INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Presença: Aplicações Conscientes de Contexto

João Manuel Pinto Peixoto

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Júri

Presidente:	Professor José Delgado
Orientador:	Professor Luís Veiga
Vogal:	Professor Pável Calado
Acompanhante:	Engenheiro Paulo Chainho

Outubro 2009

Abstract

Social interaction through information technology is become prevalent, namely in working and project environments where team work is decisive.

The objective of this project is to create a context-aware framework capable of gathering context and address information regarding a multiplicity of contact/context sources, to reason based on such information and provide the user with the best action available for the desired interaction.

During the project we analysed state of the art techniques for context representation and reasoning, social groups interaction and transformation methodologies (ETL).

An architectural design is presented based on which the project was developed, taking into consideration the information architecture as well. Specific implementation-related considerations had to be addressed, mainly due to company constraints, along which some technology-related issues had to be solved.

The evaluation of the project proved the importance and relevancy of the developed system, as well as its compliance and performance. There is still room for future work, regarding duplicate detection among other subjects, but the achieved results are already very encouraging.

Resumo

Interacção social através das tecnologias de informação é cada vez mais dominante, nomeadamente em ambientes colaborativos em que o trabalho em equipa é fundamental.

O objectivo deste projecto consiste no desenvolvimento de uma *framework* capaz de obter informação de contexto e endereços provenientes de uma multiplicidade de fontes, interpretar e processar essa informação sugerindo ao utilizador a melhor acção disponível no contexto da interacção presente.

No desenvolvimento do projecto foram estudadas diversas técnicas actuais referentes à representação e interpretação de contexto, interacção com grupos sociais assim como metodologias de transformação de informação (ETL).

A arquitectura na qual se baseia o projecto é também apresentada, não negligenciando a arquitectura de informação. São apresentadas considerações específicas da implementação, principalmente devido a normas empresarias impostas, assim como alguns problemas e alternativas relacionadas com limitações tecnológicas.

A avaliação do projecto demonstrou a sua importância a relevância, confirmando o seu desempenho e utilidade. Ainda existe espaço para trabalho futuro, nomeadamente na área de detecção de duplicados entre outros, contudo, os resultados obtidos só por si muito encorajadores.

Contents

Chapter 1

Introduction

It's undeniable the wide spread of mobile devices, from cellphones to laptops, PDAs or handhelds. This devices have the capability of easing everyday's life tasks, organize schedules, share information, allow anywhere-anytime communication and so on. This detachment from fixed locations has however major implications in software design. How can we understand the global behaviour of multiple, concurrent, independent and self-capable devices?

We do understand the behaviour of each component in isolation, however **we do not** understand the global interaction of components[2, 9], which places new challenges upon us [45]:

- Scalability
- Adaptability
- Context-Awareness

Pervasive (or ubiquitous) computing emerged from these new challenges[56]. Pervasive computing consists of countless "pervasive" devices equipped with limited resources and computing power that in conjunction support end-to-end applications which far exceed their own capabilities.

The complexity of an application is directly proportional to the difficulty to manage it, therefore, the growth of pervasive computing is pushing software engineers towards the threshold of incapability to manage such systems.

Context-aware software aims to simplify the management of simple systems (which require user-intensive interactions) or complex systems that require intensive management. But what exactly is a context-aware system? Context-aware systems are no more than applications, either in a network or in a device, that are aware of their surrounding environment and through which they are capable of adapting their operations dynamically and automatically in order to ease our interaction [20, 30, 5, 6].

A main issue arises: how to define context. The easiest resource of definitions (dictionary) states that context is

> *"The situation within which something exists or happens, and that can help explain it."*[1]

In computer science such a definition is rather generic, and several tries have been made in order to achieve a common, acceptable definition which addresses context-aware problems.

Many definitions are, in opposition, problem-specific and therefore unusable in a generic context-aware system problem.

To better understand what context is (and through this method attain a consensual definition) we will address the human behaviour.

As sentient beings, humans are context aware. We perceive our surrounding environment and we use such information to decide the best action/reaction to take in a given interaction. The capability of using information from different sources, produce knowledge about the environment and use previously acquired information (through experience or education) in order to take the best action, is a major advantage towards the usage of context. We will analyse in more detail de human-computer context usage.

No doubt humans are much more complex than machines, but still machines are very complex and as stated before, computer systems are drastically increasing in complexity. If these systems were able to perceive, process and use context, their inner capabilities would be greatly expanded and the need for human intervention either reduced or simplified. An automatic pilot being able to adapt its flight pattern given a malfunctioning engine, a medical monitoring device capable of anticipating a heart attack. The options are endless.

The environment where this project was developed is similar in the basic characteristics. Communications can take place in a wide range of manners, using several different technologies, means of communication and communication end-points. The problem is to understand the best way to communicate. This process, although simple to do for a human when it comes to a small number of contacts, is rather complex when dealing with a large number of them.

However, as Alexander Pope said, "*To err is human*". Information perceived and/or inferred is not always trustworthy. Sometimes we are not able to obtain all the information required to make a proper decision. Decisions solely based on guessing and minor observation can lead to a major crisis. Other mistakes have far less impact on a global scale, and are purely language ambiguity related, for instance: "*I saw a man in the hill with a telescope*" mean that I used the telescope to see the man? Or the other man had the telescope and I saw him with it?

The same happens in computer science. Computers perceive context in several different ways (see 2.1.3) but, as well as humans, perceived information has a level of confidence.

[1] Cambridge Advanced Learner's Dictionary

Does a context-aware system have the authority or the legitimacy to take actions on our behalf?

Context can be distinguished in two different source types. User-defined context and perceived context. As for the first one, user is responsible for defining and updating its own context, but do users perform such actions? Maintaining a coherent information database? The answer is "mostly no", not in all devices/systems at least. An autonomous system capable of updating the user's context information would keep it up-to-date, on the other hand it would take away control from the user. Most users however, though feeling a control decrease, still prefer such approach [7], but normal users are not conscious enough to comprehend the extension of delegating personal decisions to a context-aware system, privacy and security also add to this problem. Bellotti et al. strongly argue that human ultimate control should be always assured [9].

Despite the problems of user-defined context, merging different perceived contexts and infer a "general context", or be capability to dynamically conclude the user's context can provide a system with powerful intelligent communication tools.

The aim of this project is to provide a framework capable of gathering context from different sources, understand that levels of confidence are associated with such context and infer, based on dynamically defined rules, the best way to communicate with an user.

Through this dissertation we will approach several aspects related to this work.

In section 2 we will analyse existing approaches, ranging from context definition (see 2.1.1), where we will discuss the concept of *context* per se. Human and computer considerations will be taken into account. Context representation will also be address (see 2.1.2) through a series of comparisons.

Context sensing (2.1.3) will focus on techniques and used approaches to gather contextual information, distinguishing between direct (perceived) context and indirect (inferred) context.

We will then approach reasoning (see 2.1.4) for a more detailed overview of the (pre)processing possibilities for a context-aware system. Existing infrastructures (see 2.1.4.4) will also be covered at the aim of defining the most efficient and effective design paradigms.

Our architecture for the problem in hands (see 4) will follow, providing the basic concepts related to it. We will important implementation aspects (see section 5) addressing some technology-related issues (see section 5.2) in the process.

To finalise we will assess the performance of the system (see section 6) through which conclusions are made (see section 7).

Chapter 2

Related Work

The project presented in this work is built on several different technologies, areas and techniques. Context is without a doubt the biggest dimension of this work, it represents information and meta-information about an entity, information which can be used to enrich an application, providing a wider range of functionalities. It is however a complex subject. We will focus on several aspects of context, from its meaning, to its representation and reasoning.

Along side context, social groups represent a very popular source of information. It is undeniable the massive usage of these services, which make them an important source both for information, endpoint addresses and valuable users. We will focus on the architecture and protocols of these systems, their interaction with the project and the information available through them.

Having several different sources of information, extraction, transformation, and loading (ETL) of such information becomes a very important problem that must be addressed. This problem will be approached by analysing different merging techniques, conflict resolution approaches along side with available paradigms.

2.1 Context-Awareness

Any entity, being a person, object or place, has associated information, whether referring to activity status or room color, even object temperature. Obviously not all information regarding an entity is relevant, but we will address this issue later on this document, what is important however is that in the right environment, this information does indeed provide to a system parameters that can be used to better solve a problem, more efficiently and more adequately given the current context of the participants.

In figure 2.1 may we understand that context information allows an application to take the adequate action automatically and autonomously, in the example, selecting the most appropriate network connection.

It is undeniable however that this subject has several complex dimensions. Throughout

this chapter we will discuss the definition of context itself, hence allowing for the distinction between relevant and irrelevant information, enabling the system to focus on what is important.

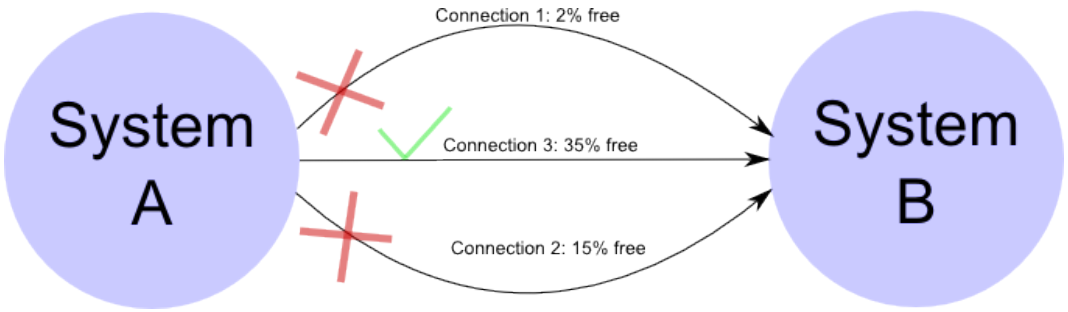

Figure 2.1: Example of Context-Aware Choice

We will then discuss context representation techniques, trying to achieve a representation that is able to encapsulate the existing relevant information, enabling actions towards such information and avoiding data loss.

Context reasoning, a more complex subject, will then be discussed. Existing techniques of inference will be addressed as well as different levels of context reasoning, ranging from active, passive and informative approaches.

We will then finalise with deployed infra-structures that already work with context in order to gather relevant solutions that might be adapted to the project's problem.

2.1.1 What is context?

When we discuss pervasive computing and context-aware systems, we must first decide what **context** is in fact. Although it is a dictionary word, its meaning is not so straightforward. Is every information regarding a particular entity relevant at any time? Can we even gather every information regarding an entity? How can we determine which information should we gather and use?

In this chapter we will deepen this subject, first by analysing human behaviour and human usage of context. As sentient beings, human interactions are a rich source of information and guidelines on how to perceive and use context.

We will follow with the computer needs and capabilities and we will try to assess a common definition of context, and ultimately how can humans and computers exchange information.

2.1.1.1 Common understanding of context

If we ask *"what context is"* most people will say "something that describes a domain, situation or environment", or "situation where *something* happens", but those people show great difficulty in defining context with words.

In the dictionary context is defined as

> *"The situation within which something exists or happens, and that can help explain it."*[1]

It is a good definition, but rather generic. In order to properly design and create context-aware systems, we must decide which information is relevant to an interaction and which is not.

To achieve this *common understanding* of context we are going to analyse the human perspective of it.

2.1.1.2 Human perspective of context

If we look around and pay attention to human interactions, we will find that humans are indeed context aware. We can infer intentions and make decisions based on the surrounding environment and personal variables.

Considering two different case studies (we will analyse this matter later in this section)

- **Situation 1**: Alice asks Bob to close the door. Bob will (correctly) assume that the door to be closed is the one from the room where they both stand in.
- **Situation 2**: Charlie is driving. When he comes across a broken vehicle, and knowing that no other car is coming in the opposite direction, he knows that he may cross the lane in order to proceed his march.

This ability to use implicit situational information makes us quite successful at conveying ideas to each other and reacting appropriately, broadening the bandwidth of a conversation or the range of an action[18].

We must address other components of context. The most desirable way would be to find a directly mappable structure to computer science. To obtain such structure in an understandable perspective, human perception and usage of context can be assessed through several human capabilities which grants us context abilities[13]:

- **Ontology sharing**, meaning that humans are able to share communication languages and vocabularies.
- **Sensing**, as humans being capable of perceiving their surrounding environment through sensing organs.
- **Reasoning**, meaning that humans are able to make sense out of perceived information, already known information, experience and social conditions.

As we can see, not all context is well interpreted or even relevant. Analysing again situation 1 (see 2.1.1.2) we can assume that location was undeniably part of the context. Although, if we analyse a slightly different interaction, for example, Alice's tone of voice when asking the question was somehow dishonest and her intention (perceivable through that tone of voice) was to get ride of Bob, sending him to close a door elsewhere. We then

[1] Cambridge Advanced Learner's Dictionary

must assume that the tone of voice is considered context as well. In many cases the robustness of human-to-human communication is based on the implicitly introduced contextual information, such as gestures, body language, and voice[49].

The mentioned implicit information is probably the most difficult to map to machines, mainly due to the fact that we have difficulties to understand the extent to which we use such implicit knowledge. We must however understand that basic object information (color, position, status, etc.) are not enough to decide or take actions upon.

2.1.1.3 Computer perspective of context

In computer science the definition of context has been widely discussed. We will analyse several approaches to defining such concept and hopefully provide an acceptable, simple and correct definition.

First definition attempts

Content-aware was first defined by Schilit and Theimer[47], refereing to context as

> "*The location, identities of nearby people, objects and changes to those objects*"

As location being the most widely used property in context-aware systems [5, 54, 4, 15], a natural reaction is to try to define context through such a property, although, as previously mentioned (see 2.1.1.2), context is a much more comprehensive concept, even including implicit information.

Ryan et al. took a more concrete approach, but still fixed in a relatively strict set of variables[41], defining context and context-awareness as

> "*The ability of the computer to sense and act upon information about its environment, such as location, time, temperature or user identity*"

In a first attempt to solve this dilemma, Dey in its Cyber desk project states[17]

> "*Context includes, but is not limited to, information the user is attending to, emotional state, focus of attention, location and orientation, date and time of day, objects and people in the user's environment*"

Some authors try to define context through synonyms, Hull et al. [26] understood context as

> "*The aspects of the current situation*"

We then fall into the common understanding of context (see 2.1.1.1), being too generic to directly design and/or prototype a context-aware system. Brown found a particularly interesting definition [10]

> "*Context consists of a combination of elements of the environment that the user's computer knows about*"

	Alice	Bob	Door	Room
Context Information	Clothing Mood Age Etc.	Clothing Mood Age Etc.	Size Color Mechanical/Automatic Etc.	Color Temperature Number of doors Etc.

Table 2.1: Context information based on Dey's definition

Proposed definition

Context is all about the whole situation relevant to an application and its set of users. We cannot enumerate which aspects of all situations are important, as this will change from situation to situation. Dey produced a very accurate definition of context that takes into account these concerns[18].

> "*Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.*"

It becomes much easier to assess context information in a specific situation/interaction using this definition, and through that assessment designing and prototyping of applications are largely empowered, but still, we must be able to identify what is important for each interaction, rather than using all information available, even if it is specific to an entity.

Lets analyse situation 1 again (see 2.1.1.2) using Dey's definition. The actors of this interaction are (roughly) Alice, Bob, the door and the room itself. These entities have properties such as age, mood, color, clothing and so on. On table 2.1 we summarize a possible set of information. Among all data available, some are truly important to the interaction, like Alice's mood, the number of doors on the room, just to point out a few. But is Bob's clothing relevant for instance? Lets study another, slightly different, interaction.

- **Situation 1b**, Alice asks Bob to adjust the air-conditioning temperature level.

Situation 1b requires a completely different set of information, for example Alice's clothing. The room's color plays no part on it.

A limitation found in Dey's definition consists on neglecting what is considered contextual information (or simply entity's information) and context per se. In a more formal way, we have:

$$\begin{aligned} Context = \quad & \exists_{i,a,b}(Interaction(i), Actor(a), Actor(b)) \wedge \\ & \wedge(\forall_{infoA=Information(a)} \wedge \\ & \wedge\forall_{infoB=Information(b)} \\ & Useful(a, i, infoA, InfoB) \wedge Useful(b, i, infoA, infoB)) \end{aligned}$$

InfoA and *infoB* are simply **every** information regarding a and b respectively. It is precisely the distinction between general information and context that we believe to be

important for context-aware systems design and prototyping.

$$\begin{aligned} Context = \quad & \exists_{i,a,b}(Interaction(i), Actor(a), Actor(b)) \wedge \\ & \wedge(\forall_{infoA \subseteq Information(a)} \wedge \\ & \wedge \forall_{infoB \subseteq Information(b)} \\ & Useful(a, i, infoA, InfoB) \wedge Useful(b, i, infoA, infoB)) \end{aligned}$$

Through our definition, *infoA* and *infoB*, for a specific interaction i, consist of a **subset** of a's information and b's information respectively, opposing to the whole set. Therefore, our definition of context is

> *Context is relevant information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. Relevant information means information that, given a specific interaction between a user and an application, is useful and of meaning to that interaction and intervening actors.*

The main difference consists in distinguishing information and context. Although an entity can be described in different levels, not all information is useful on a specific interaction.

There are immediate consequences in system performance. Processing a complete set of information opposed to a sub-set of said information. Although an overhead exists due to the distinction that has to be made in order to only use *relevant* information, the end-result can be more efficient.

2.1.2 Context representation

A context model is needed to define and store context data in a machine processable form. To develop flexible and usable context ontologies that cover the wide range of possible contexts is a challenging task[5].

Whilst combining information, context representation must also be dynamic, both referring to extensibility and interoperability.

Local processing capabilities must be taken into consideration as well. Devices that interact with the system might not possess the computing power required by some context representations or associated processing tasks.

In any interoperable system standards enable a well-defined, commonly accepted way of communication between different systems.

These represent the dimensions that must be taken in consideration.

Existing techniques

Strand and Linnhoff-Popien[52] researched this particular issue, establishing a group of used representations and their target areas.

We will analyse different representation models, namely the key-value model, mark-up scheme model, graphical model, object-oriented model, logic-based model and ontology-based model.

2.1.2.1 Key-Value models

These models represent the simplest data structure for context modelling. They are frequently used in various service frameworks, where the key-value pairs are used to describe the capabilities of a service. Service discovery is then applied by using matching algorithms which use these key-value pairs.

Schilit et al. developed a context-aware system where context information entered into the computer is tagged with context keys facilitating future retrieval [48].

The key-value modelling approach is frequently used in distributed service frameworks (e.g. Capeus[44]).

The services themselves are usually described with a list of simple attributes in a key-value manner, and the employed service discovery procedure (e.g. SLP, Jini[33]).

2.1.2.2 Mark-up scheme models

All mark-up based models use a hierarchical data structure consisting of mark-up tags with attributes and content. Profiles represent typical mark-up-scheme models. Typical examples for such profiles are the Composite Capabilities/Preference Profile (CC/PP) [53] and User Agent Profile (UAProf) [55], which are encoded in RDF/S. It has been a largely experimented approach, Capra et al used an mark-up approach, in their scenario, middleware defines the grammar, that is the rules that must be followed to write profiles, in an XML Schema; the application designer then encodes the profile in an XML document that is a valid instance of the grammar. Every change done later to the profile must respect the grammar, and this check can be easily performed using available XML parsers[12].

Held[23] assessed several requirements to a comprehensive context representation. His requirements were based on the argument "Context information is gathered, stored, and interpret at different parts of the system. A representation of the context information should be applicable throughout the whole process of gathering, transferring, storing, and interpreting of context information". Many of these requirements were already mentioned on the beginning of this section, structure, interchangeability, composability/decomposability, being uniform, extensible and standardised.

The Centaurus project[29] provides an infrastructure and communication protocol for providing 'smart' services to mobile devices. It uses XML for defining ontologies and describing properties and interfaces of Services. As this is already being widely used, their approach was belived to be helpful in integrating Centaurus with already existing systems. The information flowing in the system is strictly in the form of CCML (Centaurus Capability Mark-up Language) which is built on top of XML.

The *stick-e* project[10] in term found a solution for portability of information using SGML, a mark-up language based on which XML was developed.

In this project a document is an ordered set of SGML-encoded notes. Two approaches have been explored in order to represent the document. In the first one the notes were stored end-to-end within a single file (with a starting and matching end tag). The second approach stored a note per file, being the main document a simple list of file names, representing the notes that encompass it, thus a single copy of a note being easily shared between several different documents.

This discussion is important because presents us with a trade-off between the two points of view, extra flexibility versus extra complexity [11]

Other examples exist beside the ones presented in this section [52][16][42][27].

2.1.2.3 Graphical models

The Unified Modeling Language (UML) can be used to specify information related to the design of context-aware services, it is suitable for modelling context. Various approaches exist where contextual aspects are modelled by using UML.

Sheng and Benatallah[50] defined a language for the model-driven development of context-aware Web services based on the Unified Modelling Language.

Another modelling approach by Henricksen et al.[24] was based on two factors, firstly the models of context currently in use in context-aware systems typically lack formality and expressiveness, and are primarily concerned with representing context in a form suitable for applications to query, secondly commonly used conceptual modelling approaches, such as UML and ER (entity-relation model) are not well suited to capture special features of context information. They proposed an extension to the Object-Role Modelling (ORM) in order to represent and design context and context-aware systems.

2.1.2.4 Object-oriented models

Modelling context by using object-oriented techniques offers to use the full power of object orientation (e.g., encapsulation, re-usability, inheritance). Existing approaches use various objects to represent different context types (such as temperature, location, etc.), and encapsulate the details of context processing and representation.

Access to the information and the context processing logic is provided by well-defined interfaces. Hydrogen [25] uses such an object-oriented example.

2.1.2.5 Logic-based models

Logic-based models have a higher degree of formalism. Typically, facts, expressions and rules are used to define a context model. A logic based system is then used to manage the aforementioned terms and allows to add, update or remove new facts.

The inference (also called reasoning) process can be used to derive new facts based on existing rules in the systems. The contextual information needs to be represented in a formal way as facts.

One of the first approaches was published by McCarthy and Bucca [34].

2.1.2.6 Ontology-based models

Ontologies represent a description of the concepts and relationships. Therefore, ontologies are a very promising instrument for modeling contextual information due to their high and formal expressiveness and the possibilities for applying ontology reasoning techniques. Various context-aware frameworks use ontologies as underlying context models, such as OpenSocial(not confirmed).

Öztürk and Aamodt's starting point for a context ontology[3] was that problem solving is a deliberate process in which two basic elements are the agent and the external situation in which the problem solving occurs, ontologies were the best approach to this matter.

2.1.2.7 Analysis

Key-value models are clearly limited, in representation, influencing information richness and quality, failing to address ambiguity, requiring that each system knows about the data structure and how to use it. Partial validation should be highly desirable[52].

As for mark-up scheme models, the partial validation represents a strong concern. Validation is possible through schemas and validation tools, such as XSD or DTD, but disambiguation of context information must be done at application level. A comprehensive scheme definition is a step towards a high level formalism and thus may be used to determine interoperability[52].

The graphical approach comprehends the structure of the contextual information, facilitating applicability, although merging different models represents a limitation to some extent. Such models are mainly used for human interpretation of the problem.

Object-oriented modelling approaches are capable of a high interoperability, and inherent to the Object-Oriented paradigm, extension can be handled in a linear way.

Logic based models are very difficult to maintain, due to their extremely high level of formalism and applicability can be considered as a major drawback since in ubiquitous computing, logic reasoners are usually not available.

Ontologies can address most of interoperability issues. Validation (partial or total), formality, unambiguity and applicability make ontologies the most promising context representation technique.

2.1.2.8 Proposed representation

It is undeniable that today's approaches of interoperability are mainly based on standards (HTML, XML, etc.) and thus we aim to use such a standard definition. Among the

most promising alternatives, ontology models and markup scheme models, there are well-established standards such as OWL[2] and PIDF[3].

From the existing context representation techniques (see 2.1.2) markup models and ontology models can represent information in a structured, well defined manner, which positively collides with our aim. Representation of context must be directly connected to the environment where the context-aware system will act. Not all representations can be computed in any device, logic-based representations for instance require a high debit (compared to a cellphone) processing capability, hence the strong connection between representation and environment. Ubiquitous computing environments (see chapter 1) may contain devices with rather limited resources (computing power, memory) therefore a light processing method must be used.

Ontologies require the ability to understand the meaning of relations between entities. The application (wherever it resides) must be capable of interpreting this representation. Markup scheme model fulfils our needs to represent and share context maintaining a degree of formality capable of assuring integrity and relieving possible resource-constrained devices from additional processing actions. Markup scheme models do required validation (achievable through schemas) and reasoning at application level, although a limitation in situations where computing power is an issue, in ubiquitous computing environments such disassociation between information gathering and information reasoning might be desirable.

Being context highly dependent on the given interaction (see 2.1.1.3) extensibility must also be assured. Therefore, RPID[4] an extension to PIDF provides us with a standard representation of context/presence information which can be easily created, validated, shared and extended.

2.1.3 Context sensing

Context sensing is a delicate subject. Context information can be defined in two different types[20]: explicit information, obtained through direct observation or behaviour analysis and explicit user definition, or implicit information, inferred from known information, or perceived through indirect behaviour.

A direct mapping between the human capability of sensing context and the computer capabilities (see 2.1.1) is hardly simple. Explicit context can be somehow obtained, for instance, using sensors. A person uses his vision/hearing to determine the location of another person. A computer can, through a GPS device, determine the same information. When it comes to implicit information, mood for instance, there is a completely different level of complexity. A person might be able to perceive another person's mood through his/her facial expression, blushing or general posture. A computer might be able to use face recognition techniques and biometrical sensors to obtain such information, but still, such variables change from person to person and are not always usable.

[2]OWL Web Ontology Language: http://www.w3.org/TR/owl-features/
[3]Presence Information Data Format - RFC 3863: http://www.ietf.org/rfc/rfc3863.txt
[4]Rich Presence Extensions to the Presence Information Data Format - RFC 4480: http://www.ietf.org/rfc/rfc4480.txt

Implicit context should be user-defined as much as possible, or liable for user revision. Chen presents three approaches on how to acquire context information on computer science [14].

- **Direct sensor access**. This approach is often used in devices with sensors locally built in [30]. The client software gathers the desired information directly from these sensors, i.e., there is no additional layer for gaining and processing sensor data. Drivers for the sensors are hard-wired into the application, so this tightly coupled method is usable only in rare cases. Therefore, it is not suited for distributed systems due to its direct access nature which lacks a component capable of managing multiple concurrent sensor accesses.

- **Middleware infrastructure**. Modern software design uses methods of encapsulation to separate e.g., business logic and graphical user interfaces[19]. The middleware based approach introduces a layered architecture to context-aware systems with the intention of hiding low-level sensing details. Compared to direct sensor access this technique eases extensibility since the client code has not be modified anymore and it simplifies the reusability of hardware dependent sensing code due to the strict encapsulation.

- **Context server**. The next logical step is to permit multiple clients access to remote context sources. This distributed approach extends the middleware based architecture by introducing a managing remote component. Gathering sensor data is moved to this so-called context server to facilitate concurrent multiple access. Besides the reuse of sensors, the usage of a context server has the advantage of relieving clients of resource intensive operations. As probably the majority of end devices used in context-aware systems are mobile gadgets with limitations in computation power, disk space etc., this is an important aspect. In return one has to consider about appropriate protocols, network performance, quality-of-service parameters etc., when designing a context-aware system based on client-server architecture.

When analysing sensors themselves we will most commonly think about little electronic gadgets, like movement detectors or light sensors. Some sensors however might be virtual. A simple application that monitors a computer's network connectivity is also a sensor. We must then analyse how can sensors be divided and understand which information can be gathered from each type. Concerning the way data is captured, sensors can be classified in three groups[28, 5]:

- **Physical sensors**. The most frequently used type of sensors are physical sensors. Many hardware sensors are available nowadays which are capable of capturing almost any physical data.

- **Virtual sensors**. Virtual sensors source context data from software applications or services. For example, it is possible to determine an employee's location not only by using tracking systems (physical sensors) but also by a virtual sensor, e.g., by browsing an electronic calendar, a travel-booking system, emails etc., for location information.

Other context attributes that can be sensed by virtual sensors include, e.g., the user's activity by checking for mouse-movement and keyboard input.

- **Logical sensors**. These sensors make use of a couple of information sources, and combine physical and virtual sensors with additional information from databases or various other sources in order to solve higher tasks. For example, a logical sensor can be constructed to detect an employee's current position by analysing logins at desktop PCs and a database mapping of devices to location information.

Context-data acquisition is very important, since it predefines architectural parameters of the system, at least to some extent [14]. Our work will focus on middleware infrastructures and context server throw multiple sensor types.

Another concern we must address is the pre-processing of context-data. We will address this matter on following chapter, context reasoning.

2.1.4 Context reasoning

Useful information is the basis of a good decision. We have analysed different approaches to gathering data (see section 2.1.3) but in order to achieve our goal we must consider how raw the data is. Furthermore several conclusions might be inferred from lower level data. Data obtained through a location sensor (GPS for instance) might return the coordinates of the target. Although correct such information may be way too low-level for application direct usage, which in most cases wants to know the room/street the target is in. Other situation relies on assuming that Bob is taking shower analysing his location (bathroom), water heater status (on) and door status (closed).

An extra level of information retrieval must then be considered before reasoning about the information itself. *Pre-processing* of contextual information applies aggregation/composition of lower level context, hence raising the results of the sensor layer to a higher abstraction level information [5].

This "extra" layer is often used in context servers rather than in applications themselves, since this encapsulation advances the reusability and, therefore, eases the development of client applications.

When a computer system knows something (often a lot) about a user, it might have enough information to take actions on behalf of that user. When should an application take such actions? We will analyse different approaches to reasoning with different levels of "intrusion".

2.1.4.1 Passive context-awareness

There are genuine risks involved in allowing the system to take the initiative in any activity in which human participants are involved [40]. As we see, there is simply too much variability as to what should be done, given different conditions, for designers to successfully model appropriate outcomes in advance. This may not be problematic when the output of the application simply represents the location of a sensed object. Things get much more

complicated when applications take it upon themselves to interpret human activity and "do things" for people by invoking services[9].

Bellotti et al.[7] claim that user intervention must always be required in order to a context-aware system to be accepted[9]. Through a series of experiments, Barkhuus et al. studied human reactions facing different context-aware systems.

Passive context-awareness addresses Bellotti's human concerns, allowing applications to directly gather and reason about contextual information but, rather that automatically and dynamically taking an action[20], they simply suggest that action.

Our approach must address such human concerns, although we aim for a deeper automation.

2.1.4.2 Active context-awareness

Gu et al.[20] on the Service-Oriented Context-Aware Middleware (SOCAM) consider context-aware services as agents (hopefully through standards, such the ones defined by FIPA[5], and/or adequate languages, such as KQML[6]), applications and services that make use of different levels of context in order to adapt the way they behave according to the current context. This process does not take into account direct user intervention aiming for highly adaptable services during their life-cycle.

It is undeniable the diminishing of user control through this approach and users do feel that decrease, but still, most users prefer computers to take action on their behalf, after a suitable learning period adequate to the application purpose[7].

In our work we intend to composite passive and active context-awareness, providing services capable of automatically taking actions but based on user defined rules.

2.1.4.3 User-defined context-awareness

Another possibility of reasoning about contextual information is simply based on learning patterns.

After a learning period the system might know enough in order to provide the user with relevant information. Google Inc., the famous search engine, analysis search patterns and provide advertisement links accordingly. Analogously, Kawsar et al. use everyday objects, which maintain their original capabilities, to obtain the users context. Based on user predefined information (profession, interests, etc.) the *AwareMirror* can provide to the user relevant information (news, traffic conditions) on an usual object (mirror) while he/she is brushing his/her teeth (toothbrush is moving)[30].

Common social group's applications, such as MSN®[7], empower users with explicit context definition capabilities. Based on this attributions the application adapts (though slightly) its behaviour: a user's context of "Busy" avoids sound playing from the application.

[5] Foundation for Intelligent Physical Agents: www.fipa.org
[6] Knowledge Query and Manipulation Language: www.cs.umbc.edu/kqml/kqmlspec/spec.html
[7] Microsoft® Network

IRC[8] allows for the definition of "away messages", providing additional information to possible incoming communications.

2.1.4.4 Infrastructures

To realize new applications for new computing technologies there is a need for supporting software infrastructure to catalyse their development and operations[38].

Much of the earliest work focused on location transparency, i.e. seamlessly providing the same communications, computation, and resources that the user is offered on their desktop computing environments. Noble et al. [46] suggest a more realistic proposal, called application-aware adaptation, and a concrete implementation of their ideas in the Odyssey API, an extension to the UNIX operating system. The aim of this system was to relieve the user from explicit intervention due to communication limitations. For example, a video player application may adjust the frame rate, image size, number of colours, etc. of the video stream to make the use of the available computing and communication resources.

This ability to adapt the behaviour in runtime leads us neatly on to software infrastructure for context-awareness, where it is this very adaptation and reaction to the environment of use that is the center of interest. First tries where made by Schilit el al. [48] achieving a framework which served as basis for many applications, namely location centric yellow pages and location-based file browser.

Gu et al.[20] claim a much more abstract approach, arguing that a common context model (see 2.1.2) should be the basis for a context-aware system, since context-sources range from a wide variety of possibilities (see 2.1.3). As stated by Strand and Linnhoff-Popien [52], ontology's properties address dissemination and interpretation of context, being the advised, though not compulsory, suggestion of Gu et al.. Their solution, on an other level, consists of a set of services that perform context acquisition, discovery, interpretation and dissemination.

A main distinction between authors reside on the location of the several components. Dey and Sohn [19] try a decoupled solution, leaving context gathering and interpretation to pervasive devices themselves, though requiring a centralized location server.

SOCAM framework approach requires a centralized context server. This type of servers relieve less powerful devices from additional (pre)processing, but single-point-of-failure issues might arise.

2.1.5 Discussion

The definition of *context* is cause for discussion. Analysing the various aspects of context and situation where context plays an important role, we were able to understand the relevance of certain information relative to the situation in hands. Not all information is context in every situation.

[8] Internet Relay Chat

	Complexity	Effort required to extend	Interoperability	Computer Understandable
Key-Value	Low	High	None	Yes
Mark-up	Median	Low	Total	Yes
Graphical	Median	Low	Total	No
Object-Oriented	Median	Low	Partial	Yes
Logic-based	Very High	Very High	Unknown	Yes
Ontology	High	Median	Total	Yes

Table 2.2: Context Representation models analysis

	Key-value	Mark-up	Graphical Graphical	Object Oriented	Logic-based	Ontology Ontology
Bucca et al.[34]					X	
Capeus[44]	X					
Centaurus[29]		X				
Eclipse (plugin) [1]			X			
Hydrogen[25]				X		
Jini[33]	X					
OpenSocial [2]						X
SLP[33]	X					
StarUml [3]			X			
Stick-e[10]		X				

Table 2.3: Context Representation models and client applications analysis

Similarly, context representation has several different possible approaches (see table 2.2), each of which with different degrees of complexity and richness. Key-value pair representation, although very limited when representing context, is the most simple representation technique, enough for specific applications. Mark-up models represent the most extensible and standardised approaches to the representation of context information. Graphical models although easily extensible are complex to map into a computer understandable language. Object-Oriented models are dependent on the implementation language in order to achieve interoperability. Logic-based models are too complex, requiring specialised qualifications to develop, being also hard to maintain and extend. Finally, ontologies are very similar to mark-up models, the only difference being an additional overhead when extending, mainly due to restrictions validation.

Reasoning based on context must be ruled by privacy policies. There are several different levels of "intrusion" when speaking of context-aware applications. We analysed this levels and understood their purpose, based on the application finality.

2.2 Social Groups

The definition of group can be stated as two or more humans interacting with each other and having one or more aspects in common (expectations, obligations, identity, etc.).

However a true group exhibits some degree of social cohesion and is more than a simple collection or aggregation of individuals[21]. Characteristics shared by members of a group may include interests, affiliation, hobbies, etc.

A **social group** as we see it in the project environment is precisely a group with characteristics in common that communicate with one another through virtual means.

Why are social groups so important? Most people have used or do use social groups frequently. It is visible that the growth of these networks is not slowing at all, meaning that new users are connecting through this means everyday.

On a company's perspective it is important to consider the Metcalfe's law[9] which states that the value of a telecommunications network is proportional to the square of the number of connected users. Interacting among them consequently adds value to the system.

Other perspective is related to the richness of the information that such groups can provide, both in terms of address information (since every user on these networks have an endpoint from where they communicate) as well as context information, due to the fact that most social groups have a built-in presence system[10] , which provides availability information (context) to the project's system. It is important however to mention that this presence system is not completely trustworthy, since a user's given presence might not accurately describe the user's real presence.

We will now analyse social groups architecture, having in mind the pull-based and push-based models, followed by a closer over the most common protocols used. Information acquisition will be addressed too, followed by an analysis of real world examples, GTalk, Sapo Messenger and MSN, ending with an overhaul discussion.

2.2.1 Architecture

Most social groups such as MSN®and GTalk®follow the same basic architecture (see picture 2.2) . First of all there is an authentication system, imperative to accommodate multiple users as well as to maintain confidentiality.

A preamble phase often occurs to determine protocol versions, presence state, desired name aliases and so forth. After the mentioned preamble the contact list of the owner is obtain from the server and each entry on that list is contacted, either using a pull-based or a push-based model, in order to obtain the most up-to-date information regarding said entries. This information ranges from presence information, aliases changes, personal data changes etc.

Pull-based model

The pull-based model consists of a client periodically contacting the information provider for new, up-dated information. This model relieves the information source from managing

[9]Metcalfe's Law: en.wikipedia.org/wiki/Metcalfe's$_{l}aw$
[10]MSN: http://msnpiki.msnfanatic.com/index.php/MSN$_{P}rotocol_{V}ersion_{8}$

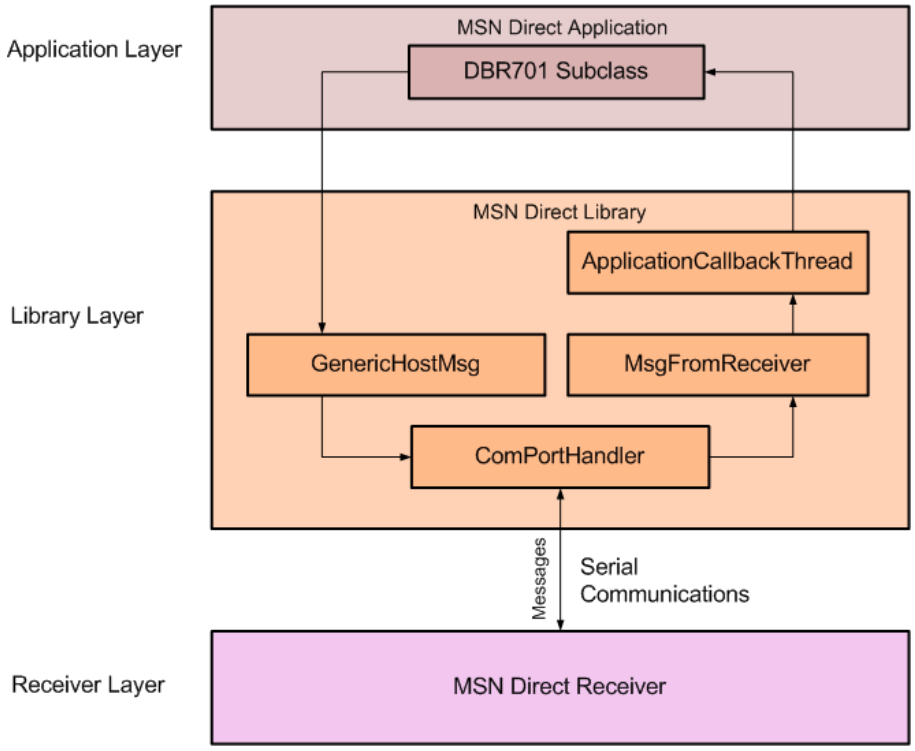

Figure 2.2: MSN Messenger Architecture

entities which are following its state changes, however this approaches brings forward an additional network overhead.

Push-based model

The push-based model is roughly the opposite of the pull-based model. A client only registers itself in the information source. The latter is responsible to contact each client (listener) when a state change occurs, usually asynchronously.

2.2.2 Protocols

Although there are opensource well-defined protocols for social group communication not all providers use the same protocol. Commercial reasons or performance factors might cause such heterogeneity, it is imperative however to understand the most common protocols features.

We will analyse some different protocols based on the services that use them, such as Microsoft®MSN Messenger, Google's GTalk and Sapo Messenger.

MSN

Microsoft Network®(MSN) uses a proprietary protocol. Their protocol, MSNP (Microsoft Notification Protocol) has been revised several times over the years and contains all basic commands required to initiate and manage an instant conversation and presence information.

MSNP, as mentioned, is a closed protocol and therefore all information available about it was obtain through reverse-engineering[11].

GTalk

Google Talk®(GTalk) uses a standard open-source protocol known as Extensible Messaging and Presence Protocol (XMPP) which allows applications to manage instant messaging, presence, multi-party chats, voice and video calls, collaboration and many more [12] [13] [14] [15] [16] [17] [18].

Sapo Messenger®uses XMPP as well.

2.2.3 Information Acquisition

Social groups to some extent can be seen as endpoints of communication. A person using such service has a new way of communicate and receive communications. This presents an interesting addition to the system, a new, intensively used and quite reliable way of reaching a person. This however represents only the address side of the conversation.

Presence is, by definition, a form of context (see 2.1.1) which could be and is used by our system to determine not only where the user is present (in terms of address) but also where is he/she available, can he/she be disturbed or should alternative ways of reaching him/her be assessed.

When we described the basic social group architecture we mentioned "contact list". Any person who is subscribed to a social group service has indeed a contact list which contains every person that the user wants to be aware of. This represents another valuable feature of social groups, being able to discover endpoints of communication regarding people that might not even know of the existence of this project's system and even acquire presence information about them.

This asset allows the system to work with entities (see section 4.3.4) regardless of their awareness of our system, the only requirement being the person who initiates the desired action to have enough privileges on our system.

2.2.4 Real world examples

It is undeniable the quantity of social group's services that exist today (AIM, IRQ, MSN, GTalk, MSN, IRC, etc.), being evident that we couldn't interact with every single one

[11] MSN: http://msnpiki.msnfanatic.com/index.php/MSN$_{P}rotocol_{V}ersion_{8}$
[12] XMPP Core - RFC 3920: http://www.ietf.org/rfc/rfc3920.txt
[13] XMPP IM - RFC 3921: http://www.ietf.org/rfc/rfc3921.txt
[14] XMPP CPIM - RFC 3922: http://www.ietf.org/rfc/rfc3922.txt
[15] XMPP E2E - RFC 3923: http://www.ietf.org/rfc/rfc3923.txt
[16] XMPP URN - RFC 4854: http://www.ietf.org/rfc/rfc4854.txt
[17] XMPP ENUM - RFC 4979: http://www.ietf.org/rfc/rfc4979.txt
[18] XMPP URI - RFC 5122: http://www.ietf.org/rfc/rfc5122.txt

(due to time and relevance restrictions), however one service would not be enough either. Extensibility and scalability are two very important conditionals that rule our system. This conditionals were indeed met and will be discussed further on.

2.2.4.1 GTalk

GTalk®is Google's approach to instant messaging. one of the main features about this service is the capability of interacting with it through several different means, from web-based applets to standalone desktop applications, cellphone widgets or even email messages.

Since it uses XMPP (see section 2.2.2) it is even possible for a person to develop their own application to interact with the service.

2.2.4.2 Sapo Messenger

Like GTalk®, Sapo Messenger®uses the XMPP protocol, meaning that it is not required to use the Sapo Messenger application to interact with the service.

One interesting feature however about their application is the capability of using one application to interact with both Sapo Messenger and MSN.

2.2.4.3 MSN

MSN®is without a doubt is the most used social group and instant messaging service. Although as mentioned above their protocol is proprietary, there are ways of interacting with this service that we explored.

2.2.5 Discussion

Social groups are roughly based on the same architecture. Although protocols might vary from service to service, the set of information provided by this services is the same: address information (communication endpoint), and context information (presence).

Address information is not as rich as a address book analogous information, it provides however a communication end-point to the user. On the other hand, context provides (with its associated confidence) a new source of information regarding the user's availability or even activity.

All analysed social group services (MSN, GTalk and Sapo Messenger) represent real world networks. We have assessed the protocols that they use along side the importance that each one represents to this project.

As we have analysed, Metacalf's law proves the value of such systems upon the present project.

2.3 Extract, Transform, and Load (ETL) Information

Information integration is today one of the most important problems of information technology. This is caused both by the abundance of available information inside an organization through ever-improving network capabilities and from outside organizations through Web services, portals etc., and by the increasing awareness of added value through information integration. The evidence for the latter reason is seen in data warehousing activities, installation of intra-organizational Web services etc [35].

Information integration projects and products have in common the use of some form of mapping from one or more data source schemata to a target schema [51]. Merged information should however be governed by two main metrics, **completeness** and **correctness** [35].

Completeness

Completeness of merged data is desirable to ensure that no data or relationship among data is ignored, when merging it from multiple sources. That is, all data stored about an object in all sources should be considered in some form in the target, as long as the target schema can store this data.

For instance, two sources reporting the same author of a book, identified by its ISBN, enforce that this person is indeed the author of that book. Sources *complement* each other if they store data about different attributes of the same object. For instance, a source storing only the publisher of a book complements a source storing only the number of pages of the same book. Sources *conflict* if they store different values for the same attribute of the same tuple. For instance, a source storing 205 as the number of pages of a book conflicts a source storing 150 as the number of pages of the same book.

Ensuring completeness is, in effect, the problem of aggregating attribute values from different tuples into a final value that appears in the merged tuple [35].

Correctness

Correctness of merged data is desirable to ensure that the combined data adheres to (possibly implicit) primary key constraints in the target database. That is, even though the data sources store multiple tuples about the same real world object, only one tuple about this object should appear in the target. To this end, one must recognize multiple tuples about the same real world object and assign a common ID. If there is a globally consistent ID, such as the URL or social security number, that is used and provided by all sources, this task is simple: Tuples with the same ID represent the same object and can be merged. In the absence of such an ID, object identification techniques can be employed [31].

These techniques find duplicates automatically by evaluating the data that is available.

Data sources can overlap in two dimensions: extensionally and intentionally [35]. The extensional overlap between two sources is the set of real world objects that are represented in both sources. For instance, multiple sources can store information about the same book. The intentional overlap between two sources is the set of attributes both sources provide. For any given book, different sources can store the same information about it, such as title and author.

To accommodate both types of overlap and to integrate data in a meaningful and useful way, identical objects represented in different sources must be recognised (object identification), and any data conflicts among values must be resolved (conflict resolution).

2.3.1 Object Identification

Merging data from different sources requires that different representations of the same real world object be identified as such [31]. This process is called object identification. Object identification is difficult, because the available knowledge about the objects under consideration may be incomplete, inconsistent, and sparse. A particular problem occurs if no natural identifiers (IDs) exist. For instance, the URL of a Web page is a natural ID for the page. A meta-search engine can use the URL of reported hits to find and integrate duplicates. On the other hand, a used car typically has no natural ID other than the registered license plate for instance and sources about used cars do not store an ID. An integrated information system for used cars has no easy way of identifying a specific car being advertised in different data sources.

Object identification in the absence of IDs, which is essentially the same problem as duplicate detection, record linkage, or object fusion [37][36], is typically approached by statistical methods. Our approach is no different.

For simplicity purposes we hereby assume that the following statistical methods, as described next, are applied to a small set of attributes, such as *name* and *local uri* (Local Uniform Resource Identifier).

Levenshtein Distance Algorithm

Levenshtein Distance Algorithm is a metric for measuring the amount of difference between two sequences of characters (i.e., the so called edit distance) [32]. By *amount of difference* we understand the number of operations (inserting, changing or removing a letter) in order for the first string to become equal to the second string.

Considering two strings, ***plate*** and ***plot***. The first step is to determine the length of each string, in this example the former has length of five whilst the latter has length of four. This represents the dimensions of the result matrix, 5 by 4.

The main logic of the algorithm is as follows, for each letter in the first string (i repre-

		p	**l**	**a**	**t**	**e**
	0	1	2	3	4	5
p	1	**0**	1	2	3	4
l	2	1	**0**	1	2	3
o	3	2	1	**1**	2	3
t	4	3	2	2	**1**	**2**

Table 2.4: Levenshtein algorithm at work

senting the index of the letter we are analysing at some point) and for each letter of the second string (j representing the index of the letter we are analysing at some point): if both letters are equal then the operation cost equals zero, otherwise it equals to one (or the custom cost for substituting a letter for another one).

In the result matrix, the position (i, j) will contain the minimum value between three possibilities:

- Adding the cost of an insertion operation to the value present in position (i - 1, j) of the result matrix
- Adding the cost of a deletion operation to the value present in position (i, j - 1) of the result matrix
- Adding the cost calculated earlier to the value present in position (i - 1, j - 1) of the result matrix

Following these steps will fill the result matrix with several values which represent all the possibilities for editing a string to obtain the target one. Applying this algorithm to our example (table 2.4) the distance between ***plate*** and ***plot*** is two.

Since we assumed that every operation had the same cost, this result means that two operations were necessary for the word *plot* to become *plate*. In table 2.4 is an highlighted "path" that emphasises these operations. The letter "o" in "plot" must be substituted with the letter "a", and finally we must insert the letter "e" at the end of *plot*.

Although this algorithm does not use any remarkable sophistication it can be used quite reliably by any system. This raw approach has a space requirement of m x n, being m the length of the first string and n the length of the second one.

The usefulness of this technique is achieved when used to detect possible duplicate entries. Although several approaches are possible, a simple one would consist in defining an **edition threshold** (for instance 4). Everytime such a threshold were respected, the two strings involved would be marked or signalled (see section 2.1.4) as referring to the same entity.

Soundex Algorithm

Soundex Algorithm[19] is a phonetic algorithm for indexing names by sound. The goal is for homophones to be encoded to the same representation so that they can be matched despite minor differences in spelling.

[19]Soundex Algorithm: http://www.archives.gov/genealogy/census/soundex.html

b	voiced	bilabial	plosive
p	voiceless	bilabial	plosive
f	voiceless	labiodental	fricative
v	voiced	labiodental	fricative
c	voiceless	palatal	plosive
j		palatal	approximation
g	voiced	velar	plosive
k	voiceless	velar	plosive
x	voiceless	velar	fricative
q	voiceless	uvular	plosive
s	voiceless	alveolar	fricative
z	voiceless	alveolar	fricative
d	voiced	alveolar	plosive
t	voiceless	alveolar	plosive
l	alveolar	lateral	approximation
m	voiced	bilabial	nasal
n	voiced	labiodental	nasal

Table 2.5: Consonants list by pronunciation

The inner workings of this algorithm are rather simple, similarly to the Levenshtein distance algorithm mentioned earlier.

The Soundex code for a name consists of a letter followed by three numerical digits: the letter is the first letter of the name, and the digits encode the remaining consonants. Similar sounding consonants share the same digit so, for example, the labial consonants B, F, P, and V (see table 2.5) are each encoded as the number 1. Vowels can affect the coding, but are not coded themselves except as the first letter. However if "h" or "w" separate two consonants that have the same soundex code, the consonant to the right of the vowel is not coded.

Summarising the stages of the algorithm, we have:

1. If "h", "w" separate two consonants with the same soundex code, change consonants to right of the vowel into "h" until they have the same soundex code.
2. Replace consonants with digits as follows (but do not change the first letter):
   - b, f, p and v become 1
   - c, g, j, k, q, s, x and z become 2
   - d and t become 3
   - l becomes 4
   - m and n become 5
   - r becomes 6

3. Collapse adjacent identical digits into a single digit of that value.
4. Remove all non-digits after the first letter.
5. Return the starting letter and the first three remaining digits. If needed, append zeroes to make it a letter and three digits.

"Robert" and "Rupert" both produce the soundex code "R163", while "Rubin" produces the soundex code "R150".

Both the present methods enable a system to be equipped with basic (although rather satisfactory) techniques to identify different objects referencing the same entity.

2.3.2 Conflict Resolution

Once different tuples have been identified as representing the same real world object, the data from them can be merged [35]. In general, a result that is integrated from tuples of different sources, contains tuples where:

1. The value for some attribute is not provided by any of the sources. Sources may not provide the value, because they do not store the particular attribute, or because they have stored a *null* value for the particular tuple. Because none of the sources provide a value, the tuple in the result has no value either (*null* value).
2. The value for some attribute is provided by exactly one source. In this case, there is also no actual data conflict. When constructing the result, the single attribute value can be used for the result tuple. If a missing value has the meaning "not applicable" instead of "unknown", the absence of data can be taken into account as well. For the remainder, we assume the "unknown" semantics for null values.
3. The value for some attribute is provided by more than one source. This case demands special attention, because several sources compete in filling the result tuple with an attribute value. If all sources provide the same value, that value can be used in the result. If the values differ, there is a data conflict and a resolution function must determine what value shall appear in the result table.

2.3.3 Rule-Based Systems

Roughly speaking a rule-based system automates problem-solving know-how, providing a means for capturing and refining human expertise [22]. Such a system enables an automation regarding conflict resolution or information merging as well.

Experts tend to express most of their problem-solving techniques in terms of a set of situation-action rules, and this suggests that rule-based systems should be the method of choice for building knowledge-intensive expert systems. Our system aims for the general computer-science related public, thus expertise should not be a requirement. However customization capabilities must be provided in order for such general users to change data perception as they see fit. This is impressively similar to expert systems based on rules.

Although many different techniques have emerged for organising collections of rules into automated experts, all rule-based systems share certain key properties [22]:

1. They incorporate practical human knowledge in conditional *if-then* rules
2. Their skill increases at a rate proportional to the enlargement of their knowledge base
3. They can solve a wide range of possibly complex problems by selecting relevant rules and then combining the results in appropriate ways
4. They adaptively determine the best sequence of rules to execute
5. They explain their conclusions by retracing their actual lines of reasoning and translating the logic of each rule employed into natural language

Hayes-Roth has also determined a set of areas where these systems and the knowledge contained within could be most useful, including:

1. Specific inferences that follow from specific observation
2. Abstraction, generalization and categorization of given data
3. Etc.

2.4 Analysis

Over this previous chapter, related work (see chapter 2) different aspects of context-awareness were addressed.

The definition of the concept "*context*" itself was the starting point of this chapter, where we analysed different approaches, starting with the common understanding of the concept (see section 2.1.1.1). We then evolved to the human perspective of context (see section 2.1.1.2), and how humans perceive and reason with context. The human perspective was then mapped to computer science and the computer perspective of this information (see section 2.1.1.3) leading us to the first definitions of context. Our definition itself is based on a previous definition (see section 2.1.1.3 and 2.1.1.3).

Having a definition of what context is and means, we addressed the information representation suitable to store it (see section 2.1.2). Different approaches were analysed, namely key-value models, mark-up models, graphical and object-oriented models, logic-based models and ontology ones.

After this analysis we assessed the more suitable representation to the project in hands (see section 2.1.2.7).

Context sensing was briefly studied (see section 2.1.3) where we analysed different architectural approaches possible to context-aware applications and scenarios where they are more effective.

Context reasoning covered the client-side concerns regarding an automated system taking actions of behalf of a client (see section 2.1.4). Context-aware applications may have different levels of pro-activity, namely passive context-awareness, where they roughly act as a warning system, leaving the ultimate action to the client; active context-awareness, where the application automatically and autonomously takes an action on behalf of the client; and user-defined context, where the client himself/herself defines the context information associated with him/her.

Social groups, as an important part of this project, were analysed on different levels. Their architecture (see section 2.2.1) and protocols (see section 2.2.2), the information that can be acquired through these services (see section 2.2.3) and actual real-world services that interact with this project (see section 2.2.4).

Extract, Transform, and Load (ETL) Information addressed different areas of information treatment. Object identification, along side with duplicate detection was addressed (see section 2.3.1) were addressed with special emphasis on comparison algorithms such like Levenstein String Edition Distance and the Soundex algorithm.

Conflict resolution was another level of ETL addressed (see section 2.3.2) where we specially analysed situations within this project, where conflict might occur.

As an automated system for conflict resolution, rule-based systems were assessed (see section 2.3.3) based on scalability, adaptability and performance among other important dimensions.

Chapter 3

Funambol Background

As we discussed previously, social groups can be regarded also as communication endpoints. Gathering and aggregating information regarding endpoints can be very difficult, specially due to the fact that endpoints and contact information is usually stored using a myriad of formats, often not interoperable. The same occurs with our contacts. Most users have their friends' and acquaintances' addresses stored in several different systems: address book, cellphone, email contact list, just for a few.

Funambol is an open-source synchronization system that allows the synchronization of several different contact storage systems, propagating changes from one to another and to some extent merging information (including conflict resolution). As we can see through picture 3.1 Funambol's approach is to separate the platform (device) being synchronized (cellphone for example) from the system that executes the synchronization itself.

Each device has its own connector to the system, This connector is in charge of gathering all contacts from the device, converting those contacts into an understandable and interchangeable representation, communicate changes to those contacts to the server and, if necessary, exchange data. It is important to understand that the Funambol system does not synchronise solely contacts, it is capable of synchronizing calendars, file system directories and virtually anything a person can think of. Its abstraction layers allow for a generic treatment of the objects to be synchronized.

Such a system would be ideal to synchronise information in the context of the already mentioned heterogeneity of the systems that a person might use to store his/her friends contact information, allowing our system to use multiple platforms that provide relevant information.

Funambol is based on SyncML (Synchronization Markup Language), the Open Mobile Alliance (OMA) standard for Data Synchronization (DS) and Device Management (DM). Many leading device manufacturers now include a SyncML client in their new devices.

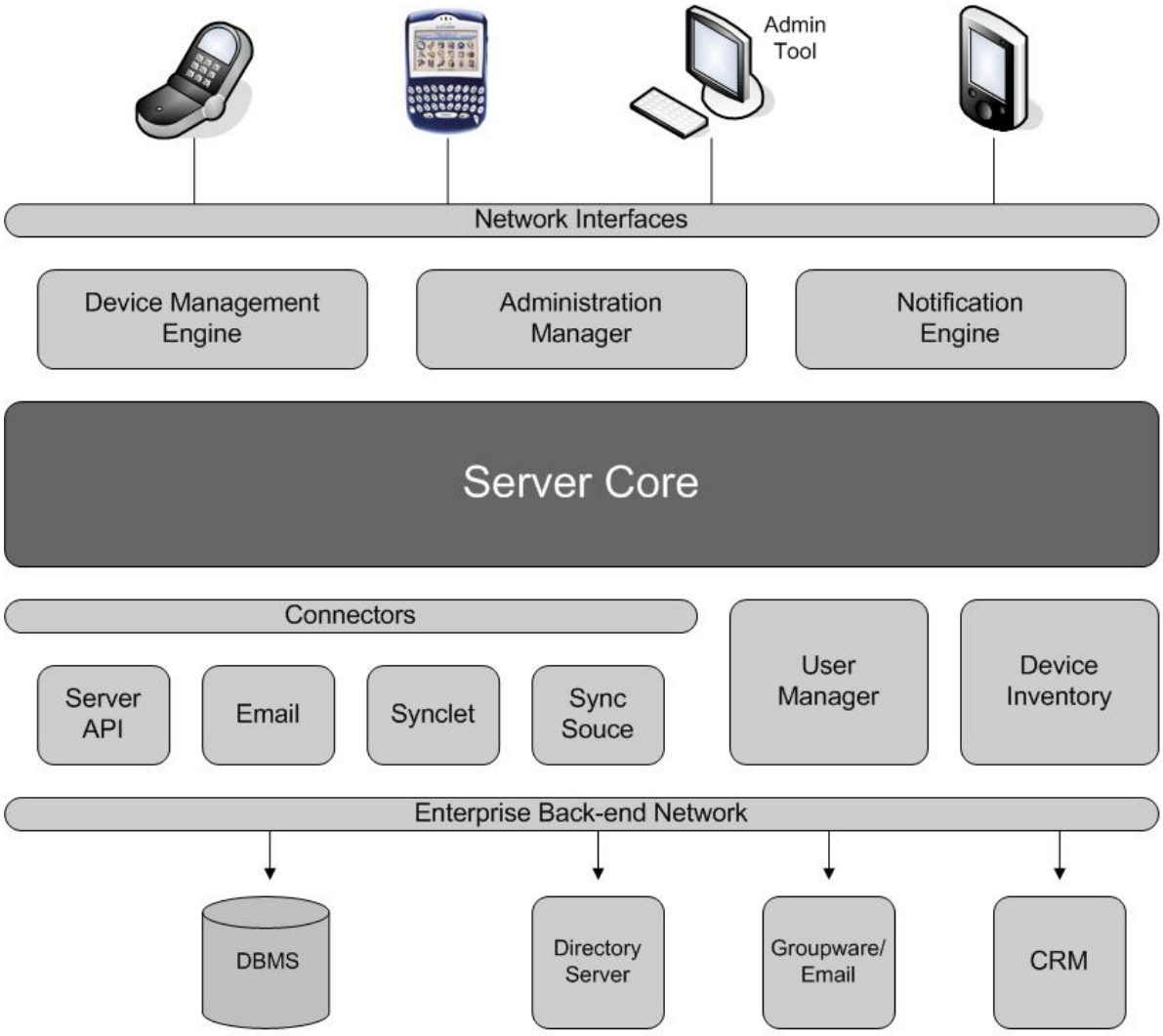

Figure 3.1: Funambol Architecture

3.1 Contact Format

When it comes to contact representation and storage there are several different possibilities. vCard for instance is a file format standard for electronic business cards. It defines how the contact information of a person should be stored along with its properties.

Some representations actually have their base in vCard, such as hCard, a microformat representation of 1:1 vCard in semantic (X)HTML or XML vCard, defined by XMPP Standards Foundation and used in protocols such as XMPP (see section 2.2.2).

Funambol is capable of dealing with these formats and several more, such as Microsoft Exchange Personal Information Management (PIM) data format.

3.2 System Internals

We will briefly introduce the synchronization protocol performed by Funambol (see picture 3.2).

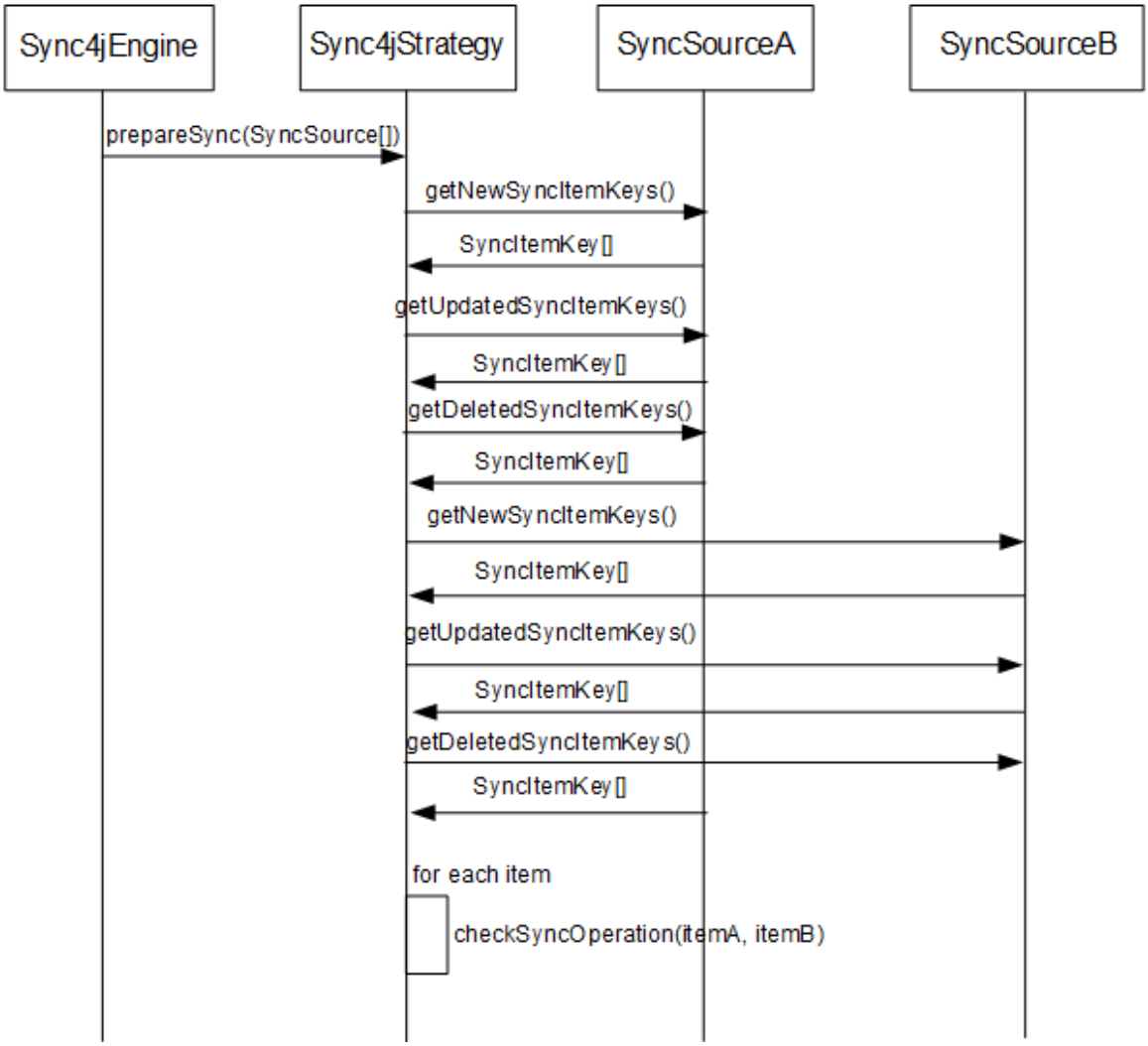

Figure 3.2: Synchronization sequence diagram

The process itself is divided in three phases:

1. Preparation
2. Synchronization
3. Finalization

Preparation

The preparation phase is the process of detecting modifications and analysing the differences between two or more databases (called SyncSources) obtaining a list of sync operations (add,

delete, update) that applied to the sources involved in the synchronization will make the databases look identical, i.e., they will converge to the same set of values.

Modification detection has several approaches[43], although the one used is based on the modified bits technique[39]. Given the information

- A - Items belonging to source A
- B - Items belonging to source B
- Am - Modified items belonging to source A
- Bm - Modified items belonging to source B
- AmBm Items modified either in source A and B (intersection between Am and Bm)
- (A-Am)Bm Items unmodified in A, but modified in B
- Am(B-Bm) items unmodified in B, but modified in A

Synchronization

The synchronization step is the phase where the sync operations prepared in the previous step are executed. Executing a SyncOperation means applying the required modification to the SyncSource involved.

Finalization

As for the finalization stage,the third and last step, it is intended for clean up purposes. Auxiliary data structures used to store temporary information are cleaned, consistency of the information is checked and the status code for each updated tuple is changed to the target status code.

3.3 Discussion

Some Funambol-specific decisions during this project will be addressed further on this dissertation (see section 5.2), however Funambol represents an example of a real-world system with multiple users, first considering the information it provides, covering a significant number of sources that can participate in it, secondly because this information is associated with real people. Along with social groups (see section 2.2) our system is being built over systems with commercial usage, therefore leveraging the commercial usage of our system as well.

Chapter 4

Architecture

We want our system to be scalable, enhancing the richness of information that can be used, achieving that by providing an easy way to extend information sources.

To provide a scalable infrastructure the system was thought as a modular application. Each module has its own responsibilities and implementation, providing however a public, well-defined, generic interface that allows external applications to access its information and functionality.

In our exposition, specific vocabulary must be considered:

- *Interfaces*: Particular virtual classes that define the methods (and their signatures) that any class implementing such interface must provide.
- *Components*: Programming logic which provides or consumes a set of services. Components are the modules of an higher level application. Although components represent modules, and implement a generic interface, their inner workings might require direct interaction with the "outside" environment, being responsible of transportation of any ingoing or outgoing package.

During our analysis we found that our system requires two types of information, one regarding context information, the other regarding address information.

Following this line of thought our system was basically divided in two parts, each one addressing each type of information.

The main component of our system, the *Group Enabler*, consumes contact and context sources in order to provide its own set of features. The number of each type's sources should not be limited, hence we used interfaces to catalyse scalability across the whole system.

As we can also see in figure 4.1, each external system has its own connector. This connector works as an adaptation layer between said system and our own. This approach allows for a standardized processing of information within our system.

We will begin by analysing the system interfaces (see section 4.1), their methods and the motivation behind those methods. We will then address the system components (see

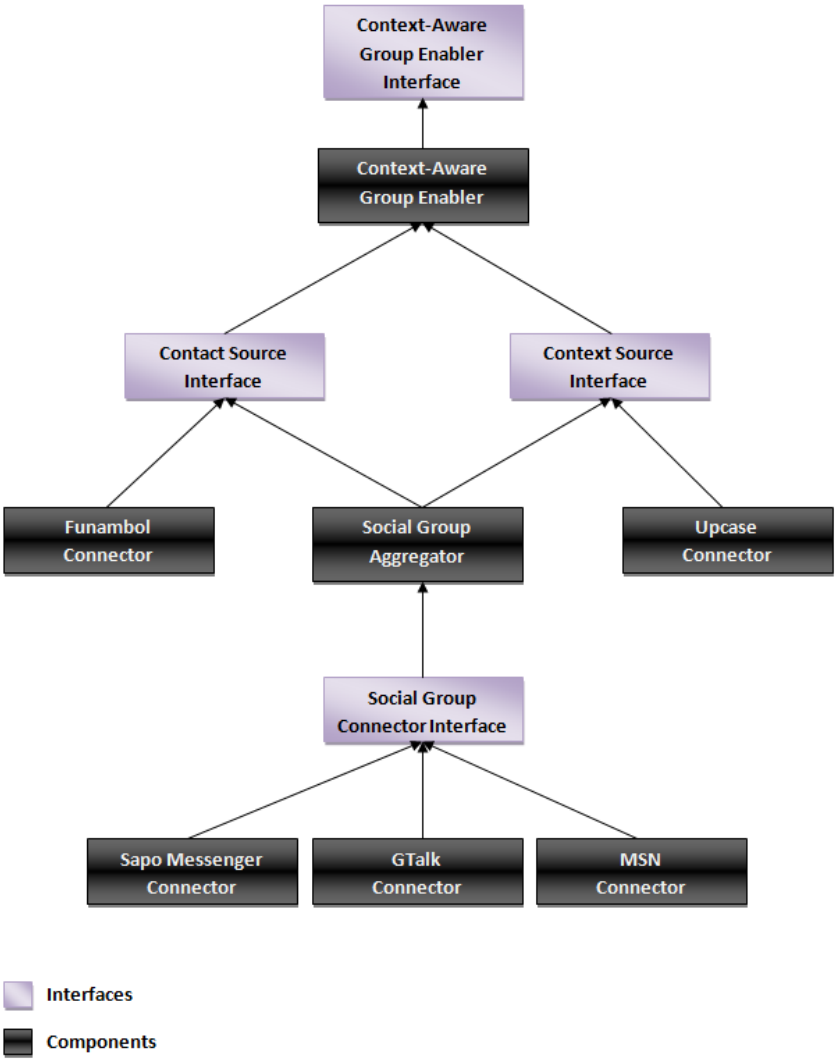

Figure 4.1: Solution's Architecture

section 4.2), which make use of the defined interfaces. Following this sections we will take a look over the information architecture of the project (see section 4.3), analysing the main information entities present in it. To finalise this chapter we will look over specific interfaces developed during the execution of this project which are closely related to external systems (see section 4.4).

4.1 Interfaces

As earlier described, interfaces define a well-known set of methods. This means that any application using a service which implements such interfaces knows precisely the methods available, regardless of the service implementation.

This approach allows multiple different implementations to produce a common interface to applications, hence stimulating scalability.

We will now take a closer look on the interfaces themselves: Social Group Connector Interface, Contact Source Interface, Context Source Interface and Context-Aware Group Enabler.

4.1.1 Social Group Connector Interface

This component represents a set of methods associated with the interaction with social group environments. Social groups will be discussed further on (see section 2.2), but it is important to notice that these external services often use their own protocols, either for exchanging information or to transport it.

This interface does not take into account any type of protocols that must be used, solely defining methods regarding the collection of information. The method signatures provided by this interface are as follow:

- *Get Connector Type*: Method to obtain the type of the connector that is being used. Since applications might not (and probably don't) know the type of the connector, only the type of the interface that it implements, it can be useful in some use-cases to have access to this information.
- *Get Contact List*: Provides a list of Social Group Entities (see section 4.3 for more details) that contains all information regarding contacts belonging to the owner's contact list.
- *Get Contact Context*: Method that, providing an user id, returns that user's context information. At this level the context information is the same as social group presence information.
- *Subscribe*: Any external application that wishes to be aware of changes regarding any contact's information (either address or context related) use this method to subscribe to it.
- *Unsubscribe*: Inverse operation of the previous method, removing the given application from the listener list.

It is explicit the simplicity of the interface *per se*. No restrictions are made as for the technology behind any building block that implements it.

Any social group connector developed must only provide this set of methods to be enabled for usage by our system.

4.1.2 Contact Source Interface

A contact source is any application or building block that provides address information regarding a specific entity or set of entities.

An address book, for example, represents a contact source (although our system does not interact with this type of manual devices).

Since address sources are vast, specially with the proliferation of personal emails, business emails, multiple cellphones per person, etc, address information can be provided by several different systems.

In order, once more, to allow scalability we define an interface that any contact source must implement.

One important note, all methods specified by this interface require, apart from eventual additional arguments, the owner's id, in order to assure that only a person authorized to do so accesses to address information.

- *Get Contact Addresses*: This method provides all known address information regarding a specific entity.
- *Get All Contacts Addresses*: Similar to the previous method, this one provides a set of information, indexed by entity id, regarding all entities address information.
- *Sync Contacts*: Method with default empty logic that when defined enforces a synchronization action between our system and the source system. This method stimulates information freshness.

Once again the set of methods do not limit at all the systems that can interact with our own system.

4.1.3 Context Source Interface

Context sources represent external systems that can provide contextual information regarding an entity or set of entities.

It is virtually impossible to standardize a representation of all context as we have discussed previously (see section 2.1.2), it is however possible to create a generic representation, based on raw objects, which in term can be cast at application level to a more meaningful type.

Lets however analyse this interface's methods.

- *Get Contact Context*: Similar to contact addresses, this method provides the context information known to a certain user. Additionally we must define which source we want to obtain the information from (based on the connector type).
- *Get Contact Context*: This method is very similar to the previous one, however it provides a set of contextual information based on all available context sources.
- *Subscribe*: Due to the fact of contextual information being highly volatile, any application that wishes to be aware of such changes can use this method to register itself over the context source.
- *Unsubscribe*: Opposite action of the previous method, removing the given application from the listeners list.

The first method is important however for building blocks that must somehow execute pre-processing over the context information provided, based on the its source.

4.1.4 Context-Aware Group Enabler

Finally we have the main component interface. This interface describes all functionality that can be obtained from our system. It does not have the objective of scalability or extensibility, its existence is mainly due to technology factors (see section 5.2). System performance was addressed in previous components, however this component acts as a front-end between a client application and the developed system itself. The connector's logic is rather simple, invoking other components' methods to perform the functionality requested.

Lets analyse the features that this interface defines.

- *Create Group (empty)*: This method allows the creation of an empty group, associated to the user that triggered the action (owner).
- *Create Group (credentials)*: It is possible to create a group based on several external services such as social groups. This method is capable of creating an unified point of view over the contacts an entity has based on the services that he/she has access to. To accomplish this the user must provide his/her credentials to each service. We will discuss problems that arise from this approach later on this document (see section 5.1.1).
- *Create Group (named)*: Similarly to the *Create Group (empty)* this method creates an empty group, however it has a name associated with it.
- *Get Group*: Returns the group associated with the given id.
- *Get Group Element*: Returns an entity representation that belongs to the group with the specified id and that has the specified entity id.
- *Get Groups From Owner*: Not all users of the system have the same privileges over our system (see section 4.3), for the users that do have privileges to create groups (Customers) this method allows to obtain the list of groups that a specific user owns.
- *Get All Group Owners*: With this method it is possible to obtain all users that are owners of at least one group.
- *Get Group Owners*: A group might be related to one or more owners. This method returns a list containing all its owners.
- *Get Group Context*: Group context is inferred based on rules over the context of its elements and attributes of the group itself. How this is obtained is analysed on section 2.3. This method allows an application to obtain such information.
- *Get Group Element Context*: Similarly to a group, a group element also has context, this one however is inferred based on the entity's context and the activities that the

element is performing on the group. This method allows an application to obtain such information.

- *Add Group Owner*: Self-explanatory method that adds a new owner to a group.
- *Add Group*: An application can create what could be called as an "offline group". This method allows the application to store such group in our system (with all further features that comes with such action).
- *Add Element*: Method to add an element to a group.
- *Join Group Elements*: Two entities obtained from different sources might reference the same entity in the real world. This method unifies two entries that meet this conditions (the merging process will be addressed on section 2.3).
- *Update Group*: A group can be altered outside the scope of our system. To keep our system up-to-date this method can be used.
- *Update Element*: Similarly to the previous method, this method scopes the perspective of a group element.
- *Remove Group Owner*: Opposite to *Add Group Owner* this method removes an owner from a group, except if he/she is the only owner left.
- *Remove Group*: Method to be used whenever a group should be deleted.
- *Remove Element*: Method that removes a specific element from a group.
- *Set Group Context Rules*: Context inference is based on rules. We will discuss this matter later on, this method however allows for the definition of the rules to be used.
- *Set Group Element Context Rules*: Like the previous method, this method specifies the context rules to be applied to a particular group element.
- *Register*: Finally we have a method so an entity can register itself in our system.

4.2 Components

Lets now analyse the building blocks that implement these interfaces and understand there relevance within our system, namely: GTalk, Sapo Messenger and MSN Connectors, Social Group Aggregator, the Funambol Connector, the Upcase Connector and the Contect-Aware Group Enabler internals.

4.2.1 GTalk, Sapo Messenger and MSN Connectors

GTalk is a social group service provided by Google Inc. Similarly Sapo Messenger is another social group service provided by Sapo and finally, MSN is the analogous service provided by Microsoft ®.

All these connectors implement the *Social Group Connector Interface* interface, therefore their inner workings is irrelevant for any other building block or application that uses them.

Their main goal is to serve as a translator between the mentioned external services and our system. Therefore if another external social group service were to be added a similar connector should be created for it. Apart from the effort required to develop a connector that understands the service and implements *Social Group Connector Interface* interface, its insertion within the core system logic is trivial, since our approach was to keep the system unaware as much as possible about the lower level implementation details.

For the system itself these connectors are simply interfaces. This means that removing or adding a new connector will make virtually no impact, apart eventually from performance metrics.

4.2.2 Social Group Aggregator

This building block aggregates all social group services, providing their information as an unified set of data. Social groups will be discussed in more detail later on (see section 2.2) however it is important to retain the multiplicity of information that these groups provide, contact information (end-points of communication) and context information (presence).

As we can see in picture 4.1 the *Social Group Aggregator* building block "consumes" building blocks that implement the *Social Group Connector Interface* interface. In turn this component provides two distinct interfaces, an address related one, *Contact Source Interface* interface, and a context related one, *Context Source Interface* interface.

4.2.3 Funambol Connector and Funambol Connector Sync Source

Funambol is a synchronization system that uses SyncML to execute a set of functionalities (see section 3). This system is external to our own, meaning that the appropriate infrastructure had to be built in order to enable communication.

Funambol Connector represents, analogously to the *Social Group Aggregator*, a translator between two systems. This translation however required another sub-component to interact on-the-fly with such system, hence *Funambol Connector Sync Source*.

This connector only provides address related information, therefore it is only required to implement the *Contact Source Interface* interface.

4.2.4 Upcase Connector

Upcase was a parallel project developed by Instituto Superior Técnico and PT Comunicaçoes. The objective of this project was to determine the user's activity (or activities) based on sensors placed on the surroundings, such as cellphone, clothes and similar.

This information represents context information, undoubtedly demanding this connector to implement the *Context Source Interface*.

4.2.5 Context Aware Group Enabler System

The core system contains a working implementation of the *Context Aware Group Enable* interface and provides all the functionalities discussed up until now.

Many aspects of the group environment have been taken into consideration. Our system aims to provide a rich set of functionalities and information that enables any application to excel at their objectives.

4.3 System Entities

As any slightly complex system different concepts are involved. Furthermore the relationships between this concepts represent the core guidelines of any project's design (see figure 4.2).

In this chapter we will present the concepts defined by our system as well as the relationships between them.

We are going to approach the information architecture's explanation using a bottom-up approach, as such, the core concept is undoubtedly the different contact types.

We will begin by describing the different contact types existent in the project, followed by a closer look over the addresses, context, entities (which include customers and external entities), group elements, group and the distinction between internal and aggregated context.

4.3.1 Contact Types

The source of information might be important, meaning that contact types are based on extensible enumerations which will be related to any contact information provided by the system.

This approach allows both a generic treatment: Disregarding the information's source using solely the information acquired; or a more specific treatment: Taking in consideration the contact source to trigger a certain action/reaction, such as acting only based on information gathered through the GTalk service.

4.3.2 Addresses

Following up the previous section, contact addresses were the next logical concept to be defined.

Simple observation is enough to acknowledge the multiplicity of means that one can use to contact another person. In our system this evidence is even more present since our system

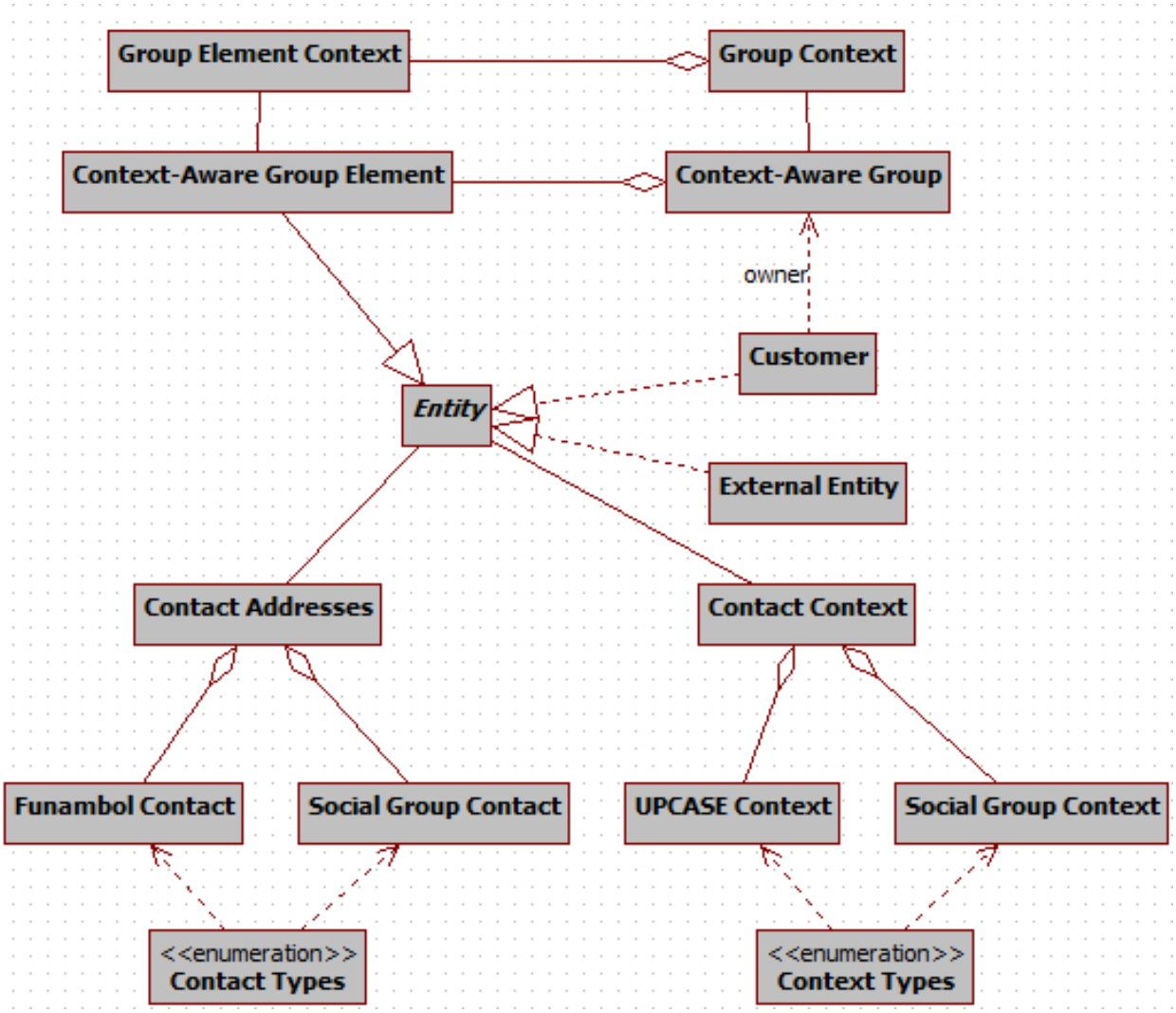

Figure 4.2: Information Architecture

itself interacts with multiple different sources of such information.

As such, this concept has the objective of encapsulating all the contact information known to the user such as street address, phone number, cellphone number and so on.

Issues however arise, specially when the merge of contacts is required (see section 2.3).

4.3.3 Context

Our system objective is to coordinate both contact addresses and context in order to provide a richer set of features. A contact's context represents a concept where the low level context information provided by the context sources is associated with that contact.

One important key element in this concept is the multiplicity of sources which provide this type of information. Since our information can be accessed by different applications building blocks or even applications themselves we must not restrict the information we provide. Following our philosophy related to contact types (see section 4.3.1) all context

information, regardless of source, is made available to any request presented.

4.3.4 Entity

An entity represents a person, system or even a group that in some way interacts with our system. The importance of this concept is that every action taken or advised by the system is based on information provided by the entities involved.

One important aspect of entities, although being defined as abstract classes/objects, is their unique identifier which exists regardless of specialization and allows the system to unambiguously identify an entity within the system.

Due to company policies URI generation must not be delegated to database management system, instead the application itself must generate such identifiers. For this project an URI generator was created, generating URI's that have a company well defined prefix followed by an unique number.

As previously mentioned, entity is a rather generic concept, therefore specializations of it were made.

4.3.4.1 Customer

A customer represents an entity that is registered in our system. This parameter implies a possible and probable bigger knowledge of the entities address and context information, turning them into valuable sources of information.

One other important aspect is related to system privileges, only customers are allowed to fully use our system's features.

4.3.4.2 External Entity

An external entity represents an entity that although not being registered in our system has information related to it, gathered through our system and providing a set of features to be used towards such entities. A contact entry of a Customer's contact list is an example of such type of entity.

The external entity might not even know about the existence of our system and does not have any privileges to use its features, even if it belongs to a group (see section 4.3.5).

4.3.5 Group Element

An interesting fact unveiled during the development of our system resides in the fact that an entity, by itself, might not be related to any group. Taking the example a newly created customer, even though the system probably possesses information about it, at this particular stage the customer is not associated with any group.

Opposed to this is the possibility of an entity being associated with multiple groups, having particular behaviours and contributions to each one. Both the former and the latter provide

valuable information for the group, since such information might be considered context (see definition of context).

As such, a group element as the name implies, represents the concept of an entity stipulating however a compulsory relation between the mentioned entity and a group (see section 4.3.6).

4.3.6 Group

We come then to the most important concept, group. Although being a very important one it is not complex at all. A group is no more that an aggregation of group elements (and implicitly entities) based on a particular criteria.

By criteria we understand for instance the list of personal contacts associated with a particular entity, or a company department.

The contact entries of a user's GTalk contact list represents a group. The criteria in this case being those entries belonging to the users contact list. In the event that those contact entries are not aware of our system, each entry represents a Group Element (see section 4.3.5) closely related with an External Entity (see section 4.3.4.2).

We can conclude that in a group the system has knowledge regarding the element's context, whether an element is associated with an external entity or a customer (see section 4.3.4.1), the context itself will be discussed further on (see section 4.3.7). As an entity, address information is also available to each entry.

Since, in this scenario, each contact entry is compulsory associated with a group, they have context within that group. We mentioned that an entity might not be related to a group unless it is closely associated with a group element, and consequently a group.

4.3.7 Internal and Aggregated Context

One important aspect of a group element is its context. We have already identified the context concept (see section 4.3.3) however, a group element or a group should not rely solely on low level context.

With this objective present every group element has associated with it what we call "aggregated context". Aggregated context contains all the low level context previously mentioned as well as higher level context inferred from rules and priorities (see section 2.3.3).

Aggregating context (or any type of data) instantly brings up several conditionals to the matter, data loss, data consistency, custom treatment and inference. We will discuss in more detail this issues further on (see section 2.3).

This aggregated context however cannot accommodate all the information that the system actually has.

4.3.7.1 Group Element Perspective

A group element is an entity associated with a group (see section 4.3.5). Every entity in our system was built gathering information from several different sources, in order to build a richer and complete representation of it.

But what about context provided by belonging to a group? Every group element has associated with it an internal context. This concept accommodates all activities that the group element is performing within the group, such as typing, presenting a slide-show, sharing desktop, etc.

We conclude therefore that a group element's context is the combination of its aggregated context along with its internal context within the group.

4.3.7.2 Group Perspective

Similarly to group elements, a group has these two components.

A group's aggregated context if inferred from the context of its elements (see section 2.3.3). Along this form of context is the group's internal context. The later is the combination of different properties of the group, ranging from group theme, interests, number of elements and virtually any property that can be found relevant to a certain situation,

4.4 Generic Interfaces

In any system the implementation techniques used, limit or catalyse the scalability of the system itself.

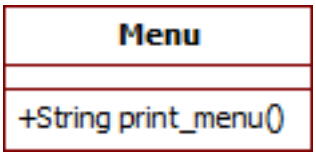

Figure 4.3: Menu Example

Taking a basic restaurant menu manager program for example (figure 4.3). The objective of such program would be to output the menu of said restaurant. Only the method *print_menu* is defined and, to some extent, it would be enough.

Lets however imagine a situation where we want our program to have different menus each day. With the architecture of figure 4.3 we would not be able to do so. Figure 4.4 shows a possible alternative. We acknowledge that the requirements are met, however, changing menus over the weeks might be problematic, since it would require to change the core program.

Both approaches shown before to note provide scalability to the system. But how about interfaces?

Interfaces generally refer to an abstraction that an entity provides of itself to the outside.

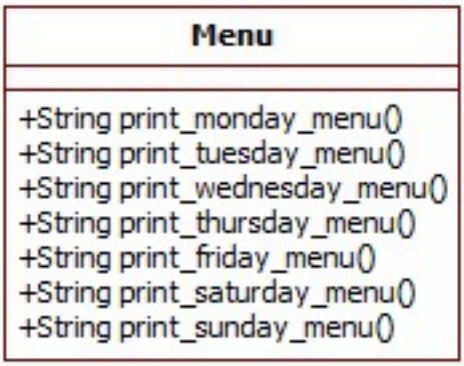

Figure 4.4: Menu Example 2

This separates the methods of external communication from internal operation (for example two different functions written in Java language have the same interface if they have the same arrangements of arguments and the same type of return value, but the function body may be implemented in different way), and allows it to be internally modified without affecting the way outside entities interact with it, as well as provide multiple abstractions of itself.

We analysed the interface possibility (figure 4.5). This approach bases its behaviour on the creation of an interface. Implementation is left for any other class that decides to implement this interface, in the example of figure 4.5 *Monday Menu*, *Tuesday Menu*, etc. What is the use of this approach? Any application that wants to obtain the restaurant menu must only know the interface. Due to metamorphism properties of Java object-oriented programming language, any class that implements such interface can be access as if it were of the interface type, not its own type. (expose better?)

Other advantage of this approach is precisely scalability. We added seven menus to the program and the way to access those menus is exactly the same, whether we had one menu or one thousand. In our example we defined menus for every day of the week, we could also define a default menu and specific menus for weekends only. The impact on the core program would be virtually none.

Interfaces also provide a generic way to access to the system. Any class with its own implementation is accessed the same way as any other.

Other major feature of interfaces is the possibility of developing a system in a modular way.

Modular programming is a software design technique that increases the extent to which software is composed from separate parts, called modules. Conceptually, modules represent a separation of concerns, and improve maintainability by enforcing logical boundaries between components. Modules are typically incorporated into the program through interfaces, as we could see in the menu example. A module interface expresses the elements that are

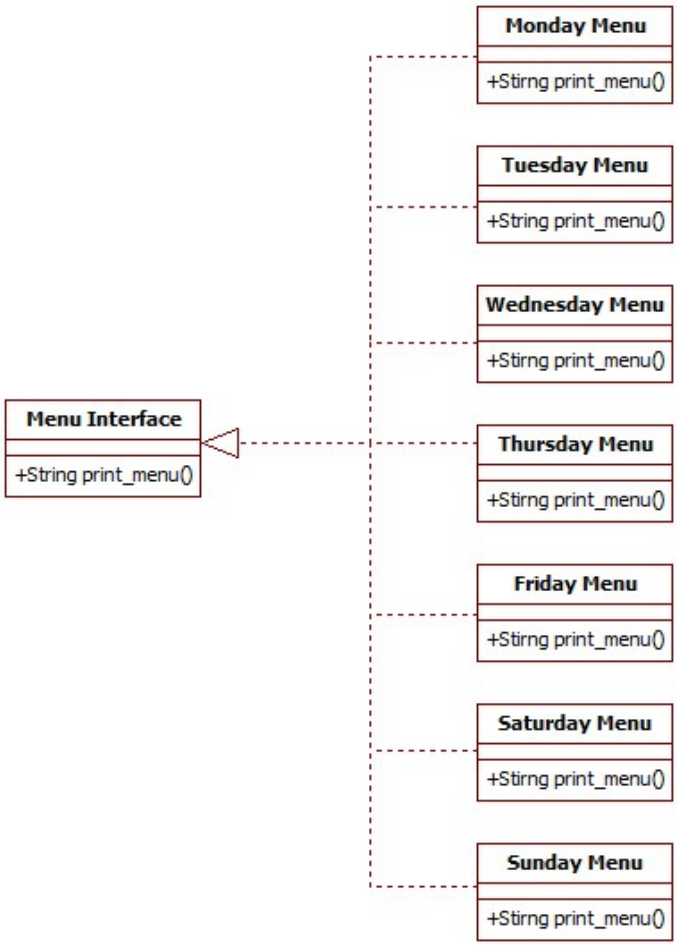

Figure 4.5: Menu Example 3

provided and required by the module. The elements defined in the interface are detectable by other modules. The implementation contains the working code that corresponds to the elements declared in the interface.

Before getting into our system, we must first look at a very important component of JBoss.

4.4.1 Integration with JNDI

Java Naming and Directory Interface (JNDI) is a Java API for a directory service that allows Java software clients to discover and look up data and objects via a name. Like all Java APIs that interface with host systems, JNDI is independent of the underlying implementation. Additionally, it specifies a service provider interface (SPI) that allows directory service implementations to be plugged into the framework. The implementations may make use of a server, a flat file, or a database; the choice is up to the vendor.

The name used is often referenced as *binding name*, being all EJBs[1] (see section 5.2) associated with one. In our system each individual social group connector is an EJB, therefore they are registered in our application server's JNDI. We defined a local naming convention for social group connectors. Each connector is bound to the name *SocialGroupConnector-Interfaces/local*. Some might ask the reason for this apparently irrelevant convention.

4.4.2 Social Group Interface

As we have seen before, social groups represent a very important part of our system (see section 2.2). Having different protocols, inner workings, authentication systems among other things, they behave much like the weekly menu. They all provide the same information, not necessarily the same way.

Hence we developed a specific interface for social groups. This interface defines all methods that a social group connector must provide, such obtaining the contact list or an entry presence information. The way each connector provides this information is up to the connector itself.

As we have previously exposed, we developed three social group connectors:

- GTalk Connector
- Sapo Messenger Connector
- MSN Connector

All these connectors in term implement the social group interface. This means that the social group aggregator module works solely with this interface, and not needing to know the connectors (and social group services) that it deals with.

Enterprise JavaBeans (EJBs) are based also on interfaces in order to provide a modular approach to software development. This is important due to the technology dependent technique used to abstract even more the connectors that are in fact in use, as well as providing weak connections between connectors. By weak connections we mean connections between components that do not make them dependent of one another.

The social group aggregator, although dealing exclusively with social group interface types, must obtain working instances of connectors that implement this interface.

The basic approach would be to explicitly write in the source code the location of each connector. The aggregator would not need to know the class of the connector, only its location.

This approach does not meet our needs. Any change in the connectors (adding or removing a connector) would require changing the aggregator, and worst, recompiling it.

[1] Enterprise JavaBeans (EJB) is a managed, server-side component architecture that encapsulates the business logic of an application, used for modular construction of enterprise applications.

At first we used property files. Property files are system files that can be changed in runtime and possess several attributes important to govern the application that uses it.

We used such file to list the connectors that the social group aggregator should use. This was a much better approach, adding and removing a social group connector would only require to change the property file. However, in our system, social group connectors might be dependent on a user basis, each user not needing necessarily all social group connectors.

This was precisely the motivation for the usage of name discovery techniques. Using the JNDI technique allows the aggregator to load any connector that its useful to it (the aggregator looks only to the local naming convention we mentioned before), being trivial the addition or removal of connectors.

4.4.3 Contact Source Interface

Following the modular programming philosophy, contact sources fall in the same level as the daily restaurant menu.

A contact source is a provider to the system. It provides addresses related to an entity. We have already discussed contact sources in our system, namely social groups (see section 2.2) and Funambol (see section 3).

The number of contact sources that system might use is not limited to the ones developed throughout this project. Once more scalability must be assured. Having this in mind, contact sources are also bounded to an interface named *Contact Source Interface*. The "Contact" part of the name refers to means of communication, addresses, and not entities in the general meaning of the word.

The basic inner workings of an application or building block that uses contact sources are rather simple. Similarly to social group connectors, contact sources provide the same set of known methods, and can be treated as being of the type of the interface. The said application or building block does not need to know the particular realizations of the interface.

In term all contact sources are used, providing a multiple source contact address information. This brings out an already mentioned problem, merging information. How can we merge information based on multiple sources? Contact address information is actually rather simple to merge, since most information is complementary to one another. In case of conflicting fields, a basic timestamp technique is enough to assure consistency and freshness of information.

4.4.4 Context Source Interface

Context sources are much alike contact sources, and most of the basic concepts apply to both of these interfaces. The main distinction between them however is a very important aspect, how to merge context.

Multiple context sources provide a rich wide variety of contextual information. We can't however make a merging decision based on timestamps for instance, since the confidence

attribute is associated with all context information. Merging information is further discussed on section 2.3.

In this chapter we analysed the system interfaces (see section 4.1), the motivation behind their existence and the features provided by them. We have also analysed the system components (see section 4.2), the *building blocks* of the system which implement the system interfaces. The information concepts, relationships and importance was discussed as well (see section 4.3) having finalised with a closer look over interfaces directly connected with external systems (see section 4.4).

Chapter 5

Implementation

In this chapter we will analyse implementation specific situations and problems that arose during the development of the project. Since this project was developed under the standards and objectives of a communications company, several implementation aspects had to address company specific concerns and rules.

We will begin by focusing on social groups and how it is possible (and relevant) to obtain and use information provided by this type of systems. A technology discussion will address Enterprise JavaBeans (EJB), the relevance of this programming framework, and specific considerations regarding the Funambol system.

We will address persistence as well, and the implementation decisions taken regarding this subject, along side a brief consideration about important database management systems to our project. To finalise this chapter we will analyse on an implementation perspective rule-based inference and management systems.

5.1 Social groups

Social groups are an undeniable rich source of information, but we also aim to go beyond the speculation and theoretical levels, hence we focused on interacting with real world, well-known services that not only provide the information we need but also adds to our application a real world value and a tangible objective towards commercial usage.

We will focus now on how does our system use these groups to acquire the said value. Roughly the process can be described in two stages, obtaining information and using information.

5.1.1 Obtain Information

In order for our system to obtain information from social groups it must have a way of representing the user before these groups.

The user must provide their authentication information to our system, which triggers a well-known issue, trust. Although our system does not use personal information beyond the

scope of its features many users feel reluctant to share personal information with third-party applications.

In a scenario where the user provides his/her authentication information our system in term accesses the services on behalf of that user. On systems that allow multiple endpoints to coexist simultaneously our system is invisible. This approach prevents the occurrence of misunderstandings regarding the users actual location and availability.

Our system follows all steps that a normal client-side application does (see section 2.2.1) not allowing however to communicate directly with anyone through it.

As any client-side application our system will be aware of the user's contact list, since the service itself provides this information.

The initial information, both address and context, is automatically gathered after successfully impersonating the owner. We however acknowledge that changes can occur quite frequently, specially information related to presence, and therefore context.

Virtually all social groups are based on the publisher-subscriber design pattern and most of them work on a push-based model (elaborate here or above?) to propagate this changes. This model assumes the usage of some type of listener to serve as a proxy of the subscriber.

Our system, to be real-world viable had to address the volatility of the information and therefore behave as most social group applications. To each social group connector we developed a listener to represent our system, that listener acts as a subscriber to every information changes to any objects existent and relevant on the service. Through this method we are able not only to obtain information at initialization stage but also to be aware when it changes on-the-fly, propagating it to the rest of the system.

5.1.2 Use Information

With the address and context information related to social groups gathered it is possible now to use it as necessary. This low level information must be converted to an agreed generic representation used throughout the system (see section 4.3.3).

In the last section (section 5.1) we discussed the importance of keeping the low-level context information as is, but one important aspect also is to be aware if the aggregated information was provided by one or several different sources.

Any object representing social group context has by default an attribute which indicates from where the information was gathered. In the event that it originated on more than one source a "generic" label is applied.

5.2 Technology-related Aspects

All these techniques have a technological background, being decisions based on future work within the company's philosophy, compliant with existing and operating software infrastruc-

tures and technology choices.

For interoperability purposes the programming language used was Java. Java is a object-oriented programming language developed and supported by Sun Microsystems. Its execution is based on pre-compiled source code (known as byte codes) that run in any Java Virtual Machine (JVM), regardless of the computer's architecture.

Microsoft's .NET framework, like Java, is an object oriented multi-programming-language framework that allow different programs, written in different programming languages, to interact with one another as long as they share a common ground for understanding, in this case .NET. Two main reasons however benefit Java over .NET

1. Released under the GNU General Public License [1] Java is free or charge and can be used without restrictions, unlike .NET which is proprietary and although being distributed for free require the payment of software licenses when it comes to development environments.

2. Java is cross-platform. JVMs are availabe for almost every operating system used today, which represents the biggest asset of this technology. On the other hand .NET officially is only supported on Windows®environments, although community projects exist to make it available on other platforms [2]

This disassociation with a computer's architecture and even operating system allows the usage of the same program anywhere with virtually no changes required.

Within the Java technology several different ways of development can be taken, from web development, applets, widgets, to enterprise applications. In our case we developed our system with the objective of an enterprise usage. To develop at this level Java provides the so called Enterprise JavaBeans Technology (EJB).

Component Organization (EJB)

Enterprise JavaBeans (EJB) technology enables rapid and simplified development of distributed, transactional, secure and portable applications.

Similarly to JSLEE specifications, EJBs are no more than building blocks that can be developed independently or can be combined to provide higher level features.

Having modularity, extensibility and scalability as main conditionals of our system (see section 4) the EJB technology provides an adequate development environment.

Our system uses Enterprise Java Beans (EJBs) as building blocks, most of which are stateless. This means that upon a request a bean is created to address it and upon completion it is eliminated. Lazy fetching the information is not possible in this situation since the objects themselves could (and most likely will) be treated in another EJB or application.

[1] GPL: http://www.gnu.org/copyleft/gpl.html
[2] Mono: http://www.mono-project.com/

EJB technology only addresses the development of building blocks, in turn these building blocks must run on an application server that provides all the technology's features and execution environment.

Once again this choice was made given the company's policy. The JBoss application server, an implementation of Java Enterprise Edition (J2EE) developed by Red Hat, was used to accommodate our EJB building blocks

Funambol-specific Considerations

Following Funambol's architecture (see chapter 3) there were two possibilities in order to achieve a successful interaction between Funambol itself and a client application.

Funambol's definition of module consists of a server extension which defines its own way to synchronize information, deal with requests and even synchronized information. This approach however brings forward an significantly complex issue, the re-implementation of all needed interfaces as well as all plugins.

There are plugins for virtually all systems and mobile devices[3]. Implementing a new module would implicate the re-programming of all plugins that we had intention of using.

This approach would allow us to synchronize the information as we wish, formatted to a standard of our choosing for example. However such commodity would not come without a huge work overhead.

For the sake of proof of concept we took a different direction, the definition of our own plugin (not module). Through a plugin, we were able to access both the core functionalities of Funambol with minimal configuration requirements but we could also easily interact, upon the completion of the plugin, with all the other plugins, enabling our system to indirectly access information provided by Outlook®, cellphone, or any other system (virtually all) that has a plugin developed for it on Funambol.

Every plugin synchronizes its information with the synchronization server. Lets analyse figure 5.1. Plugin X synchronizes with the server. The former contains the object A, while the latter contains no objects. After synchronization both contain object A.

When plugin Y, which contains object B, synchronizes with the server it will propagate that object to it and in term will receive object A from the server.

This method allows for several different systems that might not even be aware of one another to indirectly exchange information. Funambol connector behaves precisely as a plugin, capable of obtaining information from the server that was previously synchronized with it through other existing plugins.

One problem had however to be address. Funambol plugins must implement certain interfaces, which go beyond the scope of this document, however a limitation had to be address.

[3]Those plugins are intended to work with Funambol's PIM contact module.

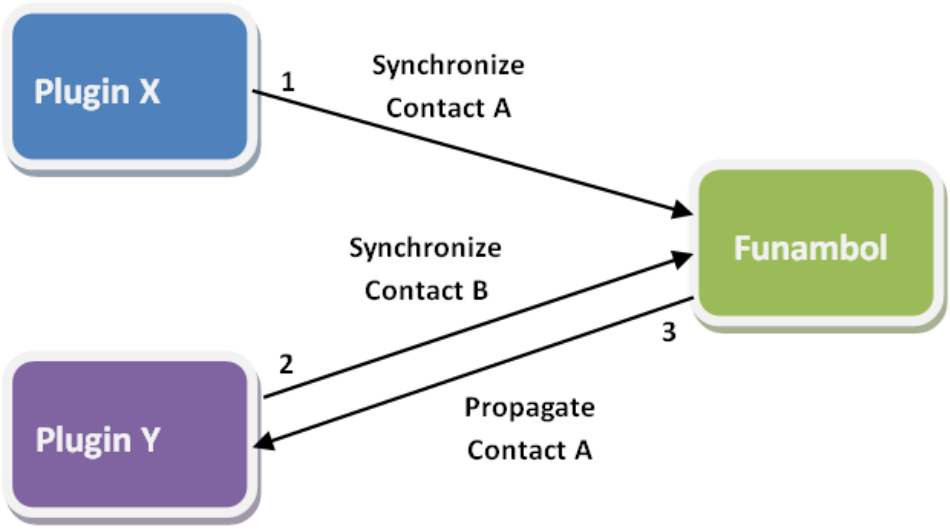

Figure 5.1: Plugin interaction

The capability of managing multiple users.

Multi-user configurations Funambol plugins do not have any built-in multiple user management system. This "flaw" had to be addressed. Developing one plugin per user of our system would make scalability almost impossible, system's complexity would increase significantly and program maintenance would also be greatly affected.

The basic authentication system used by plugins make use of property files stored in the computer to obtain multiple information, from plugin version to synchronization type (Two-way, one-way, etc.), data format and also username and password of the user using the plugin.

This information, prior to synchronization beginning, is loaded into the system's properties. Our workaround was precisely to interfere at this stage. The property file was filled with a random, non-valid authentication information. When action is required from our connector, such request must possess valid authentication information before our system. On success the system properties were changed to reflect the real user authentication information. The actual synchronization process only takes effect after this preamble.

Synchronization Source The basic requirements for a plugin to interact with a Funambol server is to implement a specific interface called *SyncSource*. This interface defines the basic operations required in any synchronization request (see section 3.2).

Although our system does not add, remove or otherwise edit any contact entry, it must store the obtained information for two main purposes:

1. **Efficiency**: Requesting full information from the synchronization server every time such information was needed would add a considerable network and processing overhead to the system.

2. **Usability**: Allowing for the rapid access of information regardless of the system trying to access it.

The solution being obviously to persist the gathered information in an easily accessible location.

5.3 Persistence

We have address throughout this document several systems that provide a rich set of information that can be used by our own system. This inf ormation however must be treated properly. We should not depend directly on external services to have access information, hence persistence is required.

5.3.1 Object-Relational Mapping

For many developers, their use of Java represents the first time they have had to contend with the challenges of object storage. Even though the Java environment includes a built-in object serialization facility to provide primitive object persistence, it is suitable only for the simplest of applications.

Persistence can be defined as a network of persistent objects whose root is serializable. Any object that a persistent object can touch automatically becomes persistent and thus a member of the network. This concept is called persistence by reachability. At storage time, the Java environment converts all the objects in the network to a byte stream for storage in a flat file [8].

The limitations of serialization stem from the fact that the entire network of objects must be accessed as a whole. It is not possible to manipulate an individual object. Each time a member of the object network is stored, the entire byte stream containing all objects in the network has to be serialized and stored. The opposite procedure occurs each time a persistent object is accessed. Moreover, object serialization lacks fundamental database capabilitieslike transactions and queriesthat multi-user enterprise applications require. Clearly, serialization is inadequate for robust application development.

Data management tasks in object-oriented (OO) programming are typically implemented by manipulating objects that are almost always non-scalar values. For example, consider an address book entry that represents a single person along with zero or more phone numbers and zero or more addresses. This could be modelled in an object-oriented implementation by a "person object" with "slots" to hold the data that comprise the entry: the person's name, a list (or array) of phone numbers, and a list of addresses. The list of phone numbers would itself contain "phone number objects" and so on. The address book entry is treated as a single value by the programming language (it can be referenced by a single variable, for instance). Various methods can be associated with the object, such as a method to return the preferred phone number, the home address, and so on.

However, many popular database products such as structured query language database management systems (SQL DBMS) can only store and manipulate scalar values such as in-

tegers and strings organized within normalized tables. The programmer must either convert the object values into groups of simpler values for storage in the database (and convert them back upon retrieval), or only use simple scalar values within the program. Object-relational mapping is used to implement the first approach.

The core of the problem is translating those objects to forms that can be stored in the database for easy retrieval, while preserving the properties of the objects and their relationships; these objects are then said to be persistent.

One aim for the Object-relational database is to bridge the gap between conceptual data modelling techniques such as Entity-relationship diagram (ERD) and object-relational mapping (ORM), which often use classes and inheritance, and relational databases, which do not directly support them.

Another, related, aim is to bridge the gap between relational databases and the object-oriented modelling techniques used in programming languages such as Java, C++ or C. However, a more popular alternative for achieving such a bridge is to use a standard relational database systems with some form of ORM software.

Due to the information-intensive nature of this project, an automated mapping between object representation of information and persistent database representation greatly reduces the development time, query complexity and adaptability. Concerns regarding database consistency are delegated to well-tested, well-defined tools.

5.3.2 Persistence Framework

In our development the formerly mentioned ORM software was Hibernate, an object-relational mapping library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. Mapping Java classes to database tables is accomplished through the configuration of an XML file or by using Java Annotation

This type of frameworks not only take care of the mapping from Java (or other programming language's) classes to database tables (and from Java data types to SQL data types), but also provide data query and retrieval facilities. It can also significantly reduce development time otherwise spent with manual data handling in SQL and JDBC (Java Database Connector)[1].

Hibernate provides transparent persistence for Plain Old Java Objects (POJOs). The only strict requirement for a persistent class is a no-argument constructor, not compulsorily public. Proper behaviour in some applications also requires special attention to the *equals* and *hashCode* methods.

Most object relations of type one-to-many are represent as sets or lists of other objects (whether primitive Java types or other custom classes). This generic collection providers are supported by the Hibernate framework. One important point to mention is the access type that can be associated with this collections. A collection of objects can be accessed in lazy mode, in which only a proxy of the actual object is obtained from the database, having solely the key attribute of that object. Additional information that is required regarding

that object is fetched on request. The other possibility is to use an eager strategy in which the entire object is loaded upon the first request.

Given our system entities and architecture (see section 4.3) all entities annotated in order to be easily processed by the Hibernate framework.

Since the basic database operations (add, update, remove) do not fit our system completely custom database access objects (DAO) add to be created, however these are not common DAOs. Hibernate provides a SQL inspired language called Hibernate Query Language (HQL) which allows SQL-like queries to be written against Hibernate's data objects. This language was used to provided customized data access to objects.

5.3.3 Database Management Systems

The abstraction provided by Hibernate between the programming language and the database management system (DBMS) allowed for a simpler versality of storage and hosting.

On a first development stage the DBMS used was MySQL, provided by IBM. Although slight configurations are required, specially at the application server (JBoss) level, our system was not aware at all of the management system in use.

Since we do not seek to restrict the technology choices of possible customers the capability of changing the DBMS was a feature to look forward to.

In order to use a company's database the system had to change to PostgreSQL, an open-source DBMS. This migration had very little impact to our application, proving the capability of changing data management systems within reasonable limits.

5.4 Rule-based Inference and Rule Management

Rule-based systems satisfy the project's needs regarding context inference, which we built into our own system in order to aid it while reasoning.

Several implementations are available as of today, providing all sets of features related to this matter. We will look now more deeply into the system used during the project development, namely *Drools Rule Engine*.

Drools Rule Engine is a rule-based system developed in Java, it is JSR94 compliant[4] and aims for an open-source rule engine.

The system inner workings are beyond the scope of this document, we will however focus on rules themselves.

Whilst considering the possibility of using a rule-based system to infer higher level context an issue arose. What should be associated with rules? The whole system should not be bounded by the same rules across it nor an administrator should be restricted to use the same custom rules for all their owned groups. Rules are case-related hence each set of rules

[4] JSR94: http://jcp.org/en/jsr/detail?id=94

```
1 rule "Under Age Rule"
2         when
3                 person : Person( )
4                 eval(person.age < 18)
5         then
6             System.out.println(pessoa.name + " is under age");
7 end
```

Listing 5.1: Rule Example

is associated with a group or an element group individually rather that being generic to all system.

The basic inner workings of such a rule are rather simple: for every object of type "Person" present in the rule-engine working memory it fires the "Under Age Rule" rule. If all conditions present in the "when" section are met then any action present in the "then" block will be triggered.

We will now analyse an example rule used in our system (listing 5.2). The objective of this rule is to assess the aggregated context of a group element considering its overhaul context.

The mechanics are simple, given a group element, the rule-based systems assesses each context entry associated with the group element, keeping a counter for each distinct type. The context type that occurs more frequently represents the aggregated context of that group element.

It is important to note however that this rule is merely an example, as such the aggregated context calculation logic can be redefined with little effort by anyone interested in doing so.

```
1  package pt.ptinovacao.rules;
2
3  import java.util.List;
4  import java.util.ArrayList;
5  import java.util.HashMap;
6  import pt.ptinovacao.information.*;
7
8  function SocialGroupPresence getMoreFrequent(ContextAwareGroupElement cage){
9          HashMap contexts = cage.getEntity().getContactContext().getContext();
10         int contadores[] = new int[SocialGroupPresence.values().length];
11         int maxIndex = 0, maxValue = -1;
12         for(int i = 0; i < contadores.length; i++){
13                 contadores[i] = 0;
14         }
15
16         for(int i = 0; i < contexts.size(); i++){
17                 Object elem = contexts.values().toArray()[i];
18                 for(int j = 0; j < SocialGroupPresence.values().length; j++){
19                         if(elem.equals(SocialGroupPresence.values()[j])){
20                                 if(++contadores[j] > maxValue)
21                                         maxIndex = j;
22                                 break;
23                         }
24                 }
25         }
26         return SocialGroupPresence.values()[maxIndex];
27 }
28
29 rule "Create rule"
30         when
31                 $tipo : ContextAwareGroupElementAction()
32                 eval($tipo.equals(ContextAwareGroupElementAction.CREATE))
33                 $element : ContextAwareGroupElement()
34                 $resultado : SocialGroupPresence() from getMoreFrequent($element)
35         then
36                 System.out.println("Creating an element");
37                 insert($resultado);
38         end
39 rule "Merge rule"
40         when
41                 $tipo : ContextAwareGroupElementAction()
42                 eval($tipo.equals(ContextAwareGroupElementAction.MERGE))
43                 $element : ContextAwareGroupElement()
44                 $resultado : SocialGroupPresence() from getMoreFrequent($element)
45         then
46                 System.out.println("Merging elements to " + $element.getUri());
47                 insert($resultado);
48         end
49 end
```

Listing 5.2: System's Example Rule

Chapter 6

Evaluation

In this section we will provide evaluation data regarding the developed system. Several dimensions must be considered and the tests undertaken simulate different usage patterns and data intensity.

This project was developed under the scope of PT Inovação, SA company. The entire project is part of a commercial solution under development by this company, being as so an important asset for the overall capabilities and flexibility of the final system load.

The entirety of the company project where our work is included is out of the scope of this project

The measurements obtained during the evaluation phase addressed different dimensions:

- **Source multiplicity**: determines if the number of sources has a relevant impact in the system (i.e., performance with the number of sources).
- **Loading time**: determines the time required for the system to be fully functional and ready (i.e. system setup latency).
- **Contact sources processing time**: analyses the cost of iterating through all contact sources (i.e. performance with the number of contacts).
- **Context gathering**: determines the required time to obtain contextual information for all entities that need so (i.e. performance with context queries).
- **Entries scalability**: determines how the system behaves given a raising number of contact entries.
- **Duplicate detection performance**: Although the algorithms are rather simple, the growth of entries might affect this dimension.
- **Database impact**: determines the impact of database persistence.

	Tester1	Tester2	Tester3
Number of Contacts	2	9	17

Table 6.1: Test users characteristics

	Average	Standard Deviation	Variance	X-min	X-max	% of total time
Load time	0.0482	0.019753	0.00039	0.027143	0.069257	0.5
Processing Contact Source time	3.058	0.140032	0.019609	2.908726	3.207274	31.2
Total time to gather context	0.0094	0.014502	0.00021	-0.00606	0.024859	0.1
Total time to persist entries	4.493	0.317954	0.101095	4.154061	4.831939	45.8
Possible matches detection time	0.0078	0.001789	3.2E-06	0.005893	0.009707	0.1
Group persistence time	0.1122	0.044824	0.002009	0.064418	0.159982	1.1
Useless	2.0078	0.001304	1.7E-06	2.00641	2.00919	20.5
Total	9.808	0.312616	0.097728	9.474752	10.14125	100.0

Table 6.2: Test user 1 - Results

6.1 Test Environment

All tests ran on a Core^{TM}2 Duo CPU computer at 2.66 GHz frequency, with 4 GB ram memory and Windows Vista^{TM}Business 32-bit operation system.

The tests were based on three prototypical test users with different characteristics:

The result tables are now presented, upon which we will draw the evaluation analysis.

Before analysing the statistical tables it is important to understand the difference, in terms of percentage, between the test users.

- Test user 2 has 350% more contacts than test user 1
- Test user 3 has 88.8% more contacts than test user 2

6.2 Loading

Analysing all test user tables (tables 6.2, 6.3 and 6.4) we conclude that the loading time is constant, regardless of the number of contacts. The system was built having in mind scalability and extensibility, as such, the connectors to be loaded should not limit the system performance and interoperability.

Furthermore the loading time represents a very small fraction of the execution time, roughly 0.5% of the total time.

	Average	Standard Deviation	Variance	X-min	X-max	% of total time	% increase
Load time	0.0568	0.028891	0.000835	0.026002	0.087598	0.5	17.8
Processing Contact Source time	3.4636	0.156292	0.024427	3.296992	3.630208	27.6	13.3
Total time to gather context	0.0102	0.009284	8.62E-05	0.000303	0.020097	0.1	8.5
Total time to persist entries	6.4996	0.425991	0.181468	6.045494	6.953706	51.9	44.7
Possible matches detection time	0.0926	0.02851	0.000813	0.062209	0.122991	0.7	1087.2
Group persistence time	0.3024	0.021149	0.000447	0.279855	0.324945	2.4	169.5
Useless	2.019	0.005196	0.000027	2.013461	2.024539	16.1	0.6
Total	12.5342	0.529033	0.279876	11.97025	13.09815	100.0	27.8

Table 6.3: Test user 2 - Results

	Average	Standard Deviation	Variance	X-min	X-max	% of total time	% increase
Load time	0.0548	0.024763	0.000613	0.028403	0.081197	0.4	-3.5
Processing Contact Source time	3.2308	0.266993	0.071285	2.946186	3.515414	23.1	-6.7
Total time to gather context	0.0264	0.021893	0.000479	0.003062	0.049738	0.2	158.8
Total time to persist entries	7.719	0.10464	0.010949	7.607454	7.830546	55.3	18.8
Possible matches detection time	0.3102	0.020389	0.000416	0.288466	0.331934	2.2	235.0
Group persistence time	0.5122	0.024601	0.000605	0.485976	0.538424	3.7	69.4
Useless	2.0362	0.005357	2.87E-05	2.030489	2.041911	14.6	0.9
Total	13.9632	0.283642	0.080453	13.66084	14.26556	100.0	11.4

Table 6.4: Test user 3 - Results

6.3 Contact Sources Processing

The iteration through all contact sources is a very important aspect to be taken into account. Although it represents solely the iteration through the entries of a contact least, its performance affects significantly and directly the overall system performance in a top load situation.

As we can analyse the, the time it takes to process contact sources grows gracefully as the number of contact entries increases. The major difference detected was when there was a 350% increase in the number of contacts (table 6.3) which increased this processing by 13.3%.

This graceful growth is very successful given the global impact of this dimension, which represents more that one quarter of the total system execution time.

6.4 Context Gathering

Context gathering occurs after the acquisition of contact entries. For each entry the system tries to obtain its context, roughly representing the cumulative of contacting context sources repeatedly. All context information providers have built-in, well-defined publish-subscribe mechanisms which reduce the number of actually executed queries to a minimum. The impact of this dimension on the overall performance is almost negligible. On test user 3 we do however witness a 188% growth (table 6.4), this represents nevertheless the global impact to grow from 0.1% to 0.2%.

It is not understandable through the statistical data acquired the reason for the mentioned growth, specially due to the much lower growth analysed for a much higher increase of contact entries (table 6.3). We believe that the external services overhead might be the cause for such disparity, being only a hypothesis.

6.5 Connectors Persistence

The developed system is data-intensive, with most information being persisted using a database. The persistence process has a high impact on the overall system performance. The results of test user 3 (table 6.4) persistence represents 55.3% of the total time.

This dimension grows gracefully as well, given a high increase in the number of contact entries, it does however represent half of the total time. Optimization in database interactions will provide great improvements on the system.

6.6 Group Persistence

It is important to distinguish between the persistence of raw auxiliary data from the persistence of object graphs with contact and context information previously processed, for which there are a number of possible optimization approaches.

This persistence has a higher increase rate compared to the previous persistence section. The global impact of the group persistence is much lower than the connector persistence, but using test user 2 results (table 6.3), for a 350% increase in contact entries, connectors persistence grew 44.7%. For the same user, the group persistence grew 169.5%. Both types of persistence are below the contact growth (i.e. sub-linear overhead) percentage but they do have an unavoidable impact on the system.

6.7 Duplicate Detection

Duplicate detection, accordingly to the test results, must be reformulated in future work. The approach taken in this project revealed to be inefficient.

For a 350% contact entries growth (table 6.3) duplicate detection grew 1087.2%, similarly for a 88.8% contact entries growth (table 6.4) the same dimension grew 235%. This is the only dimension that performs worst than linear growth, being in essence in a quadratic growth complexity order.

As future work, alternative approaches must be considered, such as the use of a different data-structure that allows for a by-letter indexation (of by-first-two-letters indexation) that avoid exhaustive tuple comparison. Optimisation is mainly a space-time trade-off problem.

Duplicate detection represents the current bottleneck of the developed system. For the tests undertaken its global impact is small, however its growth rate is reason for further future study.

6.8 Global Analysis

The developed system proved to be scalable. Given a high rate of growth in the number of contact entries, the overall performance of the system grew gracefully. The addition of new sources, either address sources or context sources, produce a minor impact in the mentioned system.

Stateless Java Beans allowed for on-the-fly resource allocation to take place.

Extensibility was achieved as well. The addition of new sources (GTalk®) required close to no effort aside from the interface implementation.

The interface approach was capable to provide a real extensible framework.

The system was ported to another environment, based on Linux operating system and PostgreSQL database management system. It was not possible due to company policies to execute performance tests in such environment. It is possible however to conclude that the system is in fact portable.

Choosing the Java language and Hibernate as the persistence manager empowered our system with portability across platforms with very little effort required.

It is important to mention some values acquired. For test user 3 (table 6.4) there are

negative increase percentage values, even when the number of contact entries grew. The only explanation for this fact relies within the external services' performance which the developed system cannot control.

On all result tables there is a dimension named "Useless". This dimension has a close to zero growth percentage. It represents system actions that can be removed with no impact at all for the system. These actions go from log functions to auxiliary console information and debug information.

During the execution of the tests these actions could influence the real final result of the relevant dimensions, therefore its impact was kept separate from other dimensions.

Chapter 7

Conclusions

During the development of this project we concluded that the need for a context-aware application in the addressed scenario was real. The application should be capable of adapting its behaviour autonomously and dynamically according to external factors.

We analysed several existing approaches to context-aware software, either in terms of representation, reasoning, and sensing. Social groups are an important component of this project. Their information value (and commercial usage) meant a challenge and an asset. Ensuring interoperability between heterogeneous systems required the definition of capable data representation formats and translators. The information provided by these systems proved to be an asset based on the information richness gathered through them.

Information merging proved to be a complex subject, specially when customisation must be supported. Although the performance regarding the duplicate detection in particular was not as initially expected, adaptability is still a reality, with users being able to change merging rules as they see fit.

The architectural design of this project proved to fit the objectives pre-determined. The system's scalability and portability were observed, being interoperability already proved, due to the interaction with external systems.

Bibliography

[1] *Hibernate - Relational Persistence for Idiomatic Java*.

[2] Boeing 777 digital design process earns technology award. Technical report, 1997.

[3] Agnar Aamodt. Towards a model of context for case-based diagnostic problem solving. In *in Context-97; Proceedings of the interdisciplinary conference on modeling and using context, (Rio de Janeiro*, pages 198–208, 1997.

[4] Gregory D. Abowd, Christopher G. Atkeson, Jason Hong, Sue Long, Rob Kooper, and Mike Pinkerton. Cyberguide: A mobile context-aware tour guide. *Wireless Networks*, 3:421–433, 1997.

[5] Matthias Baldauf. A survey on context-aware systems. *Int. J. Ad Hoc and Ubiquitous Computing*, 2, 2007.

[6] Louise Barkhuus. Context information vs. sensor information: A model for categorizing context in context-aware mobile computing. *Symposium on Collaborative Technologies and Systems*, pages 127–133, 2003.

[7] Louise Barkhuus and Anind Dey. Is context-aware computing taking control away from the user? three levels of interactivity examined. *Proceedings of UbiComp*, pages 150–156, 2003.

[8] Douglas Barry and Torsten Stanienda. Solving the java object storage problem. *Computer*, 31(11):33–40, 1998.

[9] Victoria Bellotti and Keith Edwards. Intelligibility and accountability: Human considerations in context-aware systems. *HUMAN-COMPUTER INTERACTION*, 16:193–212, 2001.

[10] P. J. Brown. The stick-e document: a framework for creating context-aware applications. *ELECTRONIC PUBLISHING-CHICHESTER*, 1995.

[11] P. J. Brown, J. D. Bovey, and X. Chen. Context-aware applications: from the laboratory to the marketplace. *IEEE Personal Communications*, 4(5):58–64, October 1997.

[12] Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo. Reflective middleware solutions for context-aware applications. pages 126–133, 2001.

[13] Harry Chen, Sovrin Tolia, Craig Sayers, Tim Finin, and Anupam Joshi. Creating context-aware software agents. *The Knowledge Engineering Review*, 2004.

[14] Harry Lik Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, Baltimore County, 2004.

[15] Keith Cheverst, Nigel Davies, Keith Mitchell, Adrian Friday, and Christos Efstratiou. Developing a context-aware electronic tourist guide: some issues and experiences. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–24, New York, NY, USA, 2000. ACM.

[16] Ekaterina Chtcherbina and Marquart Franz. Peer-to-peer coordination framework (p2pc): Enabler of mobile ad-hoc networking for medicine, business, and entertainment. In *In Proceedings of International Conference on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet (SSGRR2003w) (L'Aquila/Italy)*, 2003.

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

[18] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 2001.

[19] Anind K. Dey and Tim Sohn. Supporting end user programming of context-aware applications. In *CHI 2003 Workshop on End User Development*, 2003.

[20] Tao Gu, Hung Keng Pung, and Da Qing Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 2005.

[21] A. Paul Hare. *Handbook of small group research.* Free Press, 1976.

[22] Frederick Hayes-Roth. Rule-based systems. *Commun. ACM*, 28(9):921–932, 1985.

[23] Albert Held, Sven Buchholz, and Er Schill. Modeling of context information for pervasive computing applications. *Proceedings of SCI 2002/ISAS 2002*, 2002.

[24] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Generating context management infrastructure from high-level context models. In *In 4th International Conference on Mobile Data Management (MDM) - Industrial Track*, pages 1–6, 2003.

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

[26] Richard Hull, Philip Neaves, and James Bedford-Roberts. Towards situated computing. *Wearable Computers, IEEE International Symposium*, 0:146, 1997.

[27] Jadwiga Indulska, Ricky Robinson, Andry Rakotonirainy, and Karen Henricksen. Experiences in using cc/pp in context-aware systems. In *In Proc. of the Intl. Conf. on Mobile Data Management (MDM*, pages 247–261. Springer, 2003.

[28] Jadwiga Indulska and Peter Sutton. Location management in pervasive systems. In *ACSW Frontiers '03: Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, pages 143–151, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[29] Lalana Kagal, Vlad Korolev, Harry Chen, Anupam Joshi, and Timothy Finin. Centaurus: A framework for intelligent services in a mobile environment. In *In Proceedings of the International Workshop on Smart Appliances and Wearable Computing (IWSAWC*, 2001.

[30] Fahim Kawsar, Kaori Fujinami, and Tatsuo Nakajima. Experiences with developing context-aware applications with augmented artefacts. *Experiences with Developing Context-Aware Applications with Augmented Artefacts*, 2005.

[31] William Kent. The breakdown of the information model in multi-database systems. *SIGMOD Rec.*, 20(4):10–15, 1991.

[32] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.

[33] Ludwig and Maximilians. *Service Interoperability in Ubiquitous Computing Environments.* PhD thesis, University Munich, 2003.

[34] John McCarthy and Sasa Buvac. Formalizing context (expanded notes). Technical report, Stanford, CA, USA, 1994.

[35] Felix Naumann and Matthias Häussler. Declarative data merging with conflict resolution. In *International Conference on Information Quality (IQ 2002). 2002*, pages 212–224, 2002.

[36] H. B. Newcombe. *Handbook of record linkage: methods for health and statistical studies, administration, and business*. Oxford University Press, Inc., New York, NY, USA, 1988.

[37] Yannis Papakonstantinou, Serge Abiteboul, and Hector Garcia-molina. Object fusion in mediator systems. pages 413–424, 1996.

[38] Jason Pascoe. *Context-Aware Software*. PhD thesis, University of Kent at Canterburry, 2001.

[39] Neil Rhodes and Julie McKeehan. *Palm Programming: The Developer's Guide*. O'Reilly Media, 1998.

[40] Gene I. Rochlin. *Trapped in the Net: The Unanticipated Consequences of Computerization*. Princeton University Press, Princeton, NJ, 1997.

[41] N. S. Ryan, J. Pascoe, and D. R. Morse. Enhanced reality fieldwork: the context-aware archaeological assistant. In V. Gaffney, M. van Leusen, and S. Exxon, editors, *Computer Applications in Archaeology 1997*, British Archaeological Reports, Oxford, October 1998. Tempus Reparatum.

[42] Nick Ryan. Contextml: Exchanging contextual information between a mobile client and the fieldnote server. 1999.

[43] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.

[44] Michael Samulowitz, Florian Michahelles, and Claudia Linnhoff-Popien. Capeus: An architecture for context-aware selection and execution of services. In *Proceedings of the IFIP TC6 / WG6.1 Third International Working Conference on New Developments in Distributed Applications and Interoperable Systems*, pages 23–40, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.

[45] V. Sassone. Context-aware software-intensive systems. 2004.

[46] M. Satyanarayanan, Brian Noble, Puneet Kumar, and Morgan Price. Application-aware adaptation for mobile computing. In *EW 6: Proceedings of the 6th workshop on ACM SIGOPS European workshop*, pages 1–4, New York, NY, USA, 1994. ACM.

[47] B. N. Schilit and M. M. Theimer. Disseminating activemap information to mobile hosts. *Network, IEEE*, 1994.

[48] Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *In Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE Computer Society, 1994.

[49] Albrecht Schmidt. Implicit human computer interaction through context. *Personal and Ubiquitous Computing*, 3:191–199, 2000.

[50] Quanzheng Sheng and B. Benatallah. Contextuml: A uml-based modeling language for model-driven development of context-aware web services development. *International Conference on Mobile Business (4th : 2005 : Sydney, Australia)*, 2005.

[51] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236, 1990.

[52] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *In: Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

[53] W3C. Composite capability/preference profiles (cc/pp). 2004.

[54] Roy Want, Andy Hopper, Veronica Falcão, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems*, 1992.

[55] Wapforum. User agent profile (uaprof), 2001.

[56] M. Weiser. Ubiquitous computing. *IEEE Personal Communications Computer "Hot Topics"*, 1993.