

5.4.2 Findings

5.4.2.1 Recognition Exercises

As seen from the table below, element recognition of the different elements of the interface was on average very good. This was due to the fact that this application is basically based on text interactions. Sometimes even, text was in excess and some graphical elements could have helped the interaction.

Table 6. Recognition of the interface elements

Elements \ Users	01	02	03	04	05	06	07	08	09	10	Avg.
Description of pages	3	3	3	2,93	3	3	3	3	3	2,92	2,9
Available Functionality	3	3	2,73	2,86	3	3	2,85	3	3	2,92	2,9
Task Icons	3	3	2,73	2,66	2,86	3	2,85	2,85	3	2,64	2,8
Average per user	**3**	**3**	**2,82**	**2,81**	**2,95**	**3**	**2.9**	**2.95**	**3**	**2,82**	

Scale: 3 = Total recognition 2 = Partial 1 = no recognition.

5.4.2.2 Performance Exercises

Performance exercises in these type of tests should be centred on the successful of the task completion instead of how long it took completion time. This is because of the artificial time constraints of the “think aloud” method (where the test group was accompanied and asked questions while they used the application) performed in these tests. Also, it was the first time they had used the application and, in many cases, the first time they had used a smart phone device). Therefore the results were focused in the success rate for task completion rather that the time they took.

Table 7. Task achievement results

	Make Complaint	Check Complaint	Cancel Complaint	Avg. per user
User 01	3	3	3	3
User 02	3	3	3	3
User 03	2	2	3	2,33
User 04	2	2	3	2,33
User 05	2	3	3	2,66
User 06	3	3	3	3
User 07	2	3	3	2,66
User 08	2	2	3	2,33
User 09	3	3	3	3
User 10	2	2	3	2,33
Avg. per task	**2,4**	**2,7**	**3**	**2,66**

Achievement degree scale: 3 = achieved. 2 = partially achieved. 1 = not achieved. 0 = abandoned.

The success rate for most tasks was very good, all tasks were achieved in most cases though in a few cases only partially. On a user-by-user basis there were 4 users which had clear difficulties completing the tasks, nevertheless they did succeed.

5.4.2.3 User Impressions and feedback

Although the user´s had no direct interaction with the model they clearly understood the benefits derived, such as the filtering of information and some users noticed the change of elements in the menus when they change the area of testing. This was thanks to the Context component which helped the elements presented in the user interface to change according to the area. Also the users noticed that the complaints were sent to the right service. The users understood the application interface.

In summary the user's understood the user interface and application functionality, and appreciated the benefits of how context awareness was used. These features were provided through the context model and therefore, although the users had no direct interaction with it, it certainly made a beneficial impact to the application user experience.

6 General Conclusions

This dissertation presented some key areas for consideration in developing a Context Aware Mobile Government Application. Initially we presented the key requirements in order to develop such an application. A set of core applications is necessary in order to support the creation of new context aware applications, and this set of applications must understand context in the same way and hide the complexities of context from other applications. An understanding of context can be reached by using an approach to model the surrounding environment. Modeling the context with an object model result in a very compressible way to make abstractions of the world, and an object-oriented construction is capable of representing practically any part of our surrounding environment. More specifically, location, often considered an object or property unto itself, may be modelled as a relationship between two objects, so allowing any object to be considered as a place. Practically all the model's elements may be represented using the same generic object construct, including relationships, types, and class hierarchies. The object model was implemented using RDF, which allows the representation of knowledge. There are several tools that help to manipulate RDF in different stages of the developing and deployment.

The developing and deployment of a Context Aware Mobile Application was also presented. The Context Management application was developed using open standards to intercommunication with other applications. A real scenario in Vila Nova de Cerveira was also modeled and represented. In this scenario two hundred and twenty objects were modeled incliding public equipment, relationships and services.

Some challenges were faced in the development process. One issue was pertaining how the context aware applications must interact with the Context Manager application. For a complaint scenario an interaction model were defined. This interaction model is presented to the user as a context-filtered menu driven application. One of the advantages of our general approach is the freedom to use the Context Manager in several ways, by modifying or creating new interaction models and by adding different selection algorithms for object querying.

Another challenge faced in the deployment of the solution was in the provision of context information. The location context dimension of mobile phones (or users with mobile phones) was intended to be acquired by the cell identification from the mobile phone

operator. However, technical and bureaucratic issues made this source unsuitable. The solution for the location information was to create a service that feeds the location to the model using as a source a GPS embedded in the mobile device. This additional development also demonstrates in some way the openness and the flexibility that the approach presented in this dissertation brings to new extensions, deployments and developments. It also demonstrates the need to allow the evolution of the infrastructure for supporting context aware mobile government applications, mainly regarding context acquisition.

Below we present how our design decisions and developed components met the requirements of the context aware applications presented in chapter two. Each subtitle corresponds to a requirement and the text corresponds to how the solution dealt with such a requirement.

- *Requirement: A model for context information is necessary*

  The requirement was solved by using the model proposed in chapter three. The model for context information is based in objects and relationships between those objects where relationships are also modeled whit the same object construct. Modeling using objects and relationships is a very understandable form to create abstractions of the surrounding environment. The conceptual model allows modeling several kinds of entities, such as material objects (e.g. places and people), areas or regions, and virtual objects such as user-services.

- *Requirement: It is necessary to represent the model for context information using adequate data structures.*

  The model was represented withg RDF. RDF gives formality to the model and allows a consistent representation. There are tools such as Jena and Protégé [33] that support RDF. Jena is set of Java classes that allow manipulating RDF in a higher level. Protégé is a very versatile tool and helps to model knowledge. Since RDF is not intended to be human readable, initially the models for specific scenarios can be incubated in Protégé and later converted to RDF to be used in context aware applications.

- *Requirement: The management application must allow other applications to communicate with it in a standard way*

  The Context Manager is a Java Class that can be used by direct invocation or by invoking web services. An API was defined to for the both ways to interact with

the Context Manager. The definition of web services found in [38] is: “Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. Web services are characterized by their great interoperability and extensibility, as well as their machine-processable descriptions thanks to the use of XML. Programs providing simple services can interact with each other in order to deliver sophisticated added-value services”. The Web Services is a recommendation of the World Wide Web Consortium. By using the API and also web services in the Context Manager the communication requirement is achieved.

- *Requirement: The management application must allow the addition of new forms of context information.*
  This requirement can be divided and solved in two parts. One part is about accepting new forms of context and the other is about acquiring context information. Firstly the selected model allows the representation of new context information. For instance let us imagine the hearth beat of the user as new context information. Initially the hearth beat is modeled as an object and relationship can be created between beings with a heart and objects representing hearth beat. It is clear that it is possible to model new forms of context. Secondly, in order to acquire heart beat it is necessary to have a feeder that uses the provided API of the Context Manager to feed the contextual information in some way. The development of new feeders it is also achievable because the provided API.

- *Requirement: It is necessary to add value to the context information.*
  This requirement is achived by inferring new context from known context, and also by transforming context. For instance, a feeder could transform GPS coordinates into a symbolic representation (by using a GIS). Furthermore, the model infers which objects are *near-by* other objects. This knowledge (*near-by*) is an added value to the context.

- *Requirement: It is necessary to hide complexity of context from the applications by supporting them in using context*.
  By using the Context Manager the application does not need to deal with the context information acquisition, storage and management. The Context Manager is in charge of this task. The acquisition is made by feeders and the Context Manager invokes them. Regarding querying context information, although the

Context manager can be queried at a high level by using the exposed API, the context aware application must know which algorithms are suitable to be used (or new ones developed if necessary). However, the selection algorithms (such as the Is-In Algorithm) can be seen as a high level query method, and if is possible to say that most of the complexities are hidden to the applications.

References

1. José, R., et al. *Os Sistemas de Informação Geográfica no suporte a Serviços Móveis para o Cidadão*. in *ESIG 2004 - National Conference on Geographical Information Systems*. 2004. Lisbon.
2. Rodrigues, H., J. Pascoe, and C. Ariza. *On the Development of an Open Platform for m-Government Services*. in *FIP TC8 Working Conference on Mobile Information Systems (MOBIS'05)*. 2005. Leeds, UK: Springer.
3. USEMEGOVConsortium. *The USE-ME.GOV Project Web Site*. 2006 [cited 2006]; Available from: http://www.usemegov.org/.
4. Pascoe, J., H. Rodrigues, and C. Ariza. *An Investigation into a Universal Context Model to Support Context-Aware Applications (accepted paper)*. in *Second International Workshop on Context-Aware Mobile Systems*. 2006. Montpellier - France: Springer Verlag.
5. Merriam-Webster-online-dictionary.
6. Abowd, G.D., et al., *Towards a better understanding of context and context-awareness.* Handheld and Ubiquitous Computing, Proceedings, 1999. **1707**: p. 304-307.
7. Schilit, B., N. Adams, and R. Want. *Context-aware computing applications*. in *Workshop on Mobile Computing Systems and Applications*. 1994. Santa Cruz, CA, USA.
8. Mostéfaoui, G., J. Pasquier-Rocha, and P. Brézillon. *Context-Aware Computing: Aguide for Pervasive Computin Community*. in *International Conference on Pervasive Services, 2004.* 2004.
9. Cheverst, K., et al. *Experiences of Developing and Deploying a Context Aware Tourist Guide: The GUIDE Project*. in *MOBICOM*. Boston, MA, USA: Dey, A.K., Abowd.
10. Dey, A.K., G.D. Abowd, and D. Salber, *A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications.* Human-Computer Interaction, 2001. **16**(2-4): p. 97-+.
11. Chen, H., F. Perich, and T.J. Finin, A. *SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications*. in *International Conference on Mobile and Ubiquitous Systems: Networking and Services*. 2004. Boston, MA.
12. Strang, T., C. Linnhoff-Popien, and K. Fran, *CoOL: A context ontology language to enable contextual interoperability.* Distributed Applications and Interoperable Systems, Proceedings, 2003. **2893**: p. 236-247.
13. Chen, H., T. Finin, and A. Joshi. *A context broker for building smart meeting rooms*. in *Knowledge Representation and Ontology for Autonomous Systems Symposium*. 2004.
14. Lamsweerde, A.v. *Requirements engineering in the year 00: A research perspective.* in *22nd International Conference on Software Engineering*. 2000: ACM Press.
15. Strang, T. and C. Linnhoff-Popien. *A context modeling survey*. in *UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management*. 2004. Nottingham.
16. Pascoe, J., *The stick-e note architecture: extending the interface beyond the user*, in *Proceedings of the 2nd international conference on Intelligent user interfaces*. 1997, ACM Press: Orlando, Florida, United States.
17. Martin, D., et al. *DAML Services*. 2006 [cited 2006 2006/08/28]; David Martin:[Available from: http://www.daml.org/services/owl-s/.
18. Wang, X., et al., *Semantic Space: An Infrastructure for Smart Spaces.* IEEE Pervasive Computing, 2004. **Vol. 3, No. 3**: p. 32-39.
19. OpenGIS. *Geography Markup Language*. 2006 [cited 2006 17-08-2006]; Available from: http://www.opengeospatial.org/standards/gml.

20. Wang, X., et al. *Ontology Based Context Modeling and Reasoning using OWL*. in *Context Modeling and Reasoning Workshop at PerCom*. 2004.
21. Strang, T., C. Linnhoff-Popien, and K. Frank. *Integration Issues of an Ontology based Context Modelling Approach*. in *Proceedings of the IADIS International Conference WWW/Internet*. 2003. Algarve, Portugal.
22. Chen, H.L., *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*, in *Faculty of the Graduate School*. 2004, University of Maryland: Maryland.
23. Gu, T., et al. *An Ontology-based Context Model in Intelligent Environments*. in *Communication Networks and Distributed Systems Modeling and Simulation Conference*. 2004. San Diego, California, USA.
24. OpenLS. *Open Location Services*. 2006 [cited 2006 17-08-2006]; Available from: http://www.opengeospatial.org/projects/initiatives/openls-1.1.
25. Cisco-Systems. *Sweden's Stockholm Subway Implements Indoor Location-Based Services Platform with Cisco Systems and Appear Networks*. 2006 [cited 2006 18/08/2006]; Available from: http://newsroom.cisco.com/dlls/partners/news/2006/pr_prod_05-02.html.
26. Christensen, E., et al. *Web Services Description Language (WSDL) 1.1*. 2001 [cited; Available from: http://www.w3.org/TR/wsdl.
27. Pinto, H.V., Noe and R. José. *Using a private UDDI for publishing location-based information to mobile users*. in *7th ICCC/IFIP International Conference on Electronic Publishing (ELPUB2003)*. 2003. Guimarães, Portugal.
28. Martin, D., et al. *OWL-S: Semantic Markup for Web Services*. 2004 [cited 2006 24/10]; Available from: http://www.w3.org/Submission/OWL-S/.
29. W3C. *Resource Description Framework (RDF)*. [cited 2006 2006/09/21]; Available from: http://www.w3.org/RDF/.
30. *Hibernate*. [cited 2006; Available from: http://www.hibernate.org/.
31. Michael, K., L. Georg, and W. James, *Logical foundations of object-oriented and frame-based languages.* J. ACM, 1995. **42**(4): p. 741-843.
32. Ranganathan, A., et al., *Use of ontologies in a pervasive computing environment.* Knowledge Engineering Review, 2003. **18**(3): p. 209-220.
33. John, H.G., et al., *The evolution of Protégé: an environment for knowledge-based systems development.* Int. J. Hum.-Comput. Stud., 2003. **58**(1): p. 89-123.
34. Labs, H. *Jena – A Semantic Web Framework for Java*. [cited 2006 2006-10-03]; Available from: http://jena.sourceforge.net/.
35. Prud'hommeaux, E. and A. Seaborne. *SPARQL Query Language for RDF*. 2006 [cited 2006; Available from: http://www.w3.org/TR/2006/CR-rdf-sparql-query-20060406/.
36. Consortium, T.O.P. *The OSA Parlay Initiative*. [cited 2006 2006-10-15]; Available from: http://www.parlay.org.
37. The-Apache-Software-Foundation. *Apache Tomcat*. 2006 [cited 2006 2006-10-16]; Available from: http://tomcat.apache.org/.
38. W3C-Advisory-Committee. *Web Services Activity Statement*. 2006 [cited 2006 10/22]; Available from: http://www.w3.org/2002/ws/Activity.