UNIVERSITY OF TROMSØ UIT

FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

A prototype system for Context Sensitive Communication in hospitals based on an Ascom/trixbox experimental platform

Lorenzo Gironi

INF - 3981
Master's Thesis in Computer Science
June,2011

Preface

Hospital's communication infrastructure suffers from different types of common problems. Currently, this infrastructure relies mainly on the use of pagers which are devices particularly interruptive for the daily work of hospital's workers, and moreover they do not support context-awareness communication. Wireless phones are supposed to be a valid alternative to pagers and they can also be used to efficiently increase awareness between workers. Unfortunately, wireless phone can become more interruptive than pagers due to the synchronous communication channel they provide.

The aim of this thesis is to propose an implementation of a context-aware solution, based on an Ascom/trixbox communication platform, which tries to overcome this problem. In particular it is specifically designed to balance availability and interruptions gained by using the Ascom wireless phones considering contextual information relating to the users carrying these devices, and it provides several features useful to increase awareness.

This work, intended for researchers and developers who are working in the field of context-sensitive communication for hospitals, is based on an on-going research project at the Norwegian Centre for Integrated Care and Telemedicine (NST), in collaboration with the University Hospital of Northern Norway (UNN) and Telenor. The focus of this project, named *Context sensitive systems for mobile communication in hospitals* is to design and develop context-sensitive interfaces, middleware and new interaction forms for mobile devices that support multi-modal communication. These solutions have been identified as a valuable way to enhance the quality of patient care in the long run [18].

First of all I want to thank my parents. They have always supported me during this thesis and without their efforts this experience would not have been possible. This thesis is dedicated to them.

A special thanks to my supervisors, Prof. Gunnar Hartvigsen and Terje Solvol who gave me the opportunity to be a member of this project and to work at NST during these months. Their advices, the time they dedicated to me, and their guidance have been really important for my thesis. I also want to thank my supervisors from my home university, Prof. Francesca Arcelli and Prof. Claudia Raibulet who also gave me several feedback about this work.

Abstract

Purpose The aim of this report is to present a context-aware solution based on an Ascom/trixbox platform, targeted at reducing interruptions caused by wireless phones and improving awareness between users carrying these devices. The application, specifically thought to be used within hospital environment, reduces interruptions considering contextual information related to users such as location, availability status and personal committments.

Motivation Hospitals are working environment where a large amount of information is constantly exchanged between workers over a complex communication infrastructure. In order to support the needs of hospital professionals, this infrastructure should provide the possibility to exchange important information as quickly as possible and at the same time contact colleagues without interrupting their working activities. Interruptions are unpleasant situations and are source of stress and distraction that may increase the probability of taking wrong decisions. Currently, hospital communication infrastructure relies on the use of pagers which create a large amount of unnecessary interruptions. Wireless phones are less utilized, mainly because can become more interruptive than pagers, but they provide text services and vocal services in the same device which, with a suitable context-aware system able to balance interruption, can be used to provide solutions able to efficiently support the work of hospital professionals.

Methods The application has been developed by using the iterative software engineering approach Unified Process.

Results The developed context-aware application is able to block the calls directed to a phone located inside a critical area (e.g operating room), offers the possibility to route such calls to the current on-call person on duty with a shift scheduled inside Zimbra calendar, is able to block the calls directed to a device switched to 'visiting' mode or calls directed to a recipient involved in a meeting recorded inside the calendar. It sends to the callers informative messages containing the location or availability status of an unreachable recipient and sends pending call messages collected during the unreachable status when users leave a critical area or switch the phone back to 'available' mode. It also provides a feature which model the behavior of a phone as a pager: with this functionality users can be paged on the phone through an interactive message which can be used to directly call back the person who put the page. Finally, it provides a user interface from where it is possible to look at the status and location of all the users enrolled in the system from a single panel.

Conclusion A number of tests carried out after the develpment highlighted that the application must fill a large amount of gaps before being deployed in a real hospital. Some of them can be easily fixed but others, due to limits of the phone devices, not. Moreover, an analysis of performance highlighted that the system is not highly scalable and that however some strategies can be undertaken in order to improve this aspect.

Keywords context-aware, interruptions, hospitals, Ascom.

Contents

1 Introduction

1.1 Background and Motivation

Hospitals are working environment where a large amount of information, used to provide reliable and high quality services to patients, is constantly exchanged between workers over a complex communication infrastructure. There are two key factors that this infrastructure should satisfy in order to efficiently support the needs of hospital professionals. The first relates to timing constraints: information must be exchanged as fast as possible because any delay between the decision made and the action taken could cause unacceptable medical errors [32]. The second concerns the possibility to contact colleagues in a safe way, without interrupting their activities. When doctors are in the middle of a surgery inside an operating theatre or visiting a patient in a ward they do not expect to be continuously interrupted by messages or calls. This is because a large amount of interruptions can become a source of distraction that may increase the probability of taking wrong decisions during their daily activities.

Satisfying these two factors is a challenging problem to face, it requires a suitable communication infrastructure that takes in consideration timing constraints, availability status and mobility of clinicians.

Currently, hospital's communication infrastructure relies on the use of pagers which are the most common devices used to contact staff's members. *The spread of pagers is such that many hospital's workers carry several of them according to the roles they have been assigned* [76]. These devices even if cheap and small present limitations, due to their simplicity. They do not support context-awareness communication and create a large amount of unnecessary interruptions because *when a page is placed, the recipient has to stop what he or she is doing, find a telephone and call the number on the pager. By the time this has been done, the caller my not be available any more* [74, 76].

Different studies have shown that wireless phones can overcome most of the limits of pagers and facilitate the communication within hospital setting [76]. Thanks to their underlying technology are also capable to simplify information access, increase the quality of patient care in the long run and *increase availability* through a synchronous communication channel not provided by pagers [2, 14, 74, 76]. Unfortunately, even wireless phones are interruptive too and potentially can become more interruptive than pagers. According to T. Solvoll and J. Scholl [78]: *when the phone rings, the persons carrying it may feel obliged to answer and explain that they will call back, if they are busy.* Despite this some researchers discovered a number of benefits achievable by using in combination voice and text services, mainly because toghether are capable to support context-aware solutions. *Since most of the wireless phones currently available provide voice and text services in the same device, the believe is that with an*

appropriate context-aware system able to also manage interruptions, they can be the first candidate to substitute the pager-based communication infrastructure in hospitals.

1.2 Who is in this area?

In the last years several authors working in the field of context-aware computing proposed a number of solutions aimed at improving awareness within hospital setting. These solutions are characterized by a common denominator: exploit functionalities of displays, palm phones, wireless phones in order to improve communications and information gathering between clinicians. Even if the final purpose is similar, the developed systems show many differences especially on the kind of information provided and on the way they display it. In the following a brief list of solutions already developed is given.

Bardram, Hansen and Soegaard proposed in the Aware Media project [4] a solution to support close coordination and communication between clinicians. It uses displays scattered throughout the hospital to show information about what kind of operation is going on in a ward, its progress status and name of the doctors involved in the surgery.

A more nurse-oriented solution, proposed by Skov and Hoeg in the project Mobile WARD [73], uses mobile phones to provide nurses information about patients considering their daily task, time constraints and location.

Aziz et al.[1] explored the capabilities of Palm Tungsten PDAs, with built in mobile phones and web-browsers, in order to evaluate if they could be a valid alternative to traditional pagers. The Palm devices used during the study were also integrated with electronic versions of medical reference text-books, drug interactions compendium and anatomy atlases. After the study an on site assessment phase has been conducted to gather feedbacks from participants.

In the Follow Me application [50], Mitchell proposed a system that allows a user to request a video call from a terminal, e.g for asking a consultation, without knowing where the recipient is located. The system is able to automatically redirect the call to the terminal nearest to the recipient by tracking the users position using Active Badges technology.

The Context-Aware Communication system proposed by Munoz [51] uses handheld devices that allow users to send messages or data specifying when and where to deliver such information. For instance, with these devices clinicians can send a lab test to the first nurse who will enter a specific room in the next morning, after the test has been carried out on a patient.

1.3 The problem

1.3.1 Problem statement

How can a context-aware system that uses the Ascom/trixbox communication platform be built? Can it be developed to specifically balance interruptions and communication availability gained by using the Ascom wireless phones?

The Ascom system is one of the most common communication platform used within hospitals. Developing a context-aware application based on this system means providing a solution which can be easily implemented inside hospitals that does not require specific hardware to run. Therefore, in order to properly answer the question above, first of all it is important to determine which are the contextual information needed to achieve the goal and how they can be extracted from the Ascom wireless phones. In particular, since the main objective is to balance interruption and availability, the most important information that needs to be extracted are location of the devices and availability status of the users. Then, once discovered how to obtain them, find a software solution which by taking in consideration the information previously extracted, is able to control the reachability of phones by using the third party PBX trixbox system.

1.3.2 Sub-Problems

The following sub problems are aimed at refining and extending the basic capabilities of the context-aware system with more advanced functionalities.

- Which are the context-aware solutions achievable by the integration of the Ascom/trixbox platform with Zimbra Collaboration Suite?

  Zimbra Collaboration Suite has a calendar tool that can be used to easily store working shift schedules and meetings. The integration of this source of information with a context-aware system could provide intresting possibilities to further reduce interruptions. For example, by looking at the entries inside the calendar it is possible to block a call when the recipient is in the middle of a meeting or route a call directed to a person located inside a critical area (e.g operating room) to the on-call person on duty, according to the stored shift schedule. Moreover, calendar's entries are valuable information that can be used to give more feedback about the current status of a recipient, increasing awareness between workers.

- Can the Ascom wireless phones be used to preserve the behavior of pagers?

  One of the major limitations affecting pagers is the impossibility to directly reply to a page because they do not support call functionality. A solution that allows a wireless phone to behave as a pager could be a useful feature. With it users can benefit from the advantages offered by the traditional pager-based interaction and at the same time call back a person who put a page without the need to search a phone nearby.

1.4 Method and Materials

A brief description about the materials/method used during the development of the context-aware application is now given.

- Ascom Unite System: is an integrated mobile communication platform made up of different modules and a set of wireless phones specifically thought to be used within hospital environment [26]. It provides a number of services including standard/interactive messages, alarm handling, user data handling and a number of tools to control calls and messages routing.
- trixbox: is an open source Private Branch eXchange system based on Asterisk. It is capable to intercept calls between two end points and provides the possibility to change their behaviour by programming the associated dialplan.
- Zimbra Collaboration Suite: is an email exchange server with advanced functionalities to manage tasks, contacts and personal commitments thanks to a powerful calendar tool. It exposes services to external applications through the SOAP protocol which can be used to gather user's information stored inside the server.

The software engineering methodology used to develop the application is based on the Unified Process, an iterative and incremental approach that splits the development process into a series of mini-projects, called iterations. In each iteration requirement analysis, design, implementation and testing are carried out. At the end of each cycle if the identified requirements are all satisfied then the process stops otherwise a new iteration begins.

The tests have been carried out by simulating with several testers typical scenarios where the features provided by the application can be involved. After each scenario we asked them what they thought about the features just used, without guiding too much their evaluation.

1.5 Major results

The developed application is a context-aware solution that integrates the Ascom/trixbox platform with Zimbra Collaboration Suite. It provides a number of features capable to balance interruptions and communication availability of the Ascom wireless phones and send feedback messages to the users. The major results are listed below:

- According to the location of a phone the system is able to stop an incoming call if the device is inside a critical area, such as an operating room.
- When a user tries to contact another user located inside a critical area the system is able to route the call to the current 'on-call' person on duty, according to the shift schedule stored inside Zimbra's calendar.

- By checking the Zimbra's calendar the system can block a call if the recipient is in the middle of a meeting.
- The system can stop or allow communications according to the user's status configured from the phone. For example, if the status of a user is 'visiting', all his/her incoming calls are blocked.
- When a previously unavailable user become available, by quitting a critical area or switching back to 'availability' status, the application sends messages containing names and numbers of the persons who tried to call during the unavailability status (pending calls).
- When a called phone is not available for one of the reasons mentioned above, feedback messages containing contextual information about the status/activity/location of the recipient such as 'the user X is in a meeting', 'the user Y is in the Operating room' or 'the user Z is visiting a patient', are sent to the caller.
- The 'pager mode' feature, model the behaviour of the wireless phones as pagers. A call directed to a 'pager mode' phone is blocked and then, if specified by the user, can be converted into a 'page message', directly usable by the recipient to call back the caller.
- In order to control the status of all the users enrolled in the system from a single panel, a minimal user interface has been developed. From the GUI it is possible to check the current availability status of the users, their location and change some options which determine the behaviour of the context-aware system such as enable/disable the reception of feedback messages or enable/disable the reception of pending calls.
- The tests highlighted that the application have to fill several gaps before being implemented in a real hospital. In particular, some features proposed by the testers are simple extension of the functionalities already provided, some can be easily implemented, but others, due to the weaknesses of the phone devices on which the application relies, not.

1.6 Organization of the thesis

Chapter 2: in this chapter, after the definitions of context and context-aware computing, a description of the most common context-aware architectures found in literature is presented. Then, a description of the major application domains with a particular focus on hospitals is given. Finally, some solution already developed are reported.

Chapter 3: an overview about the overall framework in which the context-aware application operates and a deep description of the software/hardware materials used during the project is given. Then, the software engineering approach chosen for the development is described.

Chapter 4: this chapter provides an introduction about the scope of the application and the assumptions made during its development, then the use cases describing the functional requirements of the system are illustrated.

Chapter 5: describes the construction of the application. In particular, the information flow, the overall design of the application, the classes composing the system and finally the implementation of the use cases illustrated in the previous chapter will be described.

Chapter 6: the results of the tests carried out after the development are discussed focusing the attention on the weaknesses of the system.

Chapter 7: a deep discussion of the context-aware application is reported. Here we discuss the motivation for the chosen software architecture, quality and efficiency considerations, general considerations and finally considerations about the tests.

Chapter 8: this chapter concludes the report summarizing the major points characterizing the context-aware application.

2 Theoretical framework

In the following, after giving the formal definition of the concepts context, context-aware computing and contextual information, a discussion about the main architectures found in literature is provided. In the rest of the chapter a description of the major application domains with the most interesting applications/prototypes already developed in the field are illustrated.

2.1 Context and Context-aware computing

2.1.1 Context

The research community tried over the years to better refine the meaning of *'context'* caracterizing it in terms of mutual relationships existing between actors, events and objects within a given situation. As explained in [3] the two first interpretations were based on the focus of the problem considered which could be user oriented or system oriented. The user oriented interpretation put in the center the role of users and their relationships with different things within an environment. Its first formalization was given by Schilit and Theimer as follows:

'three important aspects of context are: where you are, who you are with, and what resources are nearby'[72]

The drawback of this definition is that in many common situations there are factors relevant for a context-aware application that can not be obtained by considering only information closely related to the users.
Lieberman and Selker proposed a new definition that relies on a more system centric point of view:

'context are all the informations that are required for the computation excluding input and outputs provided to the system' [47]

With this definition authors suggest that a system should take decision according to *any* information sensed from the environment which have a direct effect on its state. For example, an application that uses a GUI should not only change displayed data in response to events generated by the users, but also change data considering other kinds of information not strictly related to them.

Years later Dey and Abowd gave a definition that takes into account both previous points of view, abstracting the concept to a higher level. Their definition of *'context'* was the following:

'Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant for the interaction between a user and an application, including the user and applications themselves.' [19]

In this definition there is not distinction between user and application: the attention is targeted on the *relevant information* which allows interactions between different entities. By considering this point of view, developers can focus

their efforts only on the information relevant for the design of the application, ignoring from wich side of the interaction the identified information comes from.

2.1.2 Context aware computing

According to Dey and Abowd, old definitions of context-aware computing can be divided in two categories [19]:

- Using context: *context-aware computing is the ability of computing devices to detect and sense, interpret and respond to aspects of a user's local environment and the computing devices themselves* [59, 60].
- Adapting to context: *context-aware computing is the ability of applications to dinamically change or adapt their behaviour based on the context of the application and the user* [59].

Unfortunately, neither of these definition are suited to precisely define the concept of context-aware computing because there are context-aware applications that dinamically change their behaviour without sending information to the users and context-aware applications that do not adapt to the context but send information to the users. For example, an application that displays the context of the user's environment to a user does not modify its behaviour, but it is certainly a context-aware application [19]: this is a typical example where the second definition doesn't fit.

A more general definition has been introduced by Dey and Abowd [19]. They gave an interpretation that join together the previous categories:

> *A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.* [19]

It embraces both the previous point of view because first it *uses context* to provide relevant information and/or services to the user and at the same time provide the information according to the user's task, satisfying *context adaptation.*

2.1.3 Context information

What are the most common types of contextual information used by context-aware applications? How are them classified? In [48] Mizzaro, Nazzi and Vassena identified the following common types of information:

- spatial
- temporal
- social situation
- resources that are nearby
- physiological measurements

- schedules and agendas
- activities
- identity

Most of the applications already developed use only a small number of them. In particular, only the information that satisfy the requirements of the targeted project, technology available and environmental constraints are used. Despite this, in the last years the trend is to force the aggregation of much information as possible in order to provide more sofisticated and useful services to the users.

Bardram, Hansen and Soegaard, during their preliminary research study on the Aware Media project [4], suggested a classification that splits the types of information listed above along three main axes:

- Social awareness: 'where a person is', 'activity in which a person is engaged on', 'self-reported status'.
- Spatial awareness: 'what kind of operation is taking place in a ward', 'level of activity', 'status of operation and people present in the room'.
- Temporal awareness: 'past activities', 'present and future activities' that are significant for a person.

This classification group together different kind of data inside classes that describe social aspects regarding knowledge about a person, spatial aspects regarding information about a specific place and temporal aspects describing information about history and future plans of a subject.

2.2 Context-aware architectures

In this section is given an overview about different architectures, sensing infrastructures, approaches for modeling context, reasoning methods, quality and efficiency problems.

2.2.1 Main architectures

The three main architectures which guide the prototyping of a context-aware system suggested by Winograd [88] are: widget based, client-server and blackboard model.

The widget architecture [19] relies on the use of **context-widgets** which are sensor's interfaces used to retrieve in an easy way context data into applications. The interaction between applications and sensors goes through widgets by sending messages and receiving callback as soon as an information changes. Widgets can be thought as extensions of the sensor's drivers that provide a way to *hide the implementation complexity* required to communicate with them. The major advantage of this approach is that an application built on the top of the widget is completely decoupled with the specific implementation needed to interact

with a sensor. This means that even if the underlying technology used to sense the environment data changes, for example using Active Badges technology instead of floor sensors to retrieve location information, the result will not affect the whole application. Widgets can also provide an *abstraction* mechanism to provide data in a way that best suit the requirements of the application using it. For example a widget that provides information about the location of a user within a building, in most of the cases should not notify the application for any single location variation in the same room, but only notify it when the user move from one room to another. Moreover, widgets provide reusable and customizable piece of software. For instance, a widget used to track the location inside an application X which provides help during tour guides can also be used by an application Y which implements a car navigation system [26, 37].

The **client-server architecture** is usually used to enable communication between high-level components (e.g different applications) within a network. This model has been widely adopted for the development of internet based applications, using as communication protocol TCP/IP. Typically, in a client-server architecture a client finds the location of a service using a resource discovery component and then tries to connect to it in order to retrieve information needed (e.g the location of a user). The discovery component is one of the major drawback of this architecture because it has to route all the requests coming from the clients to the host that provides the requested service. Other problems affecting this architecture are network latency and available bandwidth which are critical factors for some context-aware applications due to their timing constraints [26, 37].

Unlike the previous approaches, the **black board architecture** is a data-centric model where all the components involved in the system post messages to a common shared message board in order to subscribe for receiving messages that match a predefined pattern (e.g change of location). All the communications pass through a centralized server (the blackboard) and the routing of messages is implicitly executed by the pattern matching between the message's content and the pattern specified inside the subscription. The major drawback of this approach is related to the communication efficiency because each piece of information requires two hops in order to reach the final application: the first one from the sensor to the blackboard and the second one from the blackboard to the application [26, 37].

2.2.2 Sensing infrastructure

The sensing infrastructure is one of the most important element of a context-aware system. In the last years due to the discovery of new materials, miniaturization techniques and increasing market demand, sensor technology has significantly improved making sensors more reliable, accurate and cheaper.

Indulska and Sutton [38] classified sensors in three main groups, according to

the way they capture context data:

- Physical sensors: hardware based sensor that retrieve phyisical data. Nowadays they are the most common devices used to gather information like motion of the body, location of a person, temperature level, sound level and human's body functions (biosensor) [37].
- Virtual sensors: are source of information coming from other applications and services. For example, the location of a user can be retrieved by looking at the emails or at the entries recorded inside electronic calendars [37].
- Logical sensors: these sensors combine information from virtual, physical sensors and databases in order to provide high-level information. For example, the location of a user can be inferred considering which is the computer used to log in and then combine it with the location of the machine within an office by looking into a database containing $\langle machine\_ID, position\_of\_machine \rangle$ pairs [37].

2.2.3 Context modelling

In order to retrieve in a meaningful way contextual information, a data model is often required to represent data. The most common modelling approaches are based on the data structures they use [37]:

- Key-Value model: this is the most simple data structure. It represents information using key-value pair, for example $< location, name\_of\_location >$.
- Markup scheme models: use hierarchical data structure consisting of markup tags that describe for each content its attributes.
- Graphical models: in this case contextual information is modelled by using a graphical model, such as UML or ORM (Object-Role Modelling).
- Logic based models: is one of the most formal approach available and it describes context using facts, expressions and rules. The inference process, which allow to infer new facts according to the rule previously defined, is performed by suitable reasoners.
- Ontology based models: this approach uses ontology description languages such as OWL or the less expressive RDF to describe concepts and data relationships. Due to the high expression capabilies, this approach is one of the most promising for future applications.

2.2.4 Reasoning methods

In order to infer new information from contextual data, a number of different reasoning methods are commonly used by some context-aware systems [43]:

- Artificial Neural Networks: are computational models capable to infer new information according to the network's structure trained during the learning phase. The output of a neural network try to approach as much as possible a particular pattern, hidden inside the contextual variables provided in input.
- Bayesian Networks: extract information using the probabilistic relationship existing between two or more different contextual variables linked toghether. The probabilistic model is able to provide the posterior probability of a variable, given the state of its linked parents.
- Hidden Markov Models: probabilistic model where contextual variables are assumed to behave as a markov process. This model gives the likelihood of a state considering the evolution of linked variables during the time. This kind of inference process requires a training phase that usually needs a large amount of data in order to build a reliable set of initial probability values.

2.2.5 Historical Data

For a context-aware application keeping historical data can be particularly useful. This is because the behaviour of the application could be adjusted during the time in order to provide more flexible services to users, according to the collected past values. Historical data can be used by computational models such as Artifical Neural Networks or Support Vector Machines to discover hidden patterns characterizing user habits. For example, if a user frequently follows a path inside a building, could mean that he is more interested to things located along such path and this information could be used to personalize the displayed information or provide detailed suggestions about related things around that area. Moreover, historical data can be used to analyze and extract statistics about the level of satisfacion of the provided services.

2.2.6 Quality and efficiency problems

During the design of a a context-aware application, designers need to consider a number of factors that could influence the quality of a system, both at performance and functionality level. The most important are [37]:

- Efficiency: for context-aware applications efficiency is a critical factor because they are often affected by tought timing-constraints. Although some architecture make possible to communicate in a fast way, other architectures are less efficient because due to their distribution of components, are forced to use different communication layers which could lead to a loss of efficiency. Therefore, during the design phase is important to choose the right architecture to best fulfull the timing requirements of the application to develop.
- Configurability: after the deployment, the system should be easily configurable and the job of adding and modifying compontents should be

achievable in a simple way, without compromising the stability of the system.

- Robustness: regards the level of difficulty in coping with breakdowns of the system. Programming languages provide several error handling mechanisms but they are only thought to cope expected and predefined errors. They do not consider errors that can be raised by the interaction between different components within the system, especially when they are distributed inside a network. A robust system should keep running also in case of malfunction, jam and unexpected data received.

- Simplicity: this is an important property that should be guaranteed by a well thought context-aware system. Programmers and maintainers do not have to struggle with huge implementation complexity in order to understand functionalities and keep the system working.

2.2.7 Open problems

Context-aware computing is still an open research area where models and architectures supposed to support context-aware applications have not been defined yet. As explained in [13] it is clear that the most difficult problem in developing such applications is not the access to technologies (sensors, handheld, etc) as they exist, but find the best way to model context data and architectures to support their use [13]. The two main challenges faced in the last years by researches are:

- Development of a taxonomy to uniform and standardize the representation of context types.

- Development of an infrastructure to promote the design, implementation and evolution of context-aware applications in an efficient way.

Other problems still unsolved are linked to security, reliability and privacy of data, especially in the medical domain (e.g identification, authentication, availability, integrity, confidentiality). For all these reasons different research path have been undertaken in the past and are still under investigation.

2.3 Application domains for context-aware systems

The availability of new technologies and devices providing new forms of interaction led the adoption of context-aware services in a growing variety of domains, some of them are [37]:

- Smart homes: the aim of context-aware applications is to provide useful services to home abitants in order to increase their quality of life and help disabled or elderly people to be more independent. The most common functions provided are: *security functions* to supervise the environment and send alerts detection in case of gas leakage, open tap, fire, flooding. If

needed these applications can also report critical information to the appropriate institution (eg. police or fire department). *Appliances functions* to provide control of appliances such as turning on/off the light, open/close window, turning on/off air conditioner or heater. *Supervising healthcare functions* to monitor the person's biomedical functions: glucose levels, blood pressure, heartbeats or provide reminders about daily medication [37].

- Airports: context-aware solutions are used to identify possible threats or emergency conditions providing automatic mechanisms aimed at delivering immediate security notification to the appropriate department such as maintenance, fire department or police. Services linked to passengers's behaviour have been developed as well. Most of them are able to send to the passenger's mobile device informations about shopping zones, exits, gates, arrivals and departures delay according to their location [37].

- Leisure/Entertainment: information provided, typically on mobile phone, is about nearby restaurants, theatres, festivals, events, shops and other data related to the area where the user is located [37].

- Museum: context aware applications are often used to detect user's position within a building in order to guide visitors through a predefined path. Typically, these applications are developed on suitable portable devices able to sense the location and capable to provide video/audio information relating paintings, statues and other object within a museum [37].

- Offices: services provided by context-aware systems are usually aimed at monitoring the status of the equipment and providing better allocation of human resources changing the shift schedules considering location and activity performed by workers [37].

2.4 Application domain: hospitals

One of the most promising application domain for context aware systems are hospitals. Hospitals are characterized by a strong communication infrastructure used to exchange a number of different kind of data such as patient reports, lab tests and working shifts. The management of this information is one of the most difficult challenges to face because it requires to take in consideration a wide variety of problems that should be avoided in order to properly meet the needs of hospital's professionals. Context-aware applications seem to be a valid solution which can also be used to shift part of the worker's activities to computers.

In the following sections different aspects characterizing communications within hospital's environment will be analyzed.

2.4.1 Contextual information

There are several information that clinicians need to know in order to carry out visits, lab exams, surgery and coordinate their activities [78]. This information however, is not often easily accessible because it is scattered in different places and known by different persons who are not always available and located in the same place. This tells us that in order to gain data access, physicians need to know something else: contextual information which allow them to know *when* and *where* a resource can be accessed.

Availability status is an example of contextual information. It enables a non-intrusive communication interaction and allows doctors to know when a colleague can be contacted for asking an information without interrupting his/her activity. Another important contextual information is location. Hospitals are higly dynamic environment where professionals working in different areas are always moving in different places [76]. Without a suitable mechanism able to trace their location, doctors are often difficult to find and in most of the cases the only way to get in contact with them is to use phones/pagers for asking where they are located.

It is worth to observe that availability and location are information strongly connected. Infact, when physicians are inside a patient room or inside an operating room they will be more likely performing a visit or an operation: in both cases the status will be clearly 'unavailable' and they do not expect to be disturbed.

Another interesting contextual information characterizing data access is the role covered by a person: hospital's professionals need different types of data, according to this information. When a physician goes near a patient's bed, for example, he will probably want to see the relating EHR record or the last lab test available for a patient; on the other hand a nurse near a patient's bed will be more interested to see the last medication procedure to be made.

Currently, some contextual information are shared on whiteboards where workers usually record their working shift, covered role and status information. An example of whiteboard is shown in Figure 1.

Figure 1: An example of whiteboard

2.4.2 Role based communication

It has been shown [51] that clinicians are often interested to send information or contact colleagues who covers a particular role, regardless the specific identity of the person to be contacted. For example, they may need to send a patient's test result to *'the doctor of the next shift'* or *'to the first nurse who is going to enter a specific room in the next morning'*.

Furthermore, some research studies [76] revealed that clinicians are frequently interested to contact the 'on-call person' who is responsible to receive all the calls directed inside a particular area (e.g operating room) or department. Even in this case they are not interested to the specific identity, but only on the person who cover this role.

2.4.3 Interruptions

One of the major drawback of the current hospital's communication infrastructure, discovered in different studies conducted on site [76], concerns potential interruptions raised by pagers or mobile phones.

Pagers are devices particularly interruptive because *when a page is placed, the recipient has to stop what he or she is doing, find a telephone and call the number on the pager*. Moreover *by the time this has been done, the caller my not be available any more* [74, 76].

Mobile phones are interruptive too and potentially can become more interruptive than pagers. According to T. Solvoll and J. Scholl [78]: *when the phone rings, the persons carrying it may feel obliged to answer and explain that they will call back, if they are busy.*

Interruptions are unwanted situation between clinicians because can cause lack of concentration in the activity performed and they should be minimized in order to avoid distraction that can lead to intolerable action or decision.

2.4.4 Communication technologies in hospitals

The most common comunication devices used in hospitals are pagers. Most of the hospital's workers use them to cover both personal and role based communication [78]. *The spread of these devices is such that many workers carry several of them according to the roles they have been assigned* [18, 76]. Pagers are simple personal telecommunication devices used to send short messages containing the caller's phone number that the recipient is then expected to call. Despite their simplicity and low cost, their asynchronous based interaction is often cause of delays between communication. As already said this happens because once a page has been placed, the recipient has to interrupt his/her activity, search a phone nearby and then call back the person who sent the page [78].

In the last years another communication device growing popularity in hospital setting are phones based on the Digital Enhanced Cordless Telecommunications standard (DECT). Some studies showed that thanks to their underlying technology are capable to simplify information access, increase the quality of patient care in the long run and *increase availability* through a synchronous communication channel not provided by pagers [2, 14, 74, 76]. They often offer a number of features such as programmable services, location tracking, alarms and the possibility to exchange user data not generally possible with standard phones. Despite their widespread use in every day life, in hospital environment are less utilized. The main reason is probably connected to the possible electromagnetic interferences with medical equipment [26]. However, researches demonstrated that in some situations the advantages seem to outweigh the risks [6, 42, 52] and mobile wireless phones are now more accepted [76]. For example, among others, the St. Olavs hospital (in Norway) uses a Cisco Imatis system which is a communication platform specifically thought for hospital environment based on wireless phones communicating over a VoIP network [94].

2.4.5 Context aware systems benefits

All the factors previously listed suggest that modern hospitals need an immediate requirement for systems able to enhance information and communication management. Context-aware applications can help to achieve this target introducing a number of benefits, some of them are:

- Reduce the efforts to gather information: applications can provide solutions to automatically send relevant information such as patient's report, working shifts and laboratory results when needed.
- Reduce the numbers of interruptions from mobile phones/pagers: according to a number of different contextual information such as availability and location, these applications are capable to manage when a particular device can be reached or not.
- Improve collaboration and coordination between workers: context-aware applications can increase awareness providing information about activities performed by colleagues and future plans.
- Provide a real solution for substituting the current hospital's pager-based communication infrastructure: context-aware applications can be used to join the functionalities of distinct role-based pagers in a single phone device which behave as a pager, eliminating the need to carry several of them simultaneously.

2.5 Existing context-aware solutions for hospitals

There are different context-sensitive solutions for hospital environment that have been developed in the last years, most of them are the result of a long research work conducted on site. Below, some of the main project carried out in this area are described.

2.5.1 Aware Media

The Aware-Media system [4], developed in the centre for Pervasive Healthcare at the University of Aarhus in Denmark, is a pervasive application thought to support close coordination and communication between clinicians. The system shows information on a number of large interactive touch screen displays scattered throughout the hospital. The information it provides is what kind of operation is currently executed in a specific ward, status of the operation, kind of doctors present in the room, stage of the operation through dynamic coloured bars and status of the work schedule (e.g delays or cancellations) provided displaying visual signs and text messages. Moreover, inside a little area of the display the application shows cues about what other people are doing, their location, status and future plans.

2.5.2 Personal Digital Assistants (PDAs)

The purpose of this study [1], carried out at the Academic Surgical Unit at St. Mary's Hospital (London), was aimed at verify whether PDAs with built-in mobile telephone could be an efficient solution to improve communication between hospital workers and compare them with pagers. These devices were also provided with electronic versions of commonly used UK medical reference text-books, drug interactions compendium, anatomy atlases, international classification of disease guidelines and medical calculators. During the assessment phase some Palm Tungsten PDA were given to a surgical team. The information used to evaluate the communication efficiency gained with these devices was the length of time clinicians needed to respond to a call. After 6 weeks of tests and questionnaire filled by people involved in the study, the results were encouraging because they showed a general benefit in replacing pagers with the new advanced PDA devices.

2.5.3 Mobile WARD

The aim of this study [73] was to evaluate a context-aware solution based on mobile phones capable to give nurses information about patients. The provided information considered daily tasks nurses had to deliver, timing constraints and position. Moreover, the mobile devices could also be used to insert collected data during daily work and look at previously stored patient's information in order to monitor changes. After the development, an assessment phase has been conducted. The problems identified concerned mainly the complexity of the automatic update mechanism of the devices: some subjects did not understand how to navigate between the different interfaces and they felt forced to undergo to the information displayed on the phone [73]. Others felt confused when suddenly the system changed the interface layout while they were reading information displayed [73]. Finally, some nurses expressed uncertainties about the validity of the data previously entered into the system and they wandered if the information had been saved correctly [73].

2.5.4 Context-Aware communication in hospital

This solution [51], carried out at IMSS General Hospital in Ensenada Mexico, uses handheld devices that allow users to specify when and where they want to send messages/data to other colleagues. For example, doctors can specify who will be the recipient of a patient's lab test result and automatically send it when ready. Moreover, with this system doctors can also send messages without knowing the names of the recipients by sending lab tests to any physician on duty for the next shift or to the first doctor who will enter a specified room in the next day. Fig. 2 (from [51]), shows a screenshot of the user interface from where users can select the location of the recipient required for the delivery of a message.

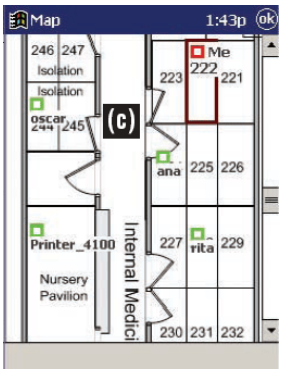

Figure 2: GUI from where the location of the recipient can be chosen, from [51].

2.5.5 Intelligent Hospital, QoS Dream Platform

This application, proposed by Mitchell [50], is based on touch-sensitive terminals ubiquitously scattered throughout the hospital. By using these terminals clinicians can, after an authentication process, request a video call to a colleague without knowing where the person they want to contact is located. The call is routed to the nearest terminal of the recipient who can choose to take the call or refuse it. The user's location is tracked by the application thanks to an Active Badge tracking system able to detect positions through badges worn by clinicians. The main application scenarios of this solution are: remote consultation between doctors (e.g discussions regarding patients and their treatments) and consultation of patient's data enabled by an event notification infrastructure that allows to push clinical data directly into the terminal's display.
Intelligent Hospital has been built to demonstrate a real application of the QoS Dream middleware platform. This platform supports context-aware, event driven applications and solutions based on multimedia contents where user mobility is a predominant factor [50]. The framework is composed by four main conceptual components: an operating system that offers resource management and overall control functionality, a dynamic multimedia streaming component based on the DJINN platform used to re-route video streaming contents ac-

cording to the movement of participants, an event-based infrastructure that uses HERALD architecture and a set of APIs for building applications using the technologies of the system [50].

2.6 A Context-Sensitive Mobile Phone: SenSay

In the following we are going to present an interesting prototype that even if it is not specifically thought for hospital environment, it is strictly related to the context-aware application discussed in this thesis.

SenSay is a context-aware mobile phone that adapts to dynamically changing environmental and physiological states [79]. It is capable to change the ringer volume, vibration, provide to callers feedback about the current context of the user's phone and make call suggestion to users when they are idle [79]. Contextual information is gathered by using the following sensors, mounted on different part of the body:

- 3-Axis accelerometers
- Bluetooth and Ambient microphone
- Light sensor

A central hub mounted on the waist, acts as a central component which receives and distributes data coming from the sensors to the decision logic module. The decision module analyze the collected data and determine the new state of the phone that should enter [79]. The four states provided by SenSay are the following: Uninterruptible, Idle, Active and Normal state. When the phone switches from one status to another, a number of settings are automatically changed.
The Uninterruptible state turn off the ringer and turn on the vibrate only if the light level is below a certain threshold. This state is entered when the user is involved in a conversation (recognized by the environmental microphone) or is involved in a meeting recorded inside the electronic calendar on the device. When a phone is in this state, all the incoming calls are blocked and feedback messages are sent to the callers. The callers have an option which allows to force the call in case of emergency. The active state is entered when high physical activity or high ambient noise level are detected by the accelerometer and by the microphones. In this case the ringer is set to high and the vibration is turned on. The idle state is activated when there is little movement and the detected sounds of the sorrounding environment are very low. When the phone is in this state, it reminds the user pending calls. In the normal state, the ringer and vibrate are configured to the default values.

3 Methods and Materials

The aim of this chapter is to provide the reader with a description of the materials and methods used for developing the context-aware application. After a description of the overall working framework, will be described the Ascom platform on which the application is built on, trixbox and Zimbra exchange server. The chapter ends describing the adopted software development methodology and how the tests of the application have been carried out.

3.1 Overall working framework

The overall framework on which the application relies is shown in Fig. 3. It is made up by different components communicating between them over a local area network. The blue squares represent the modules costituting the *Ascom Unite System*, the red square represents the IP-DECT wireless interface of the Ascom technology along with wireless phones and location sensors, the green square represents the PBX *trixbox* and the orange square *Zimbra Collaboration Suite* exchange server.

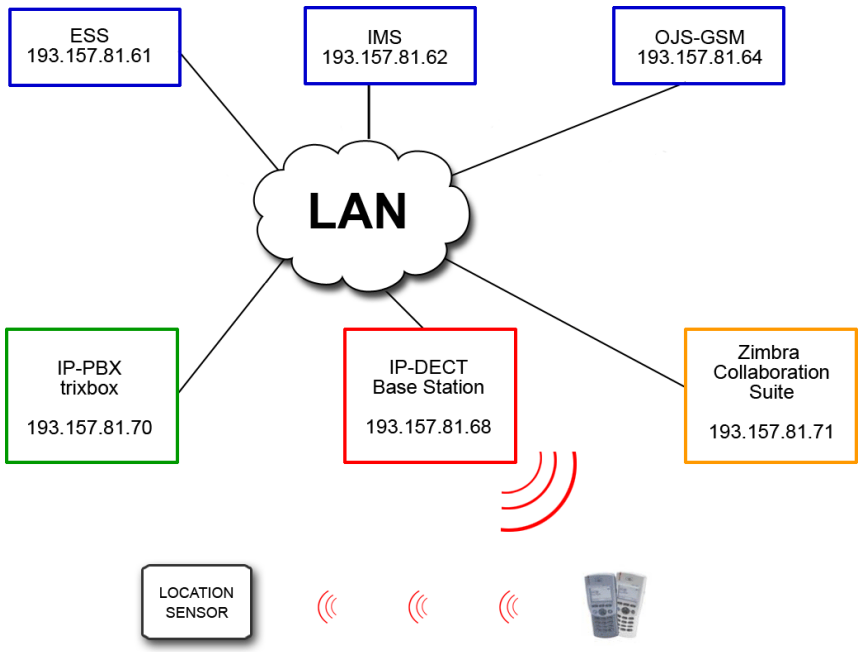

Figure 3: Framework architecture

Ascom Unite is an integrated mobile communication platform specifically thought to be used within hospital environment [26]. The modules constituting this system are seven: all toghether make it possible to manage information received by the IP-DECT base stations such as voice and data coming from the phones, over VoIP technology. In the picture above are only illustrated the three modules

needed (or replaced) by the context-aware application: the Integrated Message Server (IMS), the Open Java Server (OJS-GSM) and the Enhanced System Services (ESS).

trixbox is one of the most important component of the framework because it provides functionalities useful to manage calls. In particular, it is capable to intercept all the calls passing over an IP network and it allows to customize their behaviour by simply programming the associated dialplan.

Zimbra Collaboration suite is an exchange mail server that provides a number of advanced features to manage emails, contacts, tasks and personal commitments. It exposes services to external applications through a SOAP interface [96] that can be used to gather information stored by the users.

In the following we are going to discuss in details these components, starting with the Ascom system.

3.2 Ascom Unite System

The Ascom Unite system provides a series of modules connected to each other containing applications used to deliver services over an IP network. Its main purpose is to provide a reliable and a flexible architecture independent from the adopted radio carrier technology: in this way it is possible to integrate wireless phones, pagers, GSM phones in the same system and provide them common functionalities.

All the available modules are tightly integrated and they exchange data using the proprietary *Unite* protocol built on the top of the TCP/IP stack. Each module runs on an hardware platform called ELISE (Fig. 4) and contains: a host router which handles all the communication to, from and internally in the module, a UNS (Unite Name Server) used to translate the call IDs into internal addresses, a Web Server that provides a web-based user interface for the module's configuration, a Linux based operating system and a Host Attendant which handles the basic configuration and supervision of the installed software [26]. Each module contains specific software, marked with X and Y in Fig. 4, required to carry out specific functionalities.

All toghether, the modules constituting the Ascom Unite System, provide the following services [26]:

- Messaging: enables external applications to send text messages to a specific destination (e.g a wireless phone or a GSM phone).
- Interactive messaging: enables external applications to send particular messages containing different response options.

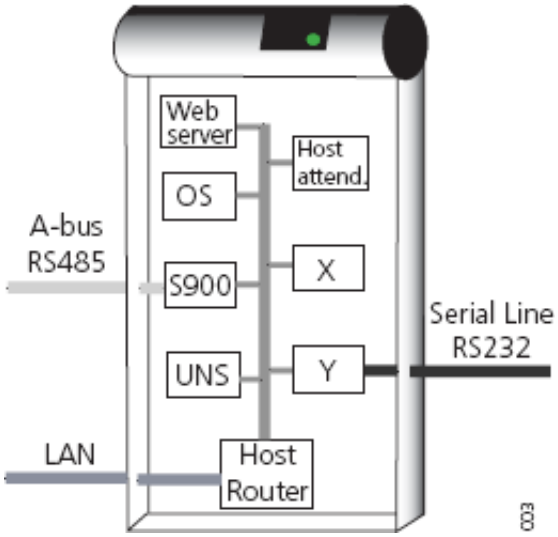

Figure 4: ELISE hardware, [TD 92243GB] pg. 7

- Personal alarm handling: enables the reception of alarm events from portables.
- User data: enables external applications to receive user data from portables.
- Remote management: it allows the management of all the modules of the Ascom system over an IP network.
- ESS centralized services: number planning, message routing, group handling, system supervision, fault and activity logging.

In the following will be given a brief description of the modules used (or replaced) by the context-aware application, the wireless phones, the IP-DECT station and the location sensors.

3.2.1 Enhanced System Services - ESS

The ESS module represents the central unit of the Ascom Unite System: it manages all the calls between phones. This component *will be replaced* by the context-aware application with the PBX trixbox in order to override the standard call management. It is worth, however, to take a look at the functionalities it provides [26]:

- Number Planning: by using the web-based interface provided it is possible to manage the centralized number planning and handle phones and pagers regardless the carrier system on which they are connected to.
- System Supervision: enables the supervision and control of all the modules installed in the system.

- Group Handling: portables from different carrier systems can be grouped together within the same category. This functionality is useful to apply the same policy/rule for a whole group of phones.
- Fault Handling: faults from other Unite modules can be collected and stored inside a dedicated log file.
- Activity Logging: collects inside a log file events, alarm data, messages and location data sent by the phones.

3.2.2 Integrated Message Server - IMS

This is a middleware between the IP-DECT base stations and the other modules of the Ascom Unite system. It handles all the data coming/directed to the phones and supports the following services [26]:

- Message distribution: it allows to specify where information coming from the phones has to be distributed. For example, *location or user data can be distributed to the Open Java Server.*
- Central Phonebook: a central phonebook accessible from all the phones can be stored inside the module.
- IMS Messaging Tool: is a web-based tool for sending messages to the handsets.

3.2.3 Open Java Server - OJS-GSM

Open Java Server is a programming server directly interfaced with the IMS. This module allows to mount a Java program on it implementing customized features not covered by the standard Ascom Unite System. The different kind

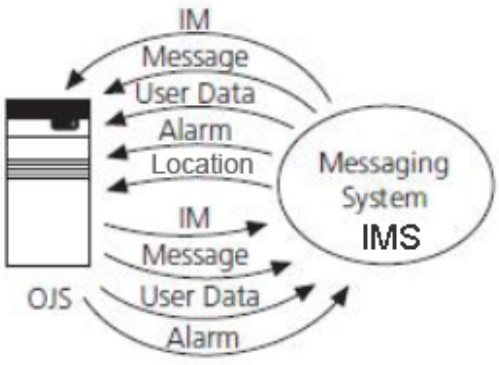

Figure 5: OJS interactions with the IMS Messaging System, [TD 92230GB] pg. 1

of data (shown in Fig. 5) that this server can receive/send from/to the IMS are:

- Standard Messages

- Interactive Messages (IM) with multiple response options.
- User Data (*e.g sent by the phones when one of the three soft keys is pressed*)
- Location
- Alarms

In order to communicate with the IMS messaging system, each program mounted on this server must use the OAJUtil Java package provided by the module. OAJUtil is an API containing specific libraries that can be used to manage all the information listed above.

3.2.4 IP-DECT Base Station

The IP-DECT Base Station, shown in Fig. 6, works as a bridge for data exchanged between the Ascom wireless phones and the Ascom Unite wired network. Its main characteristics are the following [26]:

- DECT GAP/CAP radio interface
- Supports H.323 or SIP protocol over IP
- On-air synchronization
- Web interface for configuration and software upgrade
- Roaming and handover
- handling of 8 simultaneous calls

Figure 6: Ascom IP-DECT Base station, [TD 92370GB] pg. 1

Therefore, all the communications between handsets and the Base station are carried out over the DECT protocol.

3.2.5 Handsets

The two models of phones provided by the Ascom equipment at our disposal are the 9d24 MkII and d62, shown in Fig. 7. The d62 differs primarily for its intuitive and coloured display not available on the 9d24 MkII. Both provide a

Figure 7: Ascom's wireless phones: the 9d24 MkII on the left and the d62 on the right, [92380GB] pg. 1

number of advanced functionalities such as built in alarms, SIM card for identity and personal settings, advanced messaging functions, up to 10 profile mode with customizable settings and *3 programmable soft keys for each profile which can be used to push user data to the Ascom Unite system*, highlighted with red squares in Fig. 7.

3.2.6 Location devices

The Ascom equipment is also provided with location devices used to track handset's positions. Every time the position of a phone changes and fall into an area covered by a specific location device, the ID code of the sensor is sent by the phone to the base station.

3.2.7 Data and call flow within the Ascom system

Before continuing with the description of the other framework's components, in order to make clear to the reader which components of the Ascom Unite System are involved in a typical communication scenario during the sending of data or the execution of a call between phones *without considering the context-aware application*, a brief explanation will be now given.

- The data flow is shown in Fig. 8. When a new message containing location or user data is sent from a device, it is first received by the IP-DECT base station. Then, the base station routes data towards the Integrated Message Server (IMS) which finally distributes the information to the

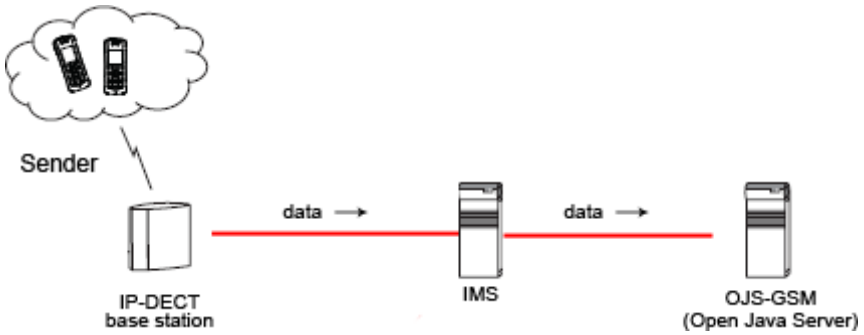

Figure 8: Data flow

Open Java Server GSM. In order to route the data in this way, the Ascom system has been specifically configured. Its configuration is given in the Appendix B.

- The flow followed by calls is slightly different from the previous one and it is illustrated in Fig. 9. All the incoming calls are routed by the IP-DECT

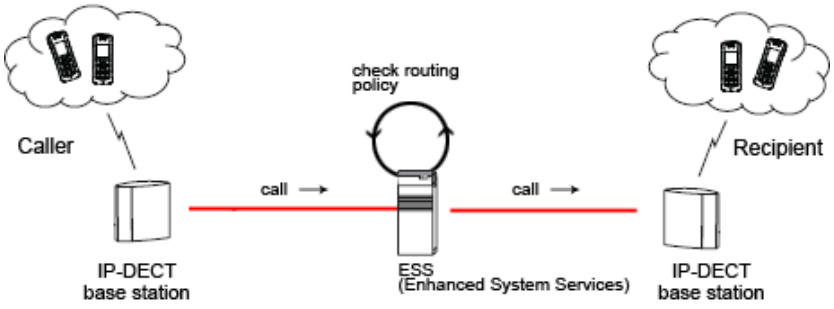

Figure 9: Call flow

base station to the Enhanced System Services component. ESS checks if there are routing policies to be applied and then, according to the outcome, deliver the call to the right recipient. The call flow just described will be completely modified by the context-aware application in order to properly manage the control of the phone calls.

3.3 trixbox

trixbox is an IP PBX system based on Asterisk. A Private Branch eXchange (PBX) is a middleman between the phone company and the extensions within an office [30]. Basically, trixbox provides web-based interfaces and a number of tools to easily maintain and configure the underlying Asterisk engine. It is made up by the following components [26]:

- CentOS 5.2: the core operating system. It is a community supported version of the Red Hat Enterprise Linux distribution.
- Asterisk 1.4: the Private Branch eXchange engine with full customization capabilities.
- Free PBX 2.5: a graphical user interface which allows an easy management of the configuration files on Asterisk.
- Flash Operator Panel (FOP): a graphical user interface designed to help call management. From this panel, a user (tipically a receptionist) can look at the status of all the extensions subscribed in the system and manage them if necessary.

One of the most important element of trixbox is the Asterisk's dialplan because it defines how to handle inbound and outbound calls. It consists of a list of command/instruction that Asterisk executes during a call directed to an extension, where an extension is a numeric identifier given to a line ringing a particular phone. The Aterisk's dialplan is specified inside a configuration file named *extensions.conf* and the four main sections characterizing it are context, priorities and applications. Contexts are named group of extensions containing several instructions. The instruction's execution order is estabilished by the priority and the action performed depends on the application called from each instruction. Some examples of applications are *answer*, *hangup*, *park* a call, *hold* a call and *say* a predefined message.

Asterisk Gateway Interface (AGI) is a programming interface for Perl, PHP, C and Pascal that allows a third party application to dinamically change the dialplan when a new call is made. A program that uses this interface (stored in the same Asterisk's host) can be directly called from the dialplan through suitable instructions.

FastAGI [95] is a Java implementation of the Asterisk Gateway Interface (AGI) that uses TCP sockets to communicate over TCP-IP with Asterisk. By using the FastAGI API within a Java program it is possible to remotely control the dialplan (and therefore the call's behaviour), over TCP-IP protocol.

3.4 Zimbra

Zimbra Collaboration Suite is an enterprise open source email server running on Linux Ubuntu Server 64 bit. The features it provides are: rich email, contact management, document sharing, document management, integration with common mobile phones such as iPhone, BlackBerry, Android and a calendar tool shareable between users accessible through a standard web browser.
Zimbra exposes services throught its SOAP [96] interface. SOAP requests, carried out over a TCP-IP connection, can be used to extract data stored inside the server by the users. For example, by using suitable SOAP requests it is possible to extract emails, contacts and appointments records for a specified user.

3.5 Hardware/software versions

The application relies on the following hardware and software versions:

- Ascom Unite platform hardware: IP-DECT Base Station v3.1.23, Handsets models: 9d24 MKII v3.71 and d62 v2.8.22, IMS2 v2.36, OJS v3.0.
- Machine running the application: Primergy TX100 S1 server, dual processor Intel Xeon CPU X3220 @ 2.40 Ghz, 2.39 Ghz, 4 GB RAM. Operating System: Microsoft Windows Server 2003 Standard x64 edition provided with VMWare-Server v2.0.2.
- trixbox v2.8.0.3.
- Ubuntu Server 10.04 LTS 64 bit.
- Zimbra Collaboration suite version 7.0.0 installed on Ubuntu Server.

3.6 Engineering approach

The software engineering approach used to develop the context-aware system is based on the Unified Process, an iterative and incremental development methodology (also known as spiral development or evolutionary development) based on the ideas of Boehm [9] and Gilb [29]. This approach split the development process into a series of short mini-project, called iterations. The purpose of an iterative approach is to increasingly enlarge and refine a system at each iteration in order to gradually approach the requirements of the targeted application. Usually, an iterative model does not start with a full specification of the requirements but begins specifying and implementing only the most important features that are subsequently imporoved and adjusted to cope missing requirements during the next iterations. The steps composing an iteration are the following:

- Requirements: the requirements are identified, collected and analyzed.
- Design: a software solution is designed by using Use Cases diagrams to capture the functional requirements, Interaction diagrams to define the interactions between software components and other graphical UML notation models to better define the overall architecture of the software to develop. The result of this phase is an architectural model describing the implementation goals for the current iteration.
- Implementation: the aim of this phase is to code the software described in the previous step, improving the system already developed.
- Testing: the new developed features are tested in order to verify if they are consistent and do not present errors.

After these steps, if the requirements are not satisfied a new iteration takes place.

3.7 Tests

The tests have been carried out by simulating typical scenarios where the functionalities offered by the application can be involved. After each scenario we asked testers what they thought about the features just used, without guiding too much their evaluation. With this simple approach we hope to have obtained sincere considerations and more useful feedback.

4 Software Requirements Specification

The purpose of the following specification is to present a description of the application, the assumptions under which it operates and its external behaviour through an illustration of the functional requirements.

4.1 Description

The context-aware solution, based on an Ascom/trixbox communication platform, is aimed at managing the balance between increased availability and increased interruptions gained with the Ascom wireless phones. Interruption management is performed by applying to calls a number of rules that take in consideration status, location and personal commitments of the users carrying the devices. Moreover, the application implements a feature that allows a phone to behave as a pager, it is capable to send feedback messages explaining the unavailability cause of an unreachable phone and send 'pending call' messages containing the name of the persons who tried to call a user during the unreachable status.

4.2 Assumptions

The application relies on the following assumptions related to behaviors that users are expected to follow in order to ensure that the system will work properly:

- Users always carry their phones with them: without this assumption would be impossible to trace their location because the phone's location could not reflect their real position.
- Users always update their availability status from the phone, when it changes: as in the previous point, if users do not change their status from the phone it is not possible to know their real availability.
- Users use Zimbra's calendar to register their appointments and on-call shift schedules: if meetings or shifts are not correctly registered inside the calendar it is not possible to properly manage reachability of phones and diversions of calls.

4.3 Functional Requirements

Here are reported the use case describing the functional requirements of the application. For each use case are given: actors involved, brief description, preconditions, main flow, alternative flow and a priority measure from 1 to 3 which describes the importance of the use case in meeting the goal of the application.

4.3.1 Use case 1: manage call (location based)

This is a high priority use case because it partially covers one of the most important points of the problem statement: avoid an interruption when the recipient of a call is inside a critical area [1].

Use Case Name	Priority	Description	Actors
UC1	1	describes the process for managing a call considering the location of the recipient	caller, recipient

UC1 Preconditions: A new call has been started by the caller.

UC1 Main Flow:

- If the recipient is not inside a critical area the call goes through.
- If the recipient is inside a critical area (e.g surgical room) the call is blocked and the caller is prompted to press the button number 1 to let the call go through:

  1. If button 1 is pressed within 4 seconds:
     (a) The call goes through.
  2. If the pressed button is different from 1:
     (a) hang up.
     (b) The caller receives a message containing the location of the recipient.
     (c) The pending call is stored inside the recipient's profile.
  3. After 4 seconds of user inactivity:
     (a) hangup.
     (b) The caller receives a message containing the location of the recipient.
     (c) The pending call is stored inside the recipient's profile.

UC1 Alternative Flow:

- If the caller hangup before the prompt message:

  1. The caller receives a message containing the location of the recipient.
  2. The pending call is stored inside the recipient's profile.

[1] Critical areas are places within the hospital defined 'critical' such as surgical theater, conversational rooms, examination rooms.

4.3.2 Use case 2: manage call (availability status based)

This is another high priority use case: avoid an interruption according to the availability status of the recipient.

Use Case Name	Priority	Description	Actors
UC2	1	describes the process for managing a call considering the availability status of the recipient	caller, recipient

UC2 Preconditions: A new call has been started by the caller.

UC2 Main Flow:

- If the recipient is available the call goes through.
- If the recipient is not available the call is blocked and the caller is prompted to press the button number 1 to let the call go through:
  1. If button 1 is pressed within 4 seconds:
     (a) The call goes through.
  2. If the pressed button is different from 1:
     (a) hang up.
     (b) The caller receives a message containing the availability status of the recipient.
     (c) The pending call is stored inside the recipient's profile.
  3. After 4 seconds of user inactivity:
     (a) hangup.
     (b) The caller receives a message containing the availability status of the recipient.
     (c) The pending call is stored inside the recipient's profile.

UC2 Alternative Flow:

- If the caller hangup before the prompt message:
  1. The caller receives a message containing the availability status of the recipient.
  2. The pending call is stored inside the recipient's profile.

4.3.3 Use case 3: manage call (pager mode)

The assigned priority for this use case is 1. This is because it has a great impact on the application's goal: it describes how to model the behaviour of a pager inside a phone. The benefit of this functionality is that once a page has been placed, the phone can be directly used to call back the person who put the page

without the need to search a phone nearby.

Use Case Name	Priority	Description	Actors
UC3	1	describes the process for managing the behaviour of a phone as a pager	caller, recipient

UC3 Preconditions: A new call has been started by the caller.

UC3 Main Flow:

- If the recipient's phone is not switched to 'pager mode' the call goes through.
- If the recipient's phone is switched to the pager mode the call is blocked and the caller is prompted to press the button number 3 to page the recipient:

    1. If button 3 is pressed within 4 seconds:
        (a) The recipient receives a message containing a page.
    2. If the pressed button is different from 1:
        (a) hang up.
    3. After 4 seconds of inactivity:
        (a) hangup.

UC3 Alternative Flow:

- If the caller hangup before the prompt message:

    1. hangup.

4.3.4 Use case 4: manage call (calendar commitment)

This use case describes the process for managing an interruption according to the user's commitments stored inside Zimbra's calendar. The selected priority is 2. The rationale uderlying this choice is that the time spent by physicians in meetings is less than the time spent inside critical areas or for visiting. The impact of this feature on the application's goal is therefore lower than the previous.

Use Case Name	Priority	Description	Actors
UC4	2	describes the process for managing a call considering the calendar committments of the recipient	caller, recipient

UC4 Preconditions: A new call has been started by the caller.

UC4 Main Flow:

- If the recipient is not involved in a meeting recorded inside Zimbra's calendar the call goes through.
- If the recipient is involved in a meeting recorded inside Zimbra's calendar the call is blocked and the user is prompted to press the button 1 to let the call go through:

    1. If button 1 is pressed within 4 seconds:
        (a) the call goes through.
    2. If the pressed button is different from 1:
        (a) hang up.
        (b) The caller receives a message explaining that the recipient is in the middle of a meeting.
        (c) The pending call is stored inside the recipient's profile.
    3. After 4 seconds of inactivity:
        (a) hangup.
        (b) The caller receives a message explaining that the recipient is in the middle of a meeting.
        (c) The pending call is stored inside the recipient's profile.

UC4 Alternative Flow:

- If the caller hangup before the prompt message:

    1. The caller receives a message explaining that the recipient is in the middle of a meeting.
    2. The pending call is stored inside the recipient's profile.

4.3.5 Use case 5: manage call (routing a call to the on-call person)

This use case is an extension of the use case UC1. Since it is a feature higly recommended because it reflects the on-role based communication between clinicians, but it is not essential for reducing interruptions, the assigned priority is 2.

Use Case Name	Priority	Description	Actors
UC5	2	describes the process for managing a call allowing the caller to route the call to the on-call person, if the recipient is located inside a critical area	caller, recipient, on-call person

UC5 Preconditions: A new call has been started by the caller.

UC5 Main Flow:

- If the recipient is not inside a critical area the call goes through.
- If the recipient is inside a critical area (e.g surgical room) the call is blocked and the caller is prompted to press the button 2 to route the call to the on-call person:

    1. If button 2 is pressed within 4 seconds:
        (a) The call is routed to the on-call person with a current shift stored inside Zimbra's calendar.
    2. If the pressed button is different from 2 and 1 (because if 1 the call is forced according to the UC1):
        (a) hang up.
        (b) The caller receives a message containing the location of the recipient.
        (c) The pending call is stored inside the recipient's profile.
    3. After 4 seconds of user inactivity:
        (a) hangup.
        (b) The caller receives a message containing the location of the recipient.
        (c) The pending call is stored inside the recipient's profile.

UC5 Alternative Flow:

- If the caller hangup before the prompt message:

    1. The caller receives a message containing the location of the recipient.
    2. The pending call is stored inside the recipient's profile.

4.3.6 Use case 6: pending calls

This use case define the conditions under which the collected pending calls have to be delivered to a user. Even if this is a useful feature to improve collaboration between users, it doesn't have a great impact in reducing interruptions or improving role based communication. This is a priority 3 use case.

Use Case Name	Priority	Description	Actors
UC6	3	describes the process for sending pending call messages when user location or availability status changes	user

UC6 Preconditions: A phone changed its location or a user changed the availability status from the device.

UC6 Main Flow:

- If the new location is not a critical area or the new status is 'available':

    1. The user receives messages containing pending calls collected when he was inside a critical area or not available.

5 Construction

This chapter starts describing the data flow and the call flow characterizing the system, then describes the design of the software architecture, the classes that make up the application and finally the implementation of the features specified in the requirement specification.

5.1 Data flow and Call flow

The two main flows of information involving the context-aware application are illustrated in Fig. 10: the red line describes the path followed by calls, the black line describes the path followed by data and the coloured stations represent the hardware components of the architecture.

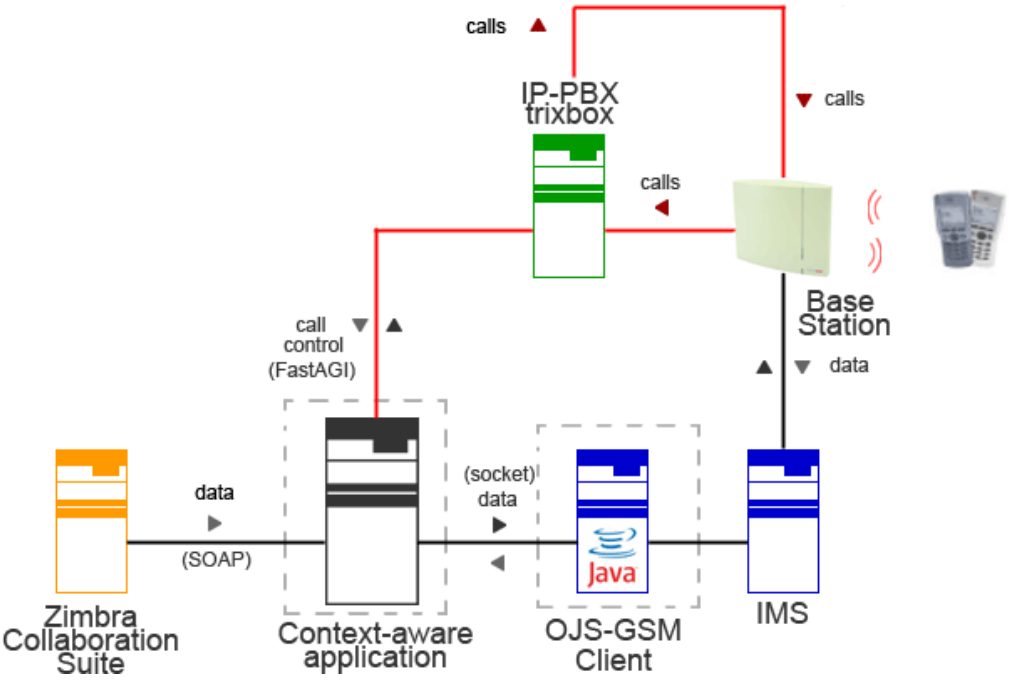

Figure 10: Information flows

The data flow from the phones up to the Open Java Server is the same as that one previously illustrated during the general description of the Ascom Unite System (section 3.2.7): when a new message containing location or user data is sent from a phone it is first received by the base station, then is propagated to the IMS and then to the Open Java Server. A customized Java program mounted on the latter that we will call *'client'* in the rest of the discussion, enables the final communication path towards the context-aware application. It routes data received by the OJS from the IMS and implements specific methods exploiting the functionalities offered by the underlying Java server that allow to

send messages and data back to the phones. All the communications between the client and the application are handled through TCP-IP sockets.
The data flow between Zimbra and the context-aware application is carried out using suitable SOAP requests sent over TCP-IP protocol. When some information is needed from the Zimbra server, the application estabilishes a new TCP-IP connection with it, send an XML-formatted request (the SOAP message) and wait for the response. Once the XML-response has been received, the XML is parsed in order to extract the information previously requested.

The call flow, highlighted with a red line in Fig. 10, is quite different compared to the path followed by calls inside the Ascom Unite system already described (section 3.2.7). This is because trixbox completely replaces the functionalities of the ESS module. When a new call starts from a phone it is immediately sniffed by trixbox as soon as it goes out from the base station, its control is left to the context-aware application which manage the call's behaviour by using the FastAGI interface.

5.2 Software architecture

Fig. 11 shows the overall software architecture of the system: on the left it is illustrated the Java client mounted on the OJS and on the right the software components constituting the context-aware application. In the following, a description of each component will be provided.

- *Java client*: is an event-driven component running on the Open Java Server which acts as a bridge between the server itself and the context-aware application. Every time new data coming from the phones reaches the OJS, it opens a new socket connection with the *Event Listener* in order to push the information just received. It also provides functionalities exploiting the API libraries offered by the underlying OJS (the OAJUtil package), used to send messages and data back to the phones.

- *Event listener*: is a listener which manage all the incoming connections coming from the *Java client*. According to the message received during each connection it interacts with the *Context Manager* in order to execute suitable operations.

- *Message Service*: is a component directly interfaced with the *Java client*. It allows the other components of the context-aware application to send messages and data directed to the phones.

- *Store Service*: it is responsible to save all the data that needs to be stored inside the application's database.

- *Retrieve Service*: the main task of this component is to retrieve information from the Zimbra server and from the application's database.

- *Context-manager*: is one of the most important part of the context-aware application. It represents the core of the system and acts as a middleware

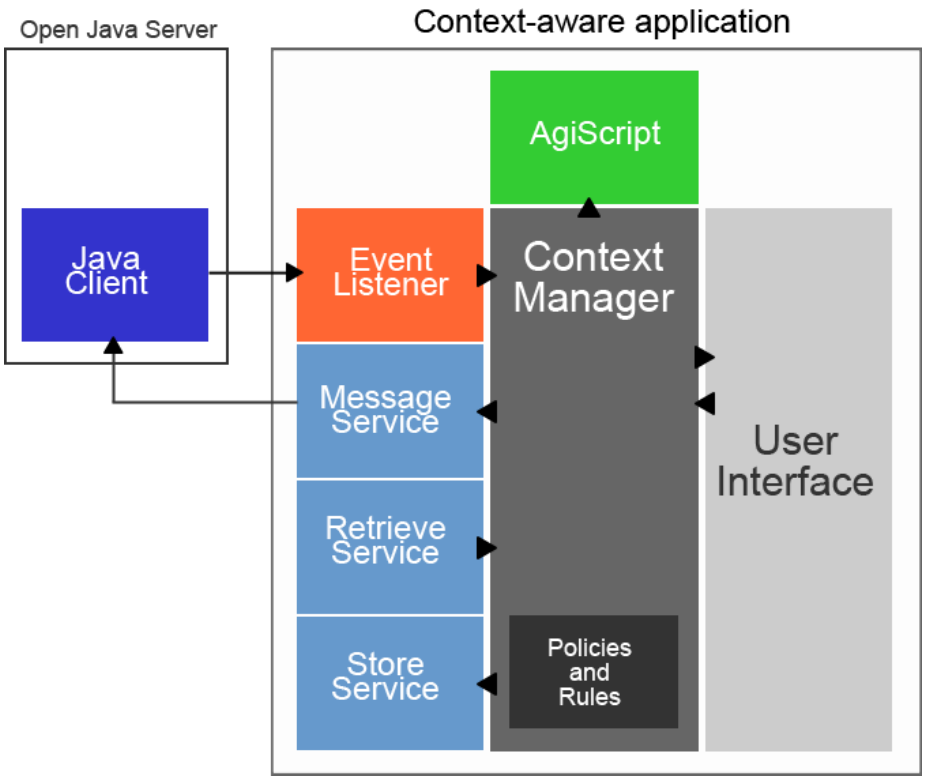

Figure 11: Software architecture

between all the other components. It communicates with the message service to send messages to the phones, with the event listener to receive data from the phones, with the store service to save different kind of data, with the retrieve service to gather stored data and with *AgiScript* to provide information about location and availability status of the users. In addition, it applies policies and rules when its functionalities are called by the other components. Policies are conditions which determine the behaviour of the context-aware application that can be configured from the user interface. Example of policies are enable/disable the reception of feedback messages and enable/disable the reception of pending calls for a specific user. Rules are variables which describe if a specific location is a critical area or not.

- *User Interface*: represents the user interface of the application. It is directly interfaced with the context manager which provides to it information to be displayed such as location and availability status of the users.

- *AgiScript*: this is the piece of software called by trixbox every time a new call takes place. It implements the logic necessary to remotely manage phone calls by using the FastAGI library. It also communicates with the context manager in order to collect information about availability status

of the users, their location and (when necessary) send messages to the phones.

Summing up, all the components composing the architecture are matched together by the *context manager*. It acts as intermediary between the *Event Listener* and the *User Interface* when location or status availability received from the former needs to be updated on the GUI and inside the application's database. It also acts as a middleware between *AgiScript* and the other components in order to provide data retrieve and messaging services.

5.3 Class diagrams

5.3.1 Context-aware application

In the following we are going to discuss the classes composing the context-aware application, illustrated in Fig. 12. The lines represent the relationships and the numbers at the ends of each line represent the cardinalities of the association. Even if the structure of the software is much more complicated than that shown in Fig. 12, only the most important methods of each class will be discussed. The other methods, though important, are not strictly necessary to understand how the application works. For further consultation, the full code is reported in the attached CD.

Run class:
This is the entry point of the program which initializes the main classes of the application. The only method belonging to it is main(): it creates instances of the *ContextManager*, *GUI* and *EventsListener* when the application starts.

EventsListener & ClientWorker classes:
These two classes implement the functionalities of the EventListener component, illustrated in the previous section. The run() method belonging to the *EventListener* class implements a socket server always listening for incoming connection from the *Java client*. When a new connection is estabilished with the socket, it creates a new thread *ClientWorker* in order to leave the control of the connection. According to the information received from the *client*, *ClientWorker* calls suitable methods from the *ContextManager* for executing specific operations (e.g if the message received contains the updated location of a phone, it calls the method newLocation()).

ContextManager class:
This is the central class of the whole application. As illustrated in Fig. 12, it contains a number of private and public methods. statusChange() and newLocation() are public methods called by *ClientWorker* in order to trigger all the operations needed to do when the availability status or location of a user changes. The method checkNextPendingCall() is called when a message containing a pending call has to be sent to a specific phone. isLocationOk() and isStatusOk() are called by *AgiScript* in order to know if the availability status of a user

is 'available' or if the location of a user is not a critical area. These methods return a boolean value that is true if the location (or the status) is not critical (available), false otherwise. It is interesting to analyze how they work: first they retreive location or availability status of a user from the retrieve service using the private methods getLocation(), getStatus() and getZimbraCalendarStatus(), then they apply rules in order to check if the information just gathered is 'critical' or not.

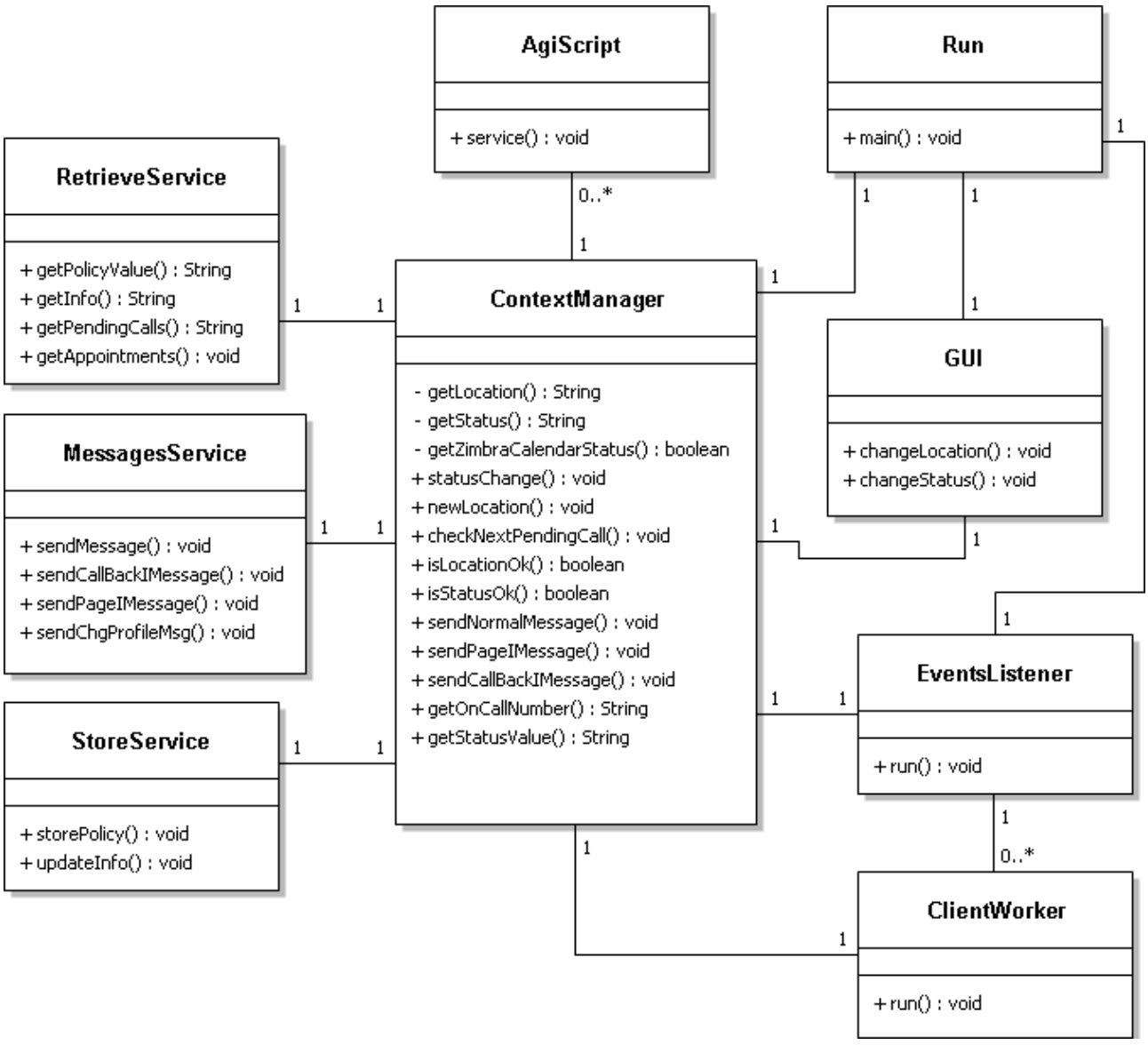

Figure 12: Classes of the context-aware application

For example, if the retrieved location (the ID-code) for a user X is '0101', the

method isLocationOk() first checks if '0101' is associated with a critical area and then returns true or false according to the outcome of the check.
sendNormalMessage(), sendPageIMessage(), sendCallBackIMessage() are used by *AgiScript* to send different kind of messages to the phones. The implementation of these methods is similar to the one previously described: first they check the policy values associated with the recipient in order to estabilish if a message can be sent or not and then they proceed (or not) with the sending of the message by calling suitable methods from the *MessagesService* class.
The method getOnCallNumber() is called by *AgiScript* in order to discover who is the current on-call person to whom a call must be routed when the recipient of a call is located inside a critical area. By using the *RetieveService* class, this method returns the current person responsible to receive routed calls, according to the on-call shift schedule stored inside Zimbra's calendar.

GUI class:
The most important methods of this class are changeLocation() and changeStatus(): when called they change the icon and the text string containing the location or availability status of a user listed inside the GUI.

AgiScript class:
This is the class called via remote connection every time a new call is sniffed by trixbox. The service() method is its starting point and contains all the instructions needed to manage calls and needed to handle exceptions that can be raised by unexpected conditions (e.g early hang up of a call).

RetrieveService class:
It contains three main methods: getPolicyValue() which searches inside the database of the application a policy entry associated to a user and returns its corresponding value, getInfo() is a method that retrieves both location and status availability of a user according to the arguments passed in input and getPendingCalls() which returns a string containing all the pending calls associated to a user.

MessagesService class:
It implements four methods used to give instruction to the *client* (through suitable text strings) in order to send different kind of messages directed to the phones. Each method contains the code required to open a socket connection and send strings like the following:

$< MSG >$
$< TYPE >$ Type of message $< /TYPE >$
$< HDR >$ Subject of the message $< /HDR >$
$< BODY >$body of the message$< /BODY >$
$< DEST >$recipient of the message$< /DEST >$
$< /MSG >$

In this way, according to the text string inside the $< TYPE >$ field, the *client* is able to determine which kind of message has to be pushed to the phones, the subject, the body and the recipient's number, all specified inside the tags $< HDR >$, $< BODY >$ and $< DEST >$.
The method sendMessage() is used to send a normal message, sendCallBackIMessage() to send an interactive message containing multiple option responses, sendPageIMessage() to send a page and finally sendChgProfileMsg() to change the profile mode on a phone.

StoreService class:
The are two methods belonging to this class: storePolicy() and updateInfo(). storePolicy() is called to store inside the application's database the policies values changed from the GUI of the application and updateInfo() is called to update location and availability status of the users.

5.3.2 Open Java Server: client

In this section will be described the classes composing the *client* mounted on the OJS, illustrated in Fig. 13.

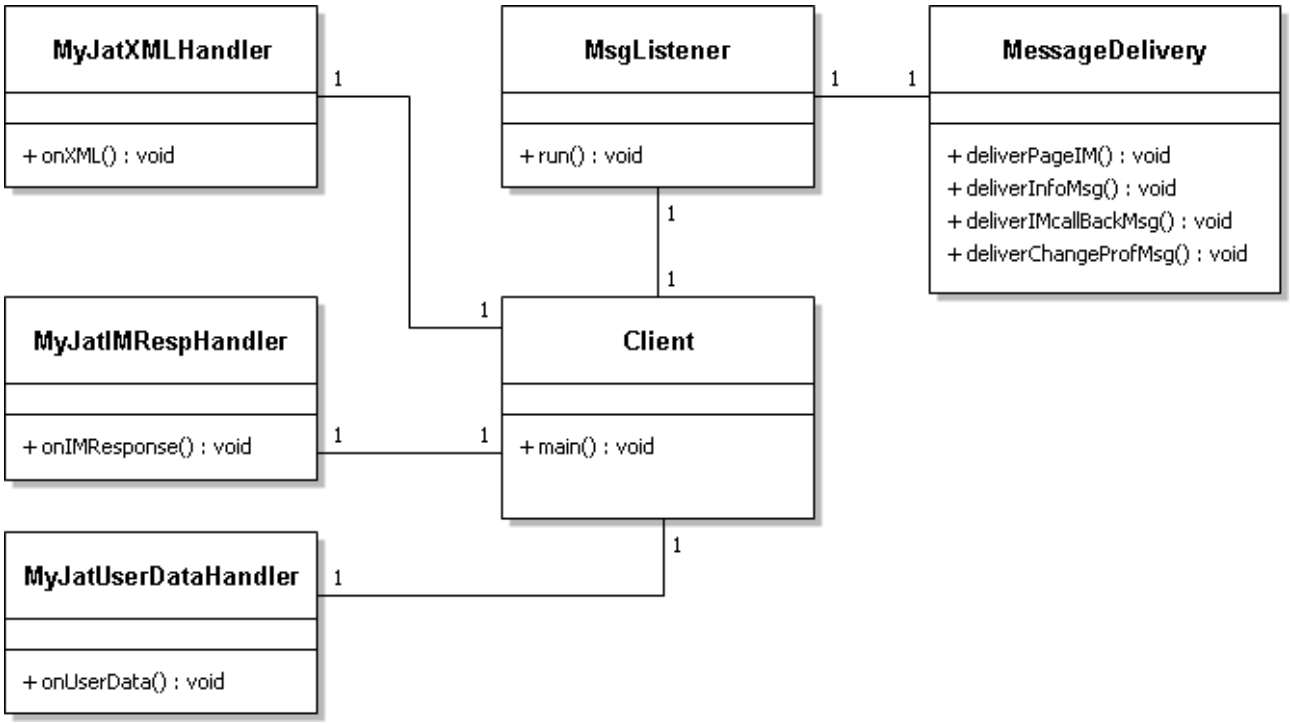

Figure 13: Client classes

Client class:
It creates all the instances of the classes needed by the *client* application. In particular, the main() method inizializes the listeners used to receive data coming from the phones and from the context-aware application, which are: MyJatXMLHandler, MyJatIMRespHandler, MyJatUserDataHandler and MsgListener.

MsgListener class:
It implements inside the run() method a server socket always listening for incoming connections coming from the *MessagesService* belonging to the context-aware application. According to the content of the text string received during each connection (described in the previous section), it calls suitable methods from the *MessageDelivery* in order to deliver messages matching the type specified inside the field $< TYPE >$.

MessageDelivery class:
By exploiting the API library provided by the OJS (OAJUtil) it implements specific methods for sending messages to the phones. deliverPageIM() is used to send interactive messages for notifying a page, deliverInfoMsg() to send normal messages containing feedback about unreachable recipients, deliverIMcallBackMsg() to send interactive messages containing pending calls with multiple option responses and deliverChangeProfMsg() to send XML messages for changing the profile mode on a phone.

MyJatXMLHandler class:
The only method belonging to this class is onXML(), called every time the new location of a phone is received by the OJS. The location is included inside an XML string passed to the argument of this method. Each time onXML() starts its execution, it opens a new connection with the *EventListener* in order to send the piece of XML just received.

MyJatIMRespHandler class:
Is a listener which receives the responses sent when a user select an option from an Interactive Message. The customized text strings associated with each option can be set during the creation of the message, by using suitable methods from the API library provided by the OJS server. All the responses are passed to the argument of the method onIMResponse() which implements a client socket used to subsequently route response data to the *EventListener*.

MyJatUserDataHandler class:
This is another listener which receives user data sent by a phone when a user press one of the three soft key present on the device. When received by the OJS, the text string associated with the soft key is passed to the argument of the method onUserData(). Even in this case, when this method is called, it

opens a new connection with the *EventListener* in order to route the received information.

5.4 Implementation

This section describes the implementation of the use cases specified in the requirements specification. It starts by illustrating which are the data structures used by the context-aware application and how they are organized.

5.4.1 Data Structures

All the information are stored inside two XML files: *context_data.xml* and *policy_data.xml*. The first contains information about location, status availability and pending calls for each user enrolled in the system. Its structure is shown in the snippet below:

```
1: <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2:  <context_data>
3:   <user ID="11">
4:    <Status_data>visiting</Status_data>
5:    <Location_Data>0101</Location_Data>
6:    <toCallList>10-15-14</toCallList>
7:   </user>

8:   <user ID="15">
9:    <Status_data>available</Status_data>
10:   <Location_Data>0202</Location_Data>
11:   <toCallList>14-10-11</toCallList>
12:  </user>
13: </context_data>
```

The root node, labelled < *context_data* > contains a number of *user nodes* uniquely indentified by an ID attribute. For each *user node* availability status (line 4), the ID-code of the last location device sensed by the user's phone (line 5) and pending calls (line 6) are listed. This file is used both by the *StoreService* and by the *RetrieveService* to store/retrieve data when requested by the *ContextManager*.
The second file, illustrated in the following snippet, contains policy data. Its structure is slightly different from the previous one:

```
1: <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2:  <policy_data>
3:   <user ID="11">
4:    <recContextMsg>1</recContextMsg><recPendCalls>0</recPendCalls>
5:    <trackHistory>1</trackHistory>
6:   </user>

7:   <user ID="15">
8:    <recContextMsg>0</recContextMsg><recPendCalls>1</recPendCalls>
9:    <trackHistory>1</trackHistory>
```

```
10:    </user>
11:  </policy_data>
```

The root node is named $< policy\_data >$. Each *user node* belonging to it contains three childs describing the policy values that can be configured within the system. The $< recContextMsg >$ node stands for 'receive context messages', its value is used to enable or disable the reception of feedback messages containing location and status availability of an unreachable recipient. $< recPendCalls >$ stands for 'receive pending call messages', its value is used to allow or deny the reception of pending call messages on a phone when a user switches from 'unavailable' to 'available' status or when a user moves from a critical area to a non critical area. The last node, labelled $< trackHistory >$, is used to enable or disable the tracking of historical data.
Each policy can take 2 value: 0 or 1. 1 means that the corresponding policy is enabled, 0 is disabled. If we consider the snipped provided above, the user with ID number 11 has the tracking of historical data enabled, is allowed to receive context messages and is not allowed to receive pending calls messages.

Before describing the single implementation of each use case, it is worth to look at how information about location and availability status of a user are collected inside the context-aware application and how they affect the layout on the phone's display.

5.4.2 Change of location

Fig. 14 illustrates the scenario where a phone changes its position: there are two areas containing location sensors placed on the left/right bottom corner. When the phone inside the Area1 moves to Area2 it detects the ID-code of the new nearest sensor and sends the value 0202 to the base station. Then, according to the data flow previously described, the station propagates this information to the OJS which in turn passes the ID-code to a suitable method of the Java *client* application.

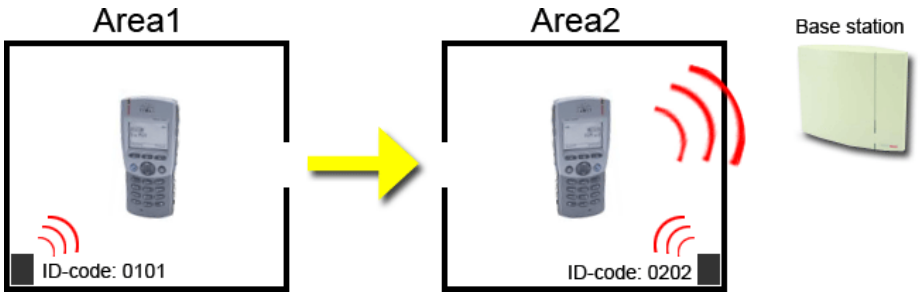

Figure 14: Change Location scenario

Every time a new ID-code reaches the *client*, the information is sent to the context-aware application and stored inside the *context_data.xml* file. The classes

needed to perform this task are five, illustrated inside the interaction diagram in Fig. 15: as soon as a new location data reaches the *client*, the first method being called is onXML(). The argument *locationData* passed to this method is an XML string containing the updated ID-code sensed by a device. This piece of XML is subsequently sent through a socket conneciton to the *EventListener* which parses it in order to extract the updated location of the device. Then, *EventListener* calls newLocation() from the *ContextManager* which in turn updates the information inside the application's database through updateInfo(), updates the user interface by calling changeLocation() and then updates the phone's display by calling sendChgProfileMsg() (only if the new location is a critical area).

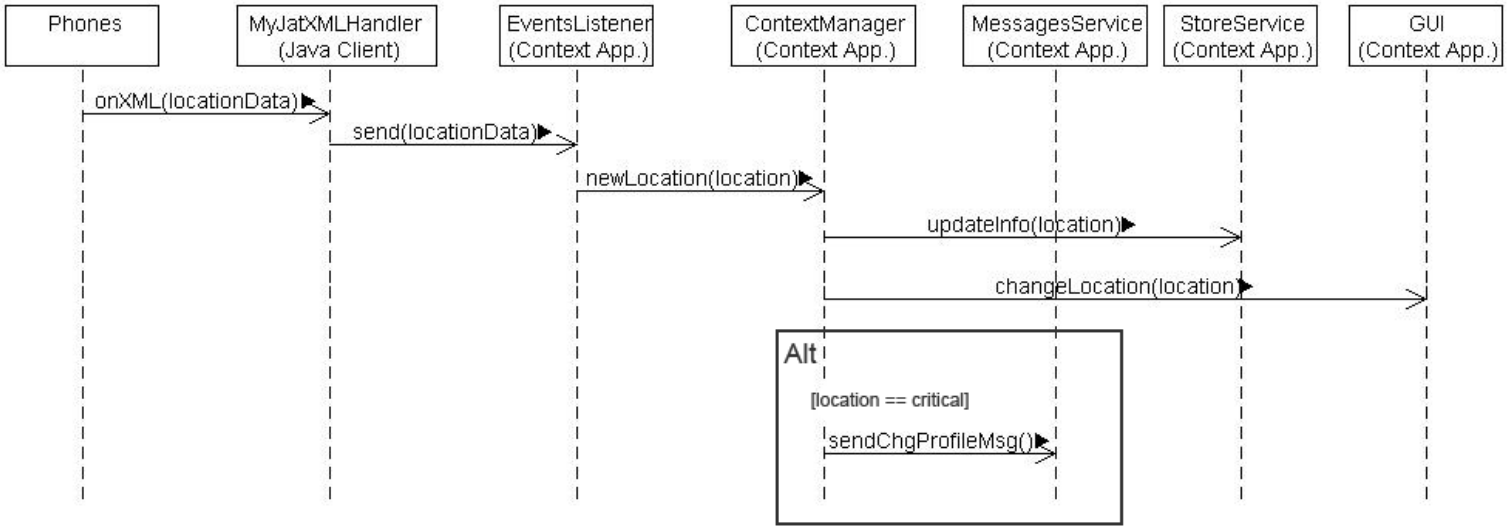

Figure 15: Change Location interaction diagram

Since the application is able to change the display configuration according to the location of the devices, the users are always aware of being inside a critical area (e.g Operating Room), and therefore they are also aware about their un-

Figure 16: 'Standard' and 'Critical area' display configuration.

reachability status. Fig. 16 illustrates the display layout when a phone is inside a safe area on the left and the display layout when a phone is inside a critical area on the right.

5.4.3 Change of availability status

In the standard display configuration, the leftmost soft key button labelled 'Visit' can be used to change the availability status of the user carrying the device. By pressing it, the string 'visiting' is sent to the base station and then routed to the *client* on the OJS server. The case scenario is illustrated in the picture below:

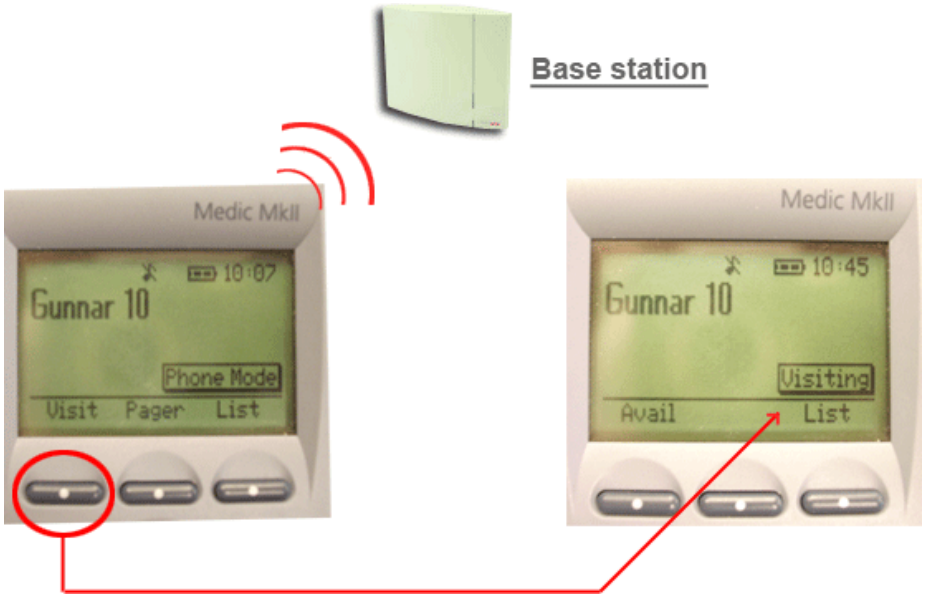

Figure 17: Change Status scenario

The classes needed to save this string inside the context-aware application are similar to those previously described for storing the location information, except for *MyJatUserDataHandler* that is the one used by the *client* to manage data coming from the soft keys. The interaction diagram for this case is shown in Fig. 18: when the 'Visit' button is pressed from a phone, the string 'visiting' is passed to the argument of the method onUserData(). onUserData() opens a new socket connection with the *EventsListener* and pushes data to it. When the informaiton is received, *EventListener* calls the method statusChange() from the *ContextManager* which saves the new status information inside the application's database by calling updateInfo(), updates the user interface through changeStatus() and finally updates the display layout by calling sendChgProfileMsg() (this step is not shown in the interaction diagram).
Therefore, even in this case, when the availability status of a user changes the phone is able to change its display configuration providing a visual feedback about the new reachability condition. The display layout during 'visiting' status

is shown on the right side of the Fig. 17. As it can be seen, in this configuration the leftmost soft key (labelled 'Avail') can be used to switch back to 'available' status and the rightmost to list pending calls.
When the soft key labelled 'Avail' is pressed, the string *available* associated to this button is stored inside the application's database in the same way as described above.

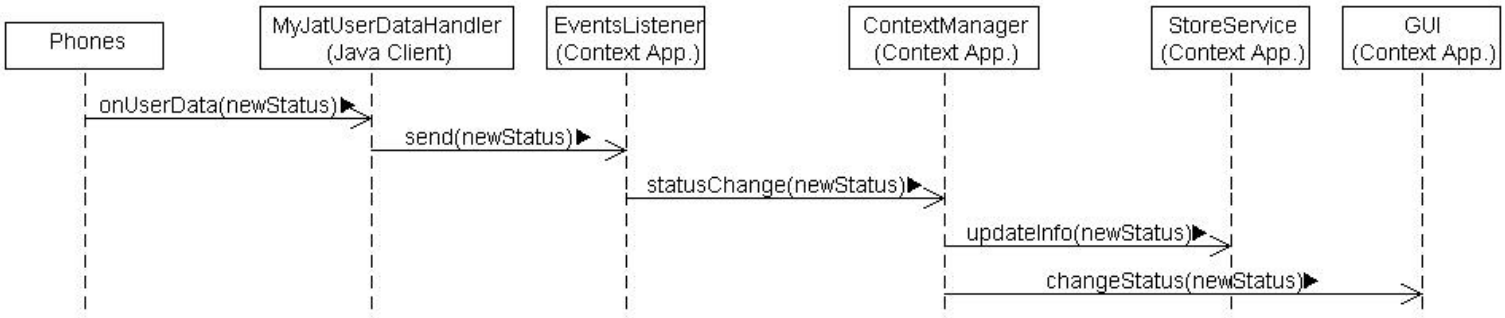

Figure 18: Change status interaction diagram

5.4.4 Pager mode

The pager mode, which allows a phone to behave as a pager, can be activated by pressing the central soft key from the standard display configuration. Fig. 19

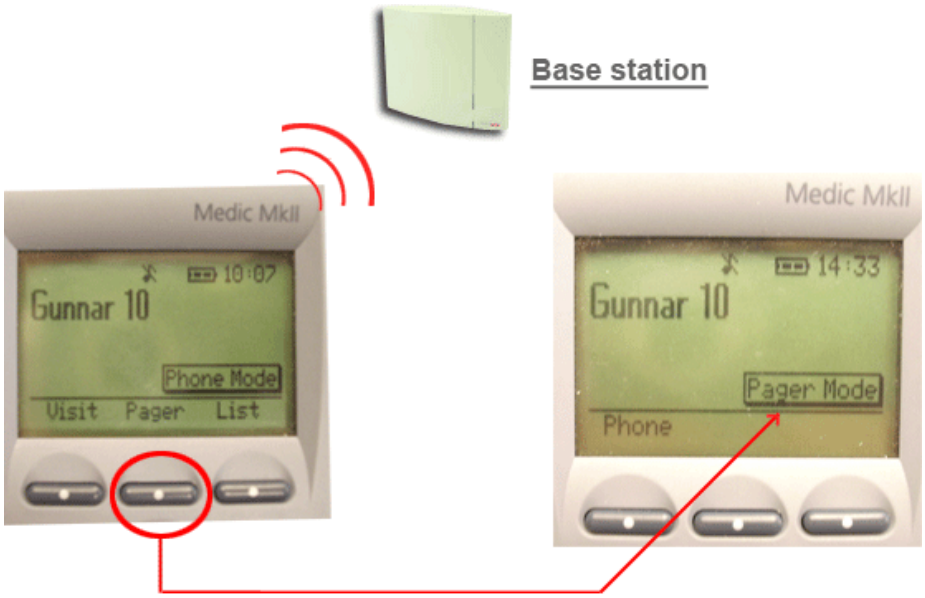

Figure 19: Change to Pager mode

illustrates the standard display configuration on the left and the *Pager Mode* layout on the right. When the display is configured in the 'Pager Mode', the leftmost button can be used to switch back to the normal mode. Since the classes needed to store the string 'pagerMode' sent when a user presses the central soft key are the same as those previously described for storing the 'visiting' status, the interaction diagram for this case will not be illustrated.

5.4.5 Use case 1

The aim of this use case is to manage a call interruption considering the recipient's location. The scenario is illustrated in Fig. 20: when the user 'Gunnar' (with ID 10) tries to call 'Terje' (with ID 11) located inside a critical area, the call is blocked (Step 1), the pending call is saved inside the recipient's profile (Step 2) and a message containing the location of the recipient is sent to the caller's phone (Step 3).

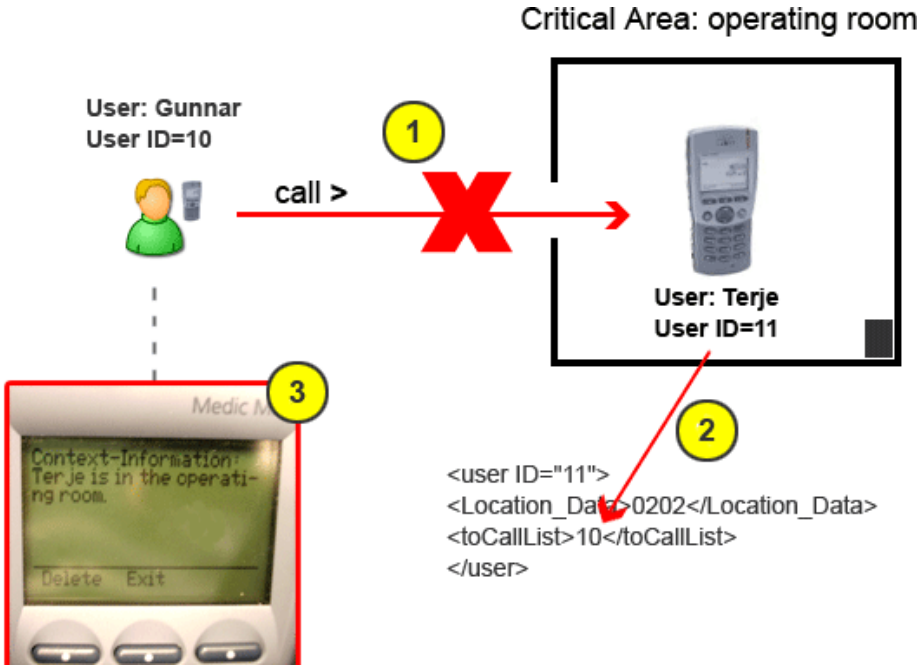

Figure 20: Use Case 1 scenario

Fig. 21 shows the interaction diagram for this use case. When a new call starts from the caller's phone, its control is left by trixbox to the *AgiScript*. By using the method isLocationOk(), *AgiScript* asks to the *ContextManager* if the current recipient's location is inside a critical area or not. In order to give a response, isLocationOk() first extracts the current recipient's location through the method getInfo() from the *RetrieveService* and then checks if the returned ID-code corresponds to a critical area by looking inside a table containing $< ID-code, is\_critical/not\_critical >$ pairs. If *locationResponse* is false, the call is stopped and a vocal message is sent to the caller for asking if he/she

wants to force the call. After the message, if the caller press the digit number 1 the script uses the call() method to force the call, otherwise (the [*else*] block in

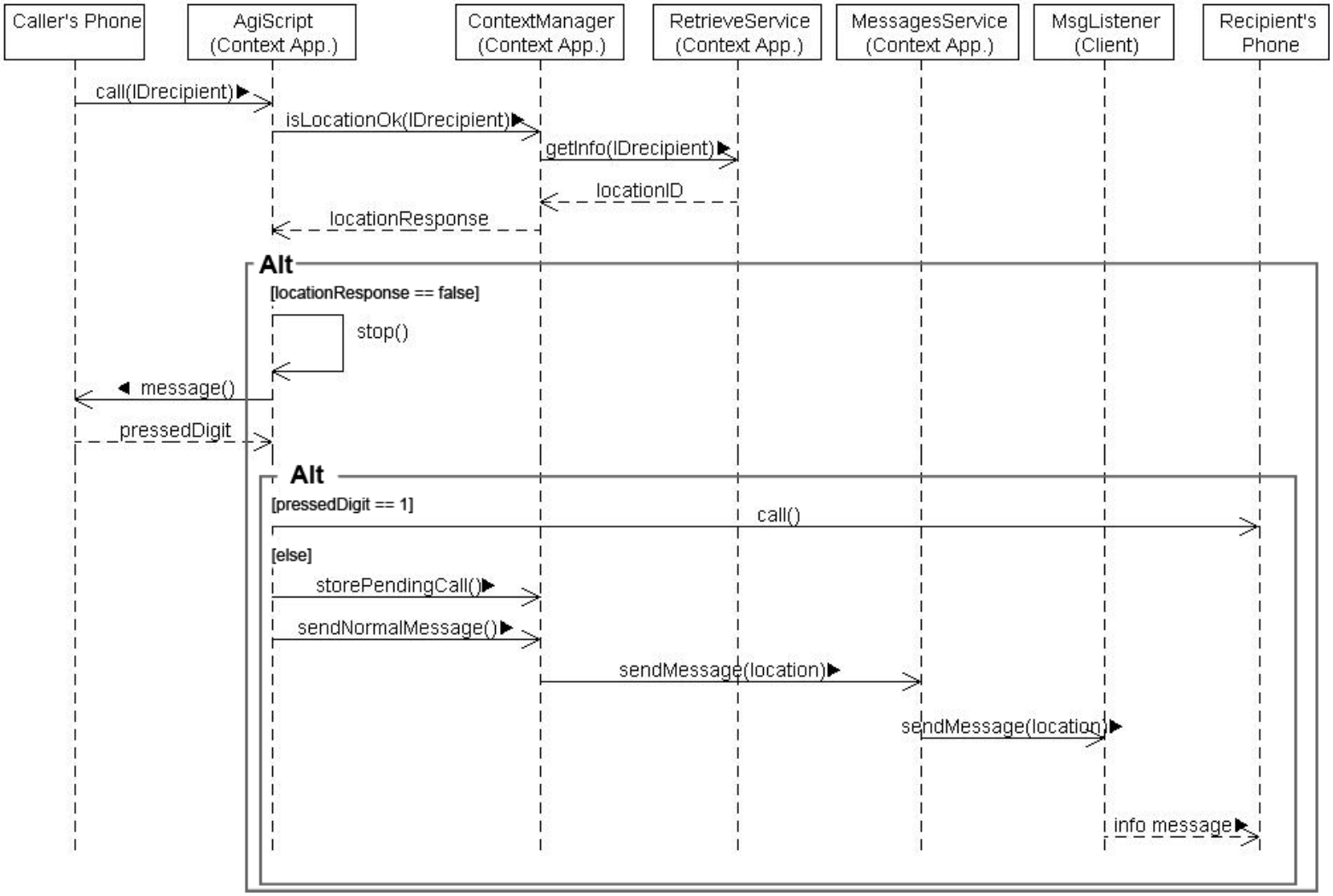

Figure 21: Interaction diagram use case 1

the interaction diagram) executes storePendingCall() to store the pending call inside the recipient's profile and sendNormalMessage() to send a feedback message containing the current recipient's position to the caller. In order to deliver this message, *ContextManager* uses sendMessage() from the *MessagesService*. This method opens a socket connection with the *MsgListener* on the *client* and pushes to it a string similar to the following:

<MSG>
<TYPE> NO <TYPE>
<HDR> Context-information <HDR>
<BODY>Terje is in the operating room.<BODY>

<DEST >10<DEST>
<MSG>

By parsing it, *MsgListener* understand that a 'Normal Message' has to be sent to the caller's phone (because inside the $< TYPE >$ tag, NO stands for Normal Message), knows the header of the message (specified by the $< HDR >$ tag), the body containing the current location ($< BODY >$ tag) and the destination number ($< DEST >$ tag). According to all these information, it calls the method deliverInfoMessage() from the *MessageDelivery* class to finally deliver the 'Normal Message' to the caller's phone containing the current location of the recipient (the last step is not shown in the interaction diagram for space reasons). If the recipient's location is not a critical area, all the previous instructions are not executed and the caller gets immediately in contact with the recipient.

5.4.6 Use case 2

The purpose of this use case is to manage a call interruption considering the availability status of the recipient. The scenario is shown in Fig. 22: on the left the user 'Terje' (with ID 11) tries to call 'Gunnar' (with ID 10), but the call is stopped because the latter is visiting a patient [2] (Step 1). The system stores the number of the caller inside the recipient's profile (Step 2) and then send a feedback message to the caller containing the current availability status of the called person (Step 3).

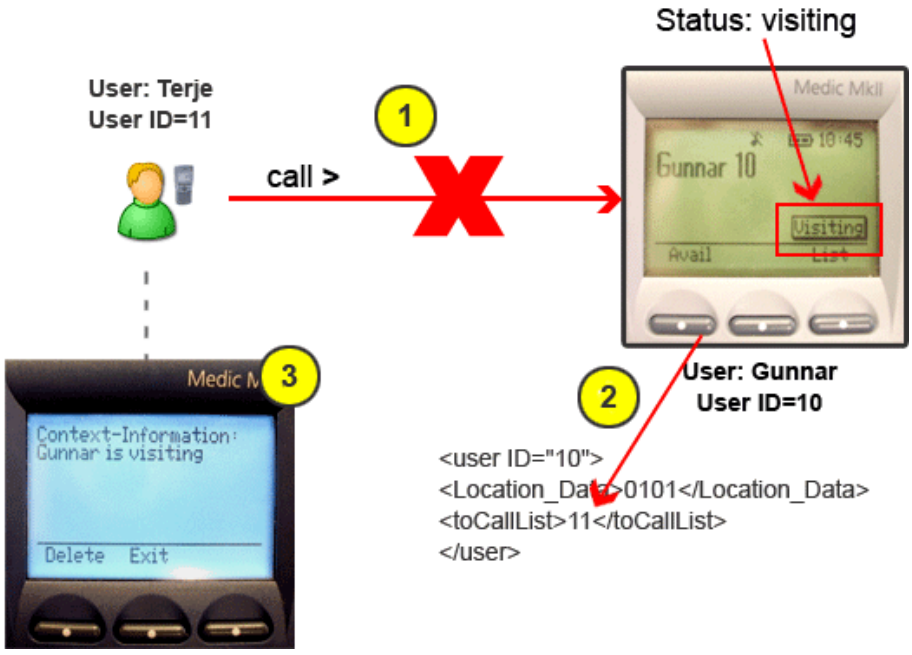

Figure 22: Use case 2 scenario

[2]The 'visiting' status intended here is just an example and can be used in any other circumstance where doctors do not want to be disturbed.

The interaction diagram for this use case is shown in Fig. 23. Basically, the classes involved are the same as those described for the previous case. When a new call starts, *AgiScript* takes its control and asks to the *ContextManager* if the recipient's status is 'available' through the method isStatusOk(). isStatusOk()

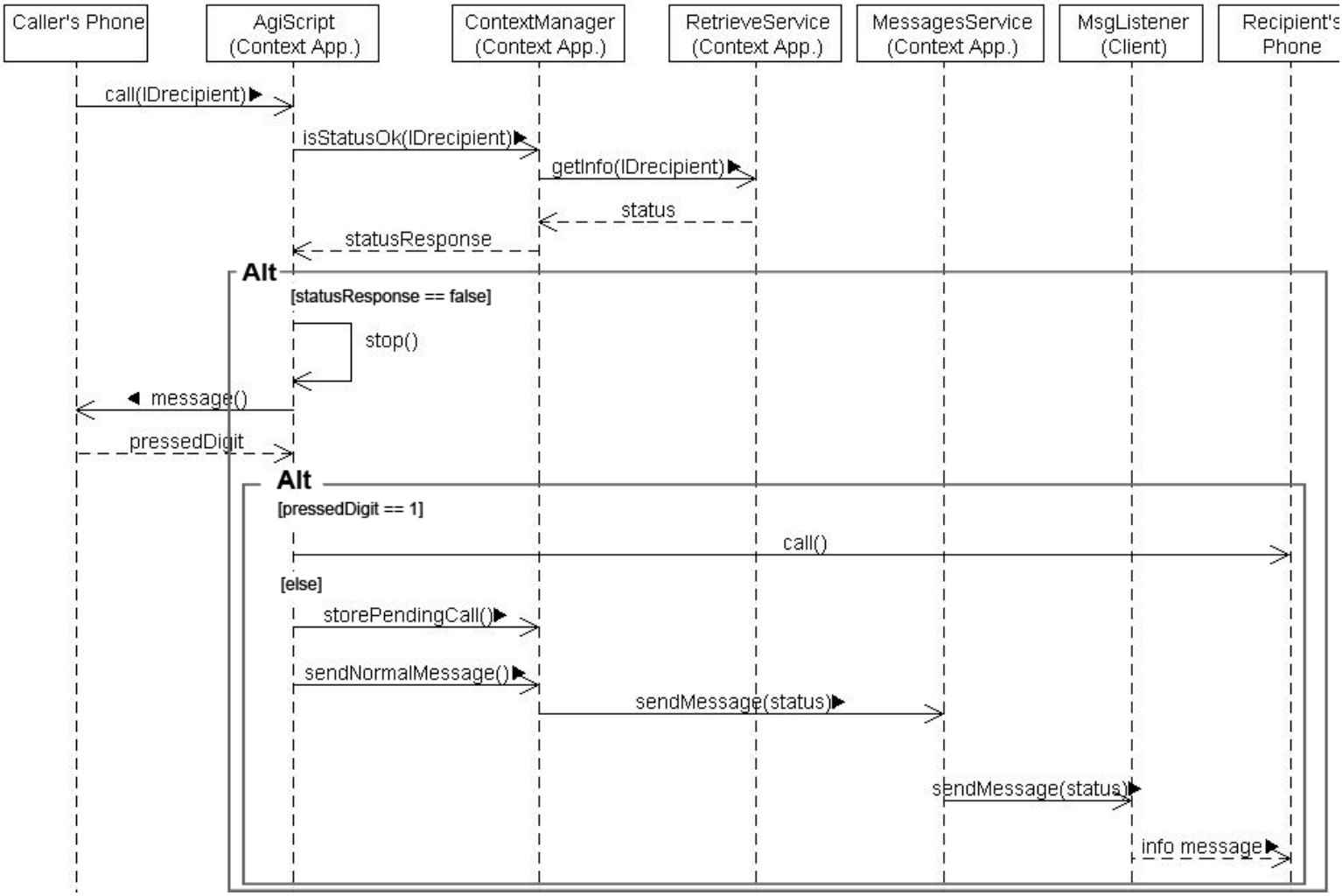

Figure 23: Interaction diagram use case 2

extracts the current recipient's status by calling getInfo() from the *RetrieveService* and then returns true or false according to the information just retrieved. If *statusResponse* is false means that the recipient is 'visiting' and the call is blocked. A vocal message asking if the call has to be forced is sent to the caller through the method message(). If the caller presses the digit 1, the method call() is executed and the caller gets in contact with the recipient's phone, otherwise the pending call is stored inside the recipient's profile and a normal message containing the current availability status is sent to the caller. Even in this case,

in order to deliver the message, *ContextManager* calls the method sendMessage() (from the MessagesService) which in turn opens a new connection with the *MsgListener* on the *client* to send a string similar to the one previously described for the use case 1:

<MSG>
<TYPE> NO <TYPE>
<HDR> Context-information <HDR>
<BODY>Gunnar is visiting.<BODY>
<DEST >11<DEST>
<MSG>

By parsing it, the *MsgListener* extracts the type of message, header, body and the destination number. Then, it calls the method deliverInfoMessage() from the *MessageDelivery* class to finally send a Normal Message containing the current recipient's availability status to the caller (this step is not shown in the interaction diagram for space reasons).

5.4.7 Use case 3

The scenario for this use case is illustrated in Fig. 24. On the left the user 'Terje' tries to call 'Gunnar' who has the phone switched to pager mode.

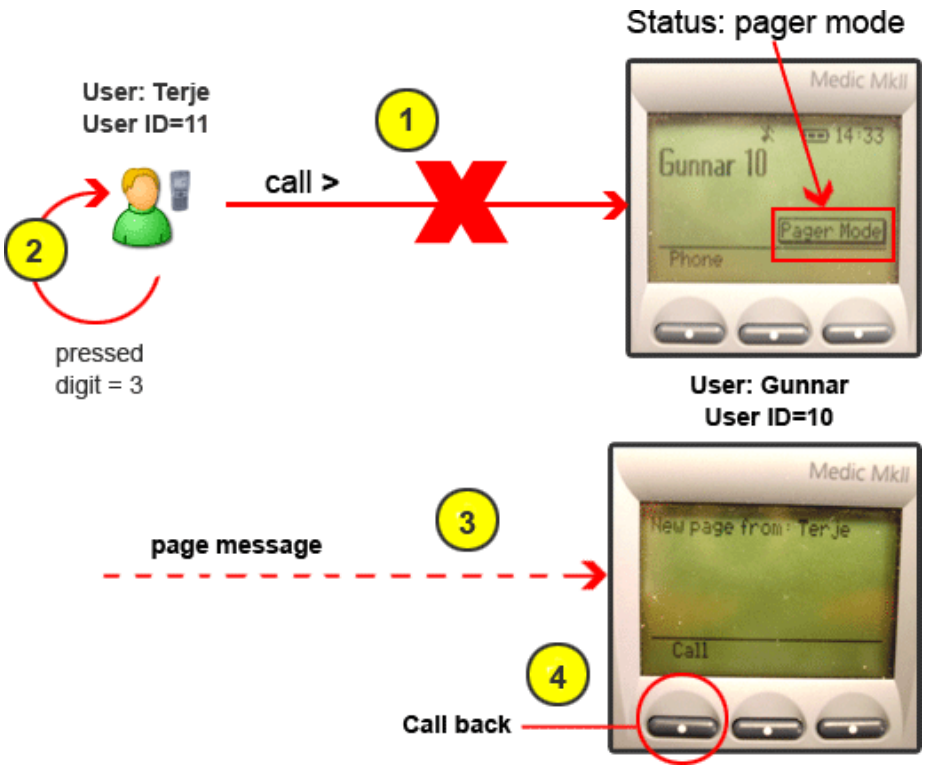

Figure 24: Use case 3 scenario

Since the pager mode does not allow the reception of incoming calls, the call is blocked (Step 1) and the caller decides to press the digit 3 in order to page the recipient (Step 2). A message which notifies a page from 'Terje' is immediately sent to the Gunnar's phone (Step 3).
In this use case, the type of messages used to notify a page are quite different from the previous ones used to send feedback about location and availability status of the recipient. With these messages (called Interactive Messages) it is possible to personalize the functionality of the three soft keys, when they are displayed on the screen. This opportunity has been exploited by the application in order to program the leftmost soft key in a way that it can be used to call back the person who put the page, directly from the page message. In Fig. 24, the Step 4 highlights the soft key labelled 'Call' which can be used to call back the user who put the page in the example: 'Terje'. An interesting advantage gained by using Interactive Messages is that even if the recipient does not want to immediately call back, the received message can be fetched in a second time from the phone's memory, in order to call back who put the page only when it is most convenient.

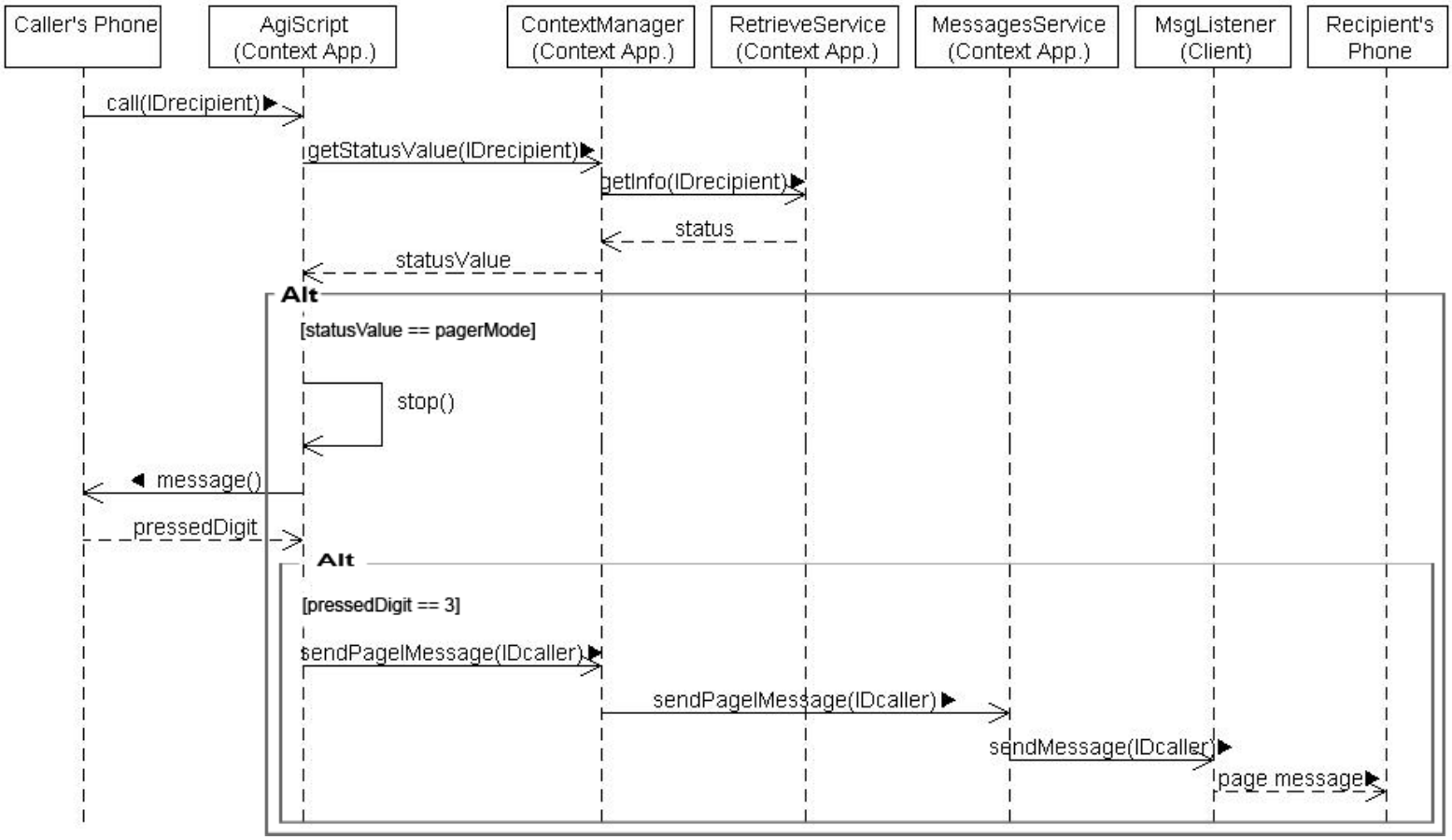

Figure 25: Interaction diagram use case 3

The interaction diagram for this use case is shown in Fig. 25. When a new call starts, *AgiScript* gathers the status value of the recipient's phone by calling getStatusValue() from the *ContextManager*. If the returned *statusValue* equals the string 'pagerMode', then the call is stopped and a vocal message is sent to the caller for asking if he/she wants to send a page message to the recipient. If the user press the digit number 3, *AgiScript* sends a page message by calling the method sendPageIMessage() from the *ContextManager*. This method routes the request to the method sendPageIMessage() from the *MessagesService* which opens a new socket connection with the *MsgListener* on the *client* in order to push the following:

<MSG>
<TYPE> PageIM <TYPE>
<HDR> New page from: Terje <HDR>
<SENDER> 11 </SENDER>
<DEST> 10 <DEST>
<MSG>

By parsing it, *MsgListener* understand that an interactive page message (specified inside the $< TYPE >$ tag through the *PageIM* string) has to be sent to the destination ($< DEST >$ tag) and that the call back number to be configured in the leftmost soft key needed to respond to the page is the one specified inside the tag $< SENDER >$. Since this time the type is *PageIM*, *MsgListener* calls the method deliverPageIM() from the *MessageDelivery* class which finally delivers the interactive message to the recipient's phone (the last step is not reported inside the interaction diagram for space reasons).

5.4.8 Use case 4

The purpose of this use case is to manage an interruption considering user's commitments stored inside Zimbra calendar. The scenario is illustrated in Fig. 26. When the user 'Gunnar' tries to call 'Terje' who is in the middle of a meeting recorded inside the calendar, the call is blocked (Step 1), the pending call is stored inside the application's database (Step 2) and a message explaining that the recipient is involved in a meeting is sent to the caller (Step 3).

Since the interaction diagram for this use case is similar to that described for the use case 2, it is not reported. However it is interesting to look at how the application determines if the recipient of a call is involved in a meeting.
As always, when a new call starts its control is taken by *AgiScript* which immediately checks the recipient's status throught the method isStatusOk(). This method, after performing the instructions already explained in the use case 2, calls getAppointments() from the *RetrieveService*. getAppointments() sends a SOAP request to Zimbra in order to extract all the appointments recorded inside the recipient's account and then sends them back to isStatusOk(). An example of SOAP response is reported in the following XML snippet:

```
<appt uid="99132b3" name="Meeting" dur="11700000">
 <or a="terje@mydomain.com" url="terje@mydomain.com" />
```

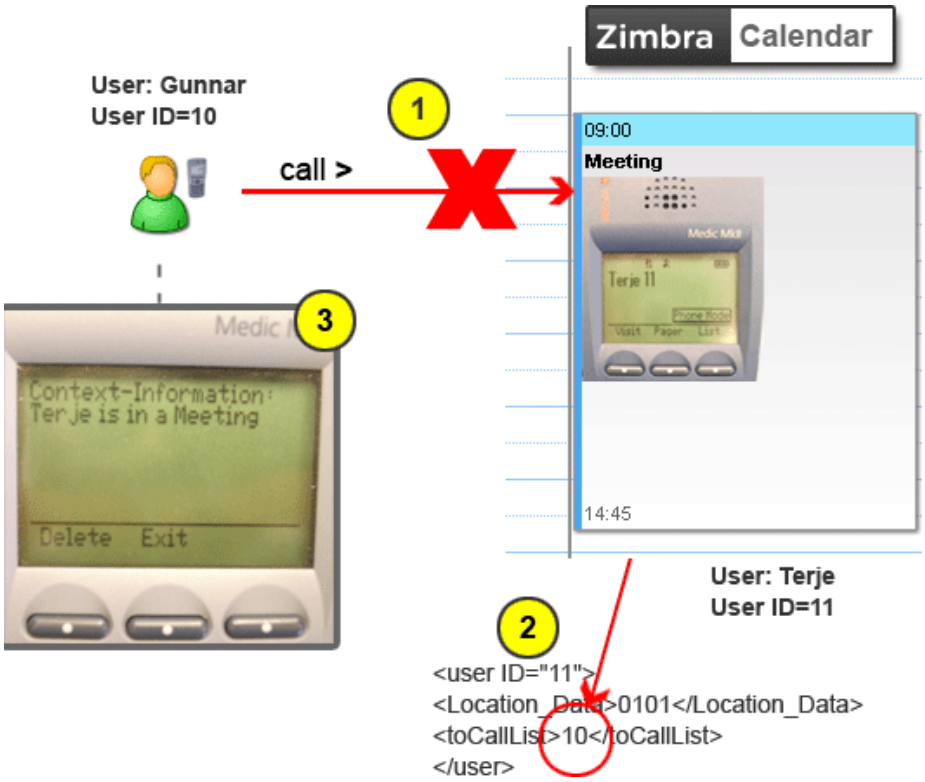

Figure 26: Use case 4 scenario

```
 <inst ridZ="20110301T103000Z" />

<appt uid="99132b3" name="Meeting" dur="11700000">
 <or a="terje@mydomain.com" url="terje@mydomain.com" />
 <inst ridZ="20110301T103000Z" />
```

Each meeting is described by an *appt* node containing a number of information such as *name*, durate in milliseconds (*dur* attribute), user name of the creator, date and time of start (identified by the *ridZ* attribute belonging to the *inst* node).
All the meetings just returned are parsed in order to extract date, start time and durate of each record. For each record the durate is added to the start time and a check against the current system time is carried out in order to see if exists an appointment currently in progress. According to the outcome of this comparison, isStatusOk() returns true or false to *AgiScript*: if false, the call is blocked and a vocal message is sent to the caller in order to ask if he/she wants to force the call. If the user press the digit number 1 the script executes the call() method to let the caller get in contact with the recipient, otherwise a feedback message explaining that the recipient is in a meeting is sent to the caller by using the procedure already explained for the use case 1 and 2.

Unfortunately, the implementation of this use case does not support a mechanism able to change the display configuration of a phone when a meeting recorded inside the calendar starts. This is because Zimbra does not provide a functionality able to notify external applications when events inside the calendar occur. The result is that *even if during a meeting the display configuration looks like in the standard mode, the application block all the incoming calls directed to the device.*

5.4.9 Use case 5

This feature allows a user who tries to contact another user located inside a critical area, to route the call to the on-call person on duty chosen according to the shift schedule stored inside Zimbra calendar. The case scenario is illustrated in Fig. 27: 'Gunnar' tries to call 'Terje' but the call is blocked because the recipient is located inside a critical area (Step 1). The caller decides to press the digit number 2 in order to route the call to the on-call person (Step 2) and the call is subsequently routed to 'Stefano' who is currently on duty according to the shift schedule stored inside the calendar (Step 3).

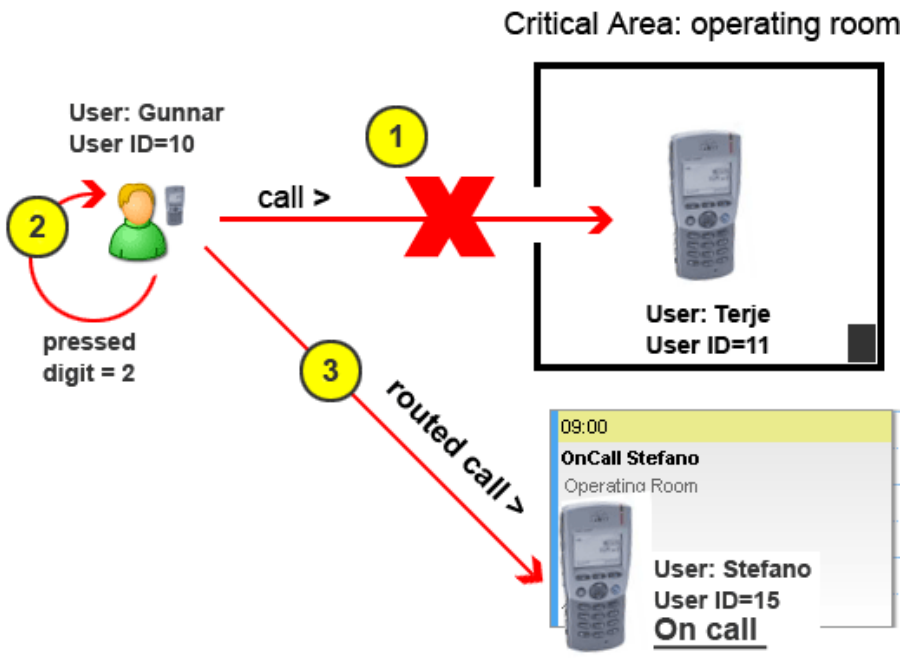

Figure 27: Use case 5 scenario

The interaction diagram for this use case is reported in Fig. 28. The first part of the implementation is the same as that described for the use case 1: when

a new call starts, *AgiScript* takes its control and checks the recipient's location through the method isLocationOk(). If this method returns false the call

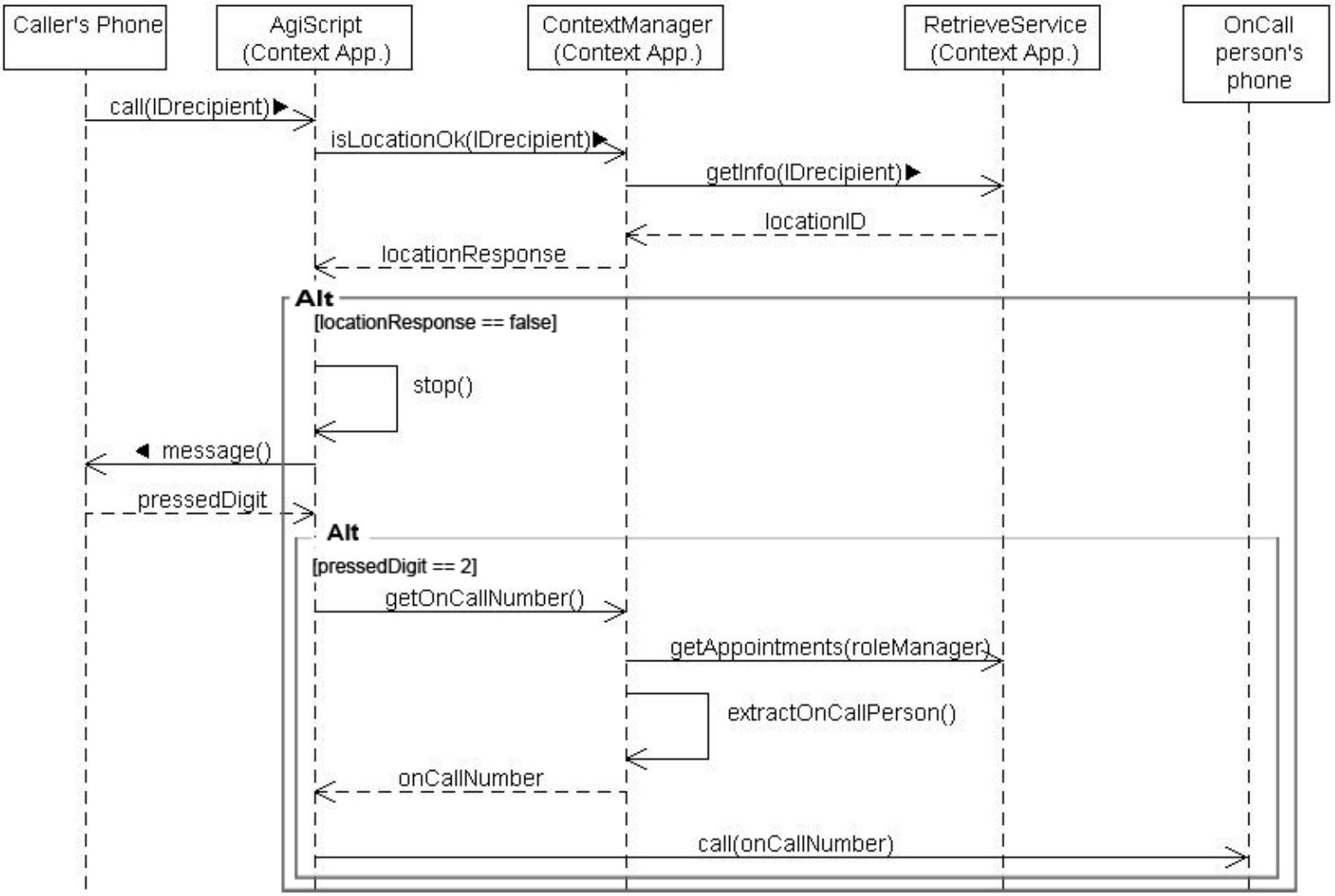

Figure 28: Interaction diagram use case 5

is stopped because the recipient is currently located inside a critical area and a vocal message asking to the caller if he/she wants to route the call to the on-call person is sent by using message(). If the caller press the digit number 2, *AgiScript* executes the method getOnCallNumber() which extracts from Zimbra server the on-call shift schedule corresponding to the critical area, by calling getAppointments(roleManager). The argument *roleManager* passed to this method means that the appointments (representing working shifts in this case) must be retrieved from a special account where the on-call shift schedule must be recorded. After the response, extractOnCallPerson() sums for each returned appointment the start time with the durate and perform a check against

the system time and date in order to find the current on-call person on duty. The corresponding *onCallNumber* of the discovered on-call person is returned to *AgiScript* which in turn executes the call() method with argument *onCallNumber* to finally put in contact the caller with the on-call doctor.
In order to get this feature working each meeting has to be named with the string *OnCall '**Name of the on-call person**'*. In this way, the system is able to parse the title of an appointments (contained inside the tag labelled name) for extracting the name of the person on duty. For example, suppose that the meetings returned after a request made by getAppointments(roleManager) are the following:

```
<appt uid="99132b3" name="OnCall Terje" dur="1200000">
 <or a="roleManager@mydomain.com" url="roleManager@mydomain.com" />
 <inst ridZ="20110301T103000Z" />

<appt uid="92172a3" name="OnCall Stefano" dur="13700000">
 <or a="roleManager@mydomain.com" url="roleManager@mydomain.com" />
 <inst ridZ="20110301T103000Z" />
```

and suppose that the second appointment is taking place during a call. Then the name of the on-call person extracted by extractOnCallPerson() will be '*Stefano*' because the substring after 'OnCall' is '*Stefano*'. After identifying the on-call person, the recipient's number is determined by looking inside a table containing $< userName, phoneNumber >$ pairs.

5.4.10 Use case 6

This functionality allows users to receive messages listing pending calls when they change the status from 'visiting' to 'available' or when they move the phone from a critical area to a normal area. The case scenario for the former is illustrated in Fig. 29. When the soft key labelled 'Avail' is pressed from the 'Visiting' display configuration (Step 1), the phone changes its layout and it switches to the normal 'Phone Mode' configuration (Step 2). After a few seconds, the phone starts receiving messages containing pending calls collected during the unavailability status (Step 3). As it can be seen from the picture, these messages contain three different options:

- Call: by choosing this option the user can call back the person who tried to call during the 'Visiting' status.
- Send availability message: with this option a message explaining that the recipient of the call previously made is now available, is sent to the caller.
- Erase the message: the pending call message is deleted from the phone's memory.

The interaction diagram for this use case is illustrated in Fig. 30: when a user changes its status from a phone, the new status is received by the method onUserData(). This information is sent through a socket connection to the *EventsListener* which calls statusChange(newStatus) from the *ContextManager*.

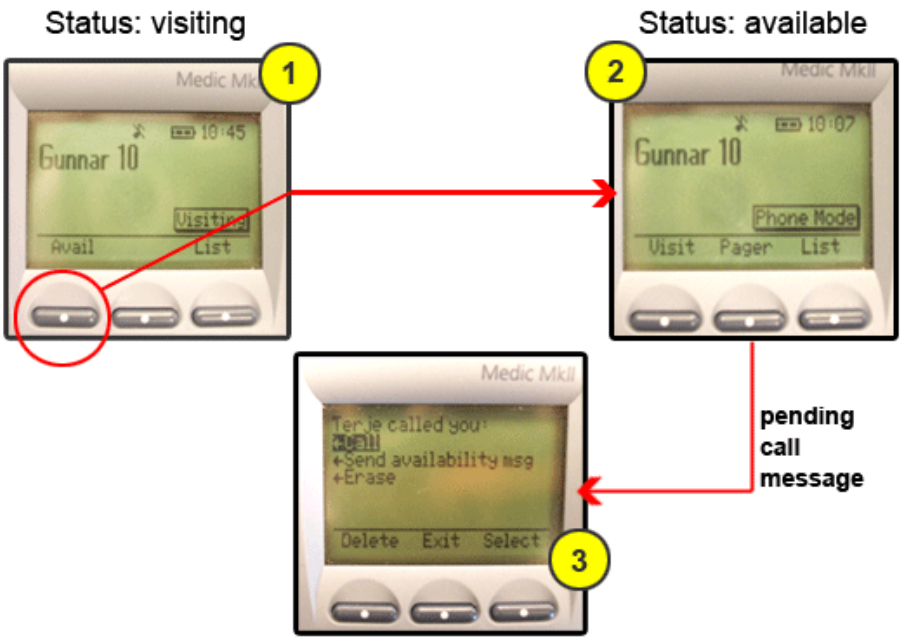

Figure 29: Use case 6 scenario

If the argument *newStatus* equals the string 'available', getPendingCalls() is called in order to extract from the application's database the calls collected during the '*visiting*' status. For each pending call, an Interactive Message is sent to the user's device through the method sendCallBackIMessage(). This method send to the *MsgListener* a string similar to the following:

<MSG>
<TYPE> IM <TYPE>
<HDR> Terje called you: <HDR>
<SENDER> 11 </SENDER>
<DEST> 10 <DEST>
<MSG>

By interpreting it, *MsgListener* understand that an Interactive Message with subject specified inside the $< HDR >$ tag has to be sent to the recipient's number ($< DEST >$ tag) and that the number to be associated with the '*Call*' option selectable from the message is the one specified inside the field $< SENDER >$. According to these information, *MsgListener* executes deliverIMcallBackMsg() from the *MessageDelivery* class in order to finally send the interactive message to the phone (the last steps are not shown in the interaction diagram).

It should be pointed out that, as it can be seen from the picture in Fig. 29 (Step 1), the display configuration during the 'Visiting' status allows always users to list the pending calls collected up to now, by pressing the rightmost button la-

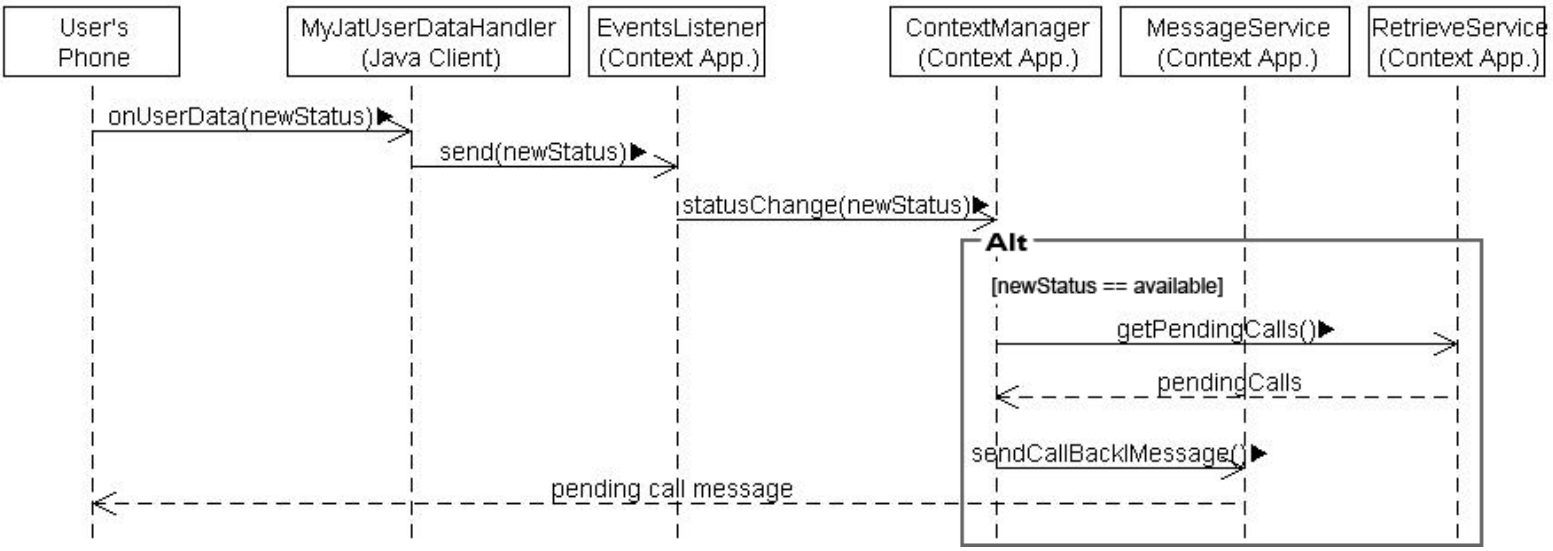

Figure 30: Interaction diagram use case 6

belled 'List'. Thanks to this possibility it is also possible to check pending calls even while keeping the 'Visiting' status.

The reception of pending calls when a user moves from a critical area to a non critical area has been implemented in the same way as explained above. The only difference is that getPendingCalls() and sendCallBackIMessage() are called from the method newLocation() every time a phone sends a new location not belonging to a critical area.

It remains to discuss how users can check pending calls when they are in the middle of a meeting. As previously explained, it was not possible to find a clean solution able to change the display configuration when a meeting starts, because Zimbra does not provide a mechanism able to notify the beginning of calendar events to external application. Therefore has been decided to dedicate the rightmost soft key of the standard display configuration for this task. This button, labelled 'List', can be used anytime to list the pending calls collected up to now during a meeting (Fig. 29, Step 2).

5.4.11 User Interface

The purpose of the developed user interface is to provide a tool from where it is possible to look at the reachability status of all the users enrolled in the system. A screenshot of the GUI is shown in Fig. 31. Each user has a dedicated panel containing information about status availability and location. As it can be seen from the picture, the first panel on the top shows information about 'Terje' who is currently inside the operating room. Since this is a critical area, the orange icon on the left side of the text label highlight that his phone is

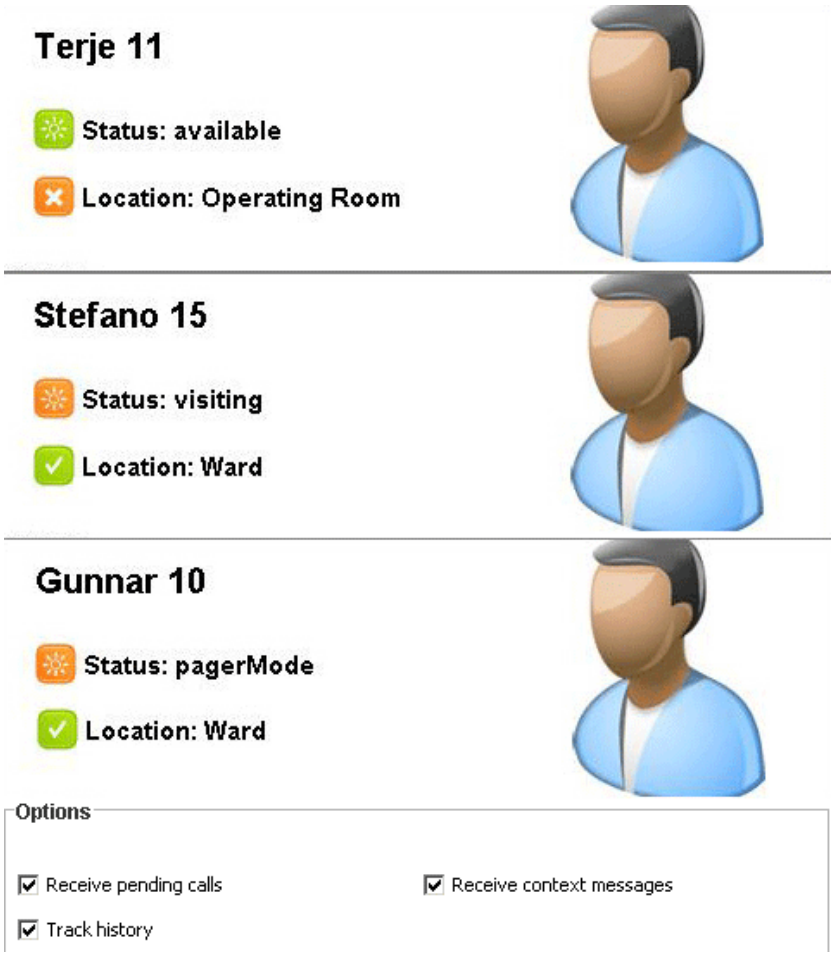

Figure 31: User Interface

currently not reachable. The second panel shows information about 'Stefano' who is performing a visit and therefore he is unreachable too. The third shows information about 'Gunnar' who has the device switched to pager mode. The last panel in the bottom of the picture contains three checkboxes from where it is possible to enable/disable the reception of pending calls, enable/disable the reception of feedback messages and enable/disable the tracking of historical data (for space reasons only the checkboxes for the user 'Gunnar' are illustrated).
Every time a phone's location changes or a user switches the status from the phone, the application immediately refresh the user interface providing always an updated overview of all the users registered in the system.
Unfortunately, for the reason mentioned in the previous section, it is not possible to see if a user is currently involved in a meeting directly from the user interface. Therefore, the GUI does not reflect all the possible unavailability status where users can be involved.

5.4.12 Historical data

The application provides the possibility to track historical data of all the users enrolled in the system. The information are stored inside a specific file named '*history_data.xml*'. Its structure is shown in the snippet below:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<history_data>

  <user ID="11">
    <Location_data>
      <loc>
        <date>2011-04-27 16:05:08</date>
        <ID_dev>0202</ID_dev>
      </loc>
      <loc>
        <date>2011-04-27 16:07:12</date>
        <ID_dev>0101</ID_dev>
      </loc>
    </Location_data>

    <Status_data>
      <status>
        <date>2011-04-27 16:08:30</date>
        <status_info>visiting</status_info>
      </status>
      <status>
        <date>2011-04-27 16:08:52</date>
        <status_info>available</status_info>
      </status>
      <status>
        <date>2011-04-27 16:09:56</date>
        <status_info>pagerMode</status_info>
      </status>
    </Status_data>
  </user>

</history_data>
```

It is composed by a root node called $< history\_data >$ containing a number of childs equals to the number of users. In the snippet provided above are listed the historical information for the user with ID 11. This user node contains two childs: the first, labelled $< Location\_data >$, lists a number of ID-code collected by the system every time the user's device moved from one area to another. The second, labelled $< Status\_data >$, contains the history of availability statuses collected when the user switched from one status to another. $< Location\_data >$ contains a number of $< loc >$ childs listing time, date ($< date >$ tag) and the ID-code of each location device sensed by the user's phone ($< ID\_dev >$ tag). The structure of $< Status\_data >$ node is similar to the previous one: each new status sent by a phone is stored inside a $< status >$ node containing the

date, reception time (< *date* > node) and the text string describing the status, received by the application (< *status_info* > tag).

This file could be used in the future to discover patterns and make statistics in order to answer questions such as 'how long a user spent time inside a particular area' or 'how long a user is switched to visiting mode during a day'.
Since historical data is an information that not everyone is willing to provide, users can always switch off their tracking from the user interface, as described in the previous subsection.

5.4.13 Elegant code

In this section we are going to discuss some pieces of developed code particulary interesting to analyze.

First we will take a look at how communications between the *client* and the *EventListener* are managed. The snippet provided below shows the implementation of the method run(), belonging to the *EventListener*. The purpose of this method is to accept new incoming connections coming from the *client* and leave their management to the *ClientWorker*. These connections take place every time location or availability status are sent by the OJS to the main application.

```
1: public class EventsListener extends Thread{

2:   public void run(){
3:    try{
4:      server = new ServerSocket(1024);
5:      }catch(IOException e) {
6:       System.exit(-1);
7:     }

8:    //wait for new connections
9:    while(true){

10:       ClientWorker client;
11:       try{
12:         client = new ClientWorker(server.accept(),conManager);
13:         Thread t = new Thread(client);
14:         t.start();
15:       }catch (IOException e){
16:         //Handler for the exception
17:       }

18:   }
19:  }
20: }
```

The instruction on line 4 creates a new ServerSocket listening on port 1024. The main cycle (line 9) loop continuously for waiting new connections. When

the *client* connects to the socket, the connection is accepted (line 12) and is passed to the argument of a new *ClientWorker*'s instance. The instruction on line 13 creates a new thread of the instance just created which is then started on line 14. From now on, this thread has the the full control over the connection and completely takes care of the communication management with the client. Once left the connection to the worker, the main cycle is ready to receive a new connection without waiting the end of communication between the *client* and the *ClientWorker*. Thanks to this solution, *EventListener* is capable to accept a greater number of connections in a short period of time, improving the speed of communication required to manage information sent by the *client*.

The second piece of code we are going to look belongs to the *MsgListener* class, part of the client application. This code *interprets* the instructions sent from the *MessagesService* when different kind of messages needs to be sent by the context-aware application to the phones. As illustrated during the discussion about the implementation of the use cases, according to the message to be sent, the methods belonging to the *MessagesService* send strings formatted in this way:

<MSG>
<TYPE> PageIM | NO | IM <TYPE>
<HDR> Subject of the message <HDR>
<BODY> Body of the message <BODY>
<SENDER> sender number <SENDER>
<DEST> destination number <DEST>
<MSG>

Where the tag $< TYPE >$ contains the type of message to be sent (page message, normal message and interactive message), $< HDR >$ the subject, $< BODY >$ the body, $< SENDER >$ the number of the sender and $< DEST >$ the destination number. The snippet below illustrates how the *MsgListener* interprets these strings:

```
1: class MsgListener extends Thread{

2: public void run(){
3:   serverSocket = new ServerSocket(12666);

4:   //accept new connections
5:   clientSocket = serverSocket.accept();
6:   in = new BufferedReader(...);
7:   if (in.readLine().startsWith("<MSG>")){
8:     do{
9:       receive = in.readLine();

10:      //Type of message: Interactive or Normal
11:      if(receive.startsWith("<TYPE>"))
12:        msgType = parseString(receive,6);
```

```
13:         if(receive.startsWith("<HDR>"))
14:           subject = parseString(receive,5);

15:         if(receive.startsWith("<BODY>"))
16:           body = parseString(receive,6);

17:         if(receive.startsWith("<DEST>"))
18:           destAdd = parseString(receive,6);

19:         if(receive.startsWith("<SENDER>"))
20:           IDsender = parseString(receive,8);

21:       }while(!receive.startsWith("</MSG>"));

22:         if(msgType.equals("IM"))
23:           MessageDelivery.deliverIMcallBackMsg(subject, body, ... );

24:         if(msgType.equals("NO"))
25:           MessageDelivery.deliverInfoMsg(subject, body, destAdd);

26:         if(msgType.equals("PageIM"))
27:           MessageDelivery.deliverPageIM(subject, body, destAdd, IDsender);
28:       //....
```

The method run() implements a socket server listening on port 12666 (line 3). Every time the *MessagesService* opens a new connection with the socket, the connection is accepted on the line 5. The instruction on line 7 checks if the first piece of string received is a request of message and if this is the case, means that the *MessagesService* wants to send a new message to the phones. In this case the string is parsed starting from the $< TYPE >$ field (line 11) and then the same is done for the header (line 13), the body (line 15), the destination (line 17) and the sender number (line 19). Every time a new field is parsed its value is stored inside a variable. The parsing is carried out by the method parseString() which extracts the text string between the beginning and the end of a tag. For example, if the string passed to this method is $< DEST > 10 < /DEST >$ the number 10 will be extracted.
After having extracted all the field's values a number of checks against the variable *msgType* are performed. It is first compared with the string 'IM': if it is equal then the method deliverIMcallBackMsg() from the *MessageDelivery* is called, otherwise a check with the string 'NO' is performed. If the two strings are equal then deliverInfoMsg() is called, otherwise the comparison continues. Every time a method from the *MessageDelivery* is called, all the variables required to send a message such as subject, body, destination address and sender ID are passed as well.

6 Tests

After the development, the application has been tested by six medical doctors. The tests were structured by simulating typical scenarios where the functionalities of the system could be involved. Before each test was explained the purpose of the application, how to interact with the devices and the user interface. After each test was asked testers what they thought about the feature just tried without guiding too much their evaluation. With this approach we hope to have obtained sincere consideration and useful suggestions. The scenarios (grouped by use case) proposed during the test sessions are reported below:

Use Case 1 & 5 (manage a call considering location): number of users 3.
Before the following scenarios was explained that one of the two location devices placed in one corner of the test room represented our hypothetical safe area and the second, placed in the opposite corner, represented our hypothetical operating room.

- *Scenario 1*: all the users are outside the critical area. User 1 tries to call the user 2 (the aim of this scenario was to help testers to familiarize with the equipment).
- *Scenario 2*: user 2 moves inside the critical area and user 3 is on-call (with a shift recorded inside Zimbra calendar). User 1 calls the user 2. Iterate the call for each possible alternative that can be chosen after the vocal message: digit 1 to force the call, digit 2 to route the call to the on-call person and hang up.
- *Scenario 3*: user 2 moves back to the safe area (in order to show how the phone receives pending calls messages).

Use Case 2 (manage a call considering availability status): number of users 2.

- *Scenario 1*: user 1 switches to 'visiting' status. User 2 tries to call user 1. Iterate the call for each possible alternative that can be chosen after the vocal message: digit 1 to force the call and hang up.
- *Scenario 2*: user 1 switches back to 'available' status (in order to show how the phone receives pending calls).

Use Case 3 (pager mode): number of users 2.

- *Scenario1*: user 1 switches to pager mode. User 2 calls user 1. Ask user 2 to page the recipient (by pressing the number 3 after the vocal message).

Use Case 4 (manage a call considering meetings): number of users 2.

- *Scenario 1*: user 1 calls user 2 currently involved in a meeting scheduled inside Zimbra's calendar. Iterate the call for each possible alternative that can be chosen after the vocal message (considering also the possibility of hang up).

The tests were not performed by strictly following the order described above because since sometimes testers suggested features already implemented in the application, we chose to anticipate some scenario in order to immediately illustrate them the concerned feature. In the following, the feedback gathered during this phase are discussed. For each new proposed functionality a brief explanation of why it is needed in hospitals is given.

During the tests of the scenarios belonging to the use case 1, a number of new features have been proposed. The first came out when a tester was trying to force a call to a user located inside the operating room. He explained that when doctors are in this room they often wear sterilized clothes and would be particularly useful to allow them to hear the caller through a speaker on the device, automatically turned on after some seconds from the beginning of the forced call. In this way doctors can use the phone without necessarily touching the device and talk without stopping their activity. Currently, when doctors wear sterile suit, other people have to pick up the phone for them. These persons are typically nurses, present in the operating room specifically for doing this task.
Another suggestion strictly related to the previous one, is aimed at avoiding the necessity to touch the phones when a new message is received on the phone. The solution proposed by a tester is a software running on the phone able to read messages as soon as they are received on the device. By hearing the messages' content, doctors who are sterile inside the operating room can evaluate their importance and act according to the information received. For example they could decide to immediately interrupt their current activity in order to call back or delay the task when it is more convenient, if a message is not important.

With the current implementation of the system, added a tester, the forced calls could be easily ignored: the problem is that some of them could be emergencies and therefore must be answered. In order to cope these situations, the tester suggested that could be particularly useful a feature which enables the caller to send emergency messages ringing until when the recipient reacts. When received, these messages should also provide a couple of options which can be used to reply back a short text string such as yes/not, allowing the caller to know if the emergency just sent can be handled or not by the recipient.

It also turned out that the feature which allows to route the calls to the on-call person is not complete. In fact, sometimes there could be three different persons on-call for the same area: the primary doctor (the junior doctor), the intermediate doctor and the senior consultant. The suggestion proposed by a tester is the following: when a person located inside a critical area is called, the system should provide the caller with the possibility to choose one of these three persons from the numeric buttons on the device, for instance:

- digit 2: junior doctor
- digit 3: intermediate doctor

- digit 4: senior consultant

Moreover, after the vocal message explaining that the recipient is not available, could also be useful to hear the name of the on-call doctor associated to each numeric button. For example: 'press digit 2 to contact *Dr. Hansen*, digit 3 to contact *Dr. Gunnar* or digit 4 to contact *Dr. Fredrik*'. The necessity of this feature is motivated by the fact that the person to whom a call could be routed depends on the problem to be solved. In order to highlight this fact, the tester said: '*If the problem concerns a patient in intensive care unit, I am not interested to the first person on-call and I need to move up the competence level*', moreover: '*it is not a good solution if the system says that I can't find another person*' referring to the secondary and the tertiary on-call.

Despite the previous points, the possibility to route the calls to the on-call person has been recognized as a useful solution which avoid doctors to search every time the on-call number to contact on the whiteboard, leaving them more time to focus on their activities and reduce their workload.

One tester pointed out that when the on-call doctors are very busy, it may happen that they ask another colleague to answer the incoming calls for them. Currently, in order to leave the on-call role to someone else, they simply give the pager to the chosen colleague, who become the new responsible to receive incoming calls. Since the phones on which the system relies are supposed to be personal, it is not recommended to directly leave the device to someone else, mainly because the calls not directed to the role could be lost. Therefore, since in these situations leave the on-call role should be done 'on the fly' and when unexpected conditions occur, the tester suggested that there should exists a feature which allows to do this task directly from the devices, and not (as the current solution proposes) by changing every time the on-call schedule inside Zimbra's calendar through a web browser.

Concerning the messages listing pending calls collected when a recipient of a call is not available, a tester said: '*Sometimes I would like the recipient to call me back but others not and therefore if he is not reachable maybe there should be an option leave a message or not*'. The tester suggested that leave a pending call message should be an option and not an imposition as it happens with the current implementation of the system. In fact, the application send a message by default either when the caller hang up or press a digit different from one or two. Therefore, when a call directed to a recipient located inside a critical area (or not available) is made, the system should provide the caller with two alternatives: hang up without living a message and hang up by leaving a message. He also made an important observation about this feature: '*If a person doesn't want to leave a message, the application could raise an unnecessary interruption*'. Here the interruption could be serious because if the recipient receives a message of a person who tried to call for asking help one hour before, the problem may be already fixed at the reception time, making useless the sending

of such message.
Another unpleasant consequence of this solution is that if for instance fifteen persons try to call a phone located inside a critical area (or switched to 'visiting' status), the recipient's phone will get fifteen messages when it quits a critical area (or when it switches back to 'available' status). This is clearly not a good solution because a large amount of messages, as in the previous case, could be irrelevant when received.
One tester added that '*would also be particularly useful to have the possibility to specify the priority of a pending call message, according to the significance of the information to be communicated*'. In this way, he explained, when the recipient receives the message can judge the importance of the missed call and decide if it is worth to immediately call back or postpone the task. The tester added that when a phone become available, it should list pending call messages according to the priority specified inside them. For example, it should present a list with the most important messages on the top and the less important on the bottom.

Pending call messages have been highly appreciated, especially the option which allows to directly call back the sender of a pending call. It avoids the need to search every time the number to be called inside the address book of the phone, simplifying the interaction with the devices.

Another problem discovered during the tests, which has not been considered during the development of the application, is that with the current implementation it is not possible to send an alarm when the recipient of a call is busy on the phone. The tester added that there should exists a way to send an alarm for an emergency, even if the recipient is on the phone with someone else. The tester highlighted that doctors should *always* be contactable for emergencies and this is one of the main reason which still force hospital workers to reply every time they are paged. Therefore, this possibility should be supported by the application, especially if one of its main purposes is to replace pagers.

During a test session, it turned out that when a doctor answers the phone, one of the most common question made by the caller is: '*when are you available?*'. This is an important information which helps hospital's staff to coordinate their daily work. Typically, according to the outcome of this question '*the caller plans the next task to do*', explained a tester. In order to satisfy this need the suggestion is a feature which allows an unreachable recipient to immediately receive a pending call message from where it could be possible to reply back with several options, containing information about '*until when the person will be occupied*'. For example, an interactive message saying '*please indicate when you will be able to call me back*' containing a number of predefined choices listing time periods should be sent to the unreachable recipient as soon as a new call is made. These choices should provide temporal frames covering the next two or three hours, like:

- option 1: I will be available in 10 minutes
- option 2: I will be available in 20 minutes
- option 3: I will be available in 30 minutes
- ...

The drawback of the current solution, said the tester, '*is that the caller knows that you are in a surgery, but he/she don't know when you will be ready. If you are not available in 30 minutes, the caller will find another person or if it is less important he could wait the end of the surgery*'. Therefore, it is clear that some kind of indication about how long a person will be engaged in an activity could be a very useful information.

Speaking about the feature which allows to reduce interruption when a user is involved in a meeting, more than one tester suggested that in this situation would be more useful to *immediately* receive pending call messages when the calls are made, without necessarily pressing every time the button labelled 'list' on the device. Some of them would like to have the phone automatically switched to pager mode, with the ringer switched off. This is because during a meeting the most disturbing thing is the sound of the ringer and not the calls or the messages themselves.

Several testers expressed different needs when page messages are displayed on the device. Currently, the system send page messages containing always the name of the person who put the page, however there are situations where doctors prefer to read the *role* covered by the person instead of the name. For example, if a nurse paged a doctor, the doctor would like to receive a message containing that a *nurse* put the page and not the name of the nurse. On the other hand if the person who paged the doctor is another doctor they prefer to read the name instead of the role.

A number of testers highlighted that the system should give the possibility to adjust its behaviour according to the specific needs of the different department where the application is supposed to be deployed. Moreover, it should be flexible enough in order to support the needs of different categories of workers. For example, one tester expressed serious concerns about the reduction of interruption provided by the system especially for some persons who are reliant on others hospital's workers such as nurses and interns. People who are not fully functional on their own, said the tester, need constantly other colleagues in order to carry out their work and therefore they could be hardly penalized by the strong reduction of interruption. Therefore, the system should also cope this aspect providing a fully customizable behaviour in order to meet the needs of different group of workers.

Several testers really appreciated the concept of the application and the easy

interaction with the phone devices. One of them said that '*this is what we really need*' and that '*all the functionalities are very simple to learn and can be activated by just pressing a button on the phone without the need to navigate every time inside the menu of the device*'. They also appreciated the automatic control of reachability when they are inside critical areas because '*since we are always moving, we do not have to worry to switch on and off the ringer every time we leave or enter these areas*', said a tester. They also expressed approval for the possibility to manually manage their reachability through the visiting and the pager mode.

7 Discussion

Before going further with the discussion it must be reminded that the context-aware solution is based on an Ascom Unite platform which is one of the most common communication system used within hospitals. Moreover, the application uses the PBX trixbox and Zimbra exchange server which are both open source softwares. As a consequence, it does not require proprietary software or specific hardware in order to run and can be deployed with a very low cost.

7.1 Motivations for the chosen architecture

All the choices are aimed at decoupling as much as possible the software components of the architecture, in order to separate the different functionalities provided by each of them.

The first design choice we are going to discuss is about *AgiScript*. As described in the construction chapter, even if during its execution this component requires the *MessageService* to send messages to the phones, in order to send them it must call specific methods provided by the context manager. The advantage of this approach is that it is possible to implement the logic needed to decide if a message can be sent or not directly inside the latter, which acts as a middleware between the policies customizable by the users and the messages requests coming from *AgiScript*. In this way, when *AgiScript* has to send a message to a phone it does not have to retrieve every time the policy value of the recipient in order to determine if it can be sent or not, but simply call specific methods provided by the manager which will take care of the delivery of the message.
The same motivation applies for the communication with the *RetrieveService*. Even in this case, when the *AgiScript* needs to extract user's data, it does not have to interact with this component. An example of disadvantage that can be obtained without using this approach is clear in the following scenario: suppose that the location of a user is directly gathered by *AgiScript* from the *RetrieveService*. In order to know if the retrieved location is a critical area or not, the script should keep inside its code a table listing for each location device the place (i.e the room) where every single sensor is located. Clearly, this is not a good solution because the logic required to control the call's behaviour implemented inside this component has nothing to do with the logic required to estabilish if some area is critical or not.

The necessity to provide the application with a central component which communicates with the other (the manager), is also motivated by the fact that the information about location and status availability coming from the phones must be propagated to the database and the user interface nearly at the same time. This solution helps to keep the code needed to perform these related operations within a single component. The alternative was to implement the code required to update the user interface inside the *StoreService* or inside the *EventListener*, which it is not a clean solution especially if the main purpose of the architecture

is to maintain separated the different functionalities provided by each software component.

This modularity results in an additional advantage in terms of fexibility of the system because in this way it is possible to easily change the implementation of distinct components without affecting the rest of the application. For example, if the technology used for tracking the location of the phones needs to be replaced, the only component of the architecture to be modified is the *EventListener*; or moreover, if the current database based on XML files needs to be replaced with a new more powerful solution based on a relational database, the components to be modified are the store and retrieve services, without touching other parts of the application. Therefore, thanks to these design choices the core functionalities of the system are preserved, even if the underlying technology should change.

7.2 Quality and efficiency considerations

7.2.1 Efficiency

The efficiency of the system is severely limited by its distribution over the network. In particular, one of the major bottleneck is caused by the communication channel between the client running on the OJS and the core application. This is because every time some information has to be exchanged between them, a new socket connection must be set up, which is widely known to be a time consuming process. The application tries to limitate this problem by managing each connection coming from the OJS inside a new thread, as described during the illustration of the elegant code. In order measure the communication efficiency, a program able to simulate the interaction between these two critical components has been developed and subsequently used for testing the performance. The aim of this program, running on the OJS, is to create a predefined amount of socket connections in order to discover how many of them can be efficiently (i.e with a reasonable amount of time) handled by the context-aware application.

The picture in Fig. 32 illustrates the components involved during a typical test: the blue square represents the *Test Program* mounted on the OJS, the red square the context-aware application and the gray square the XML file used to store timing data collected during each test. *Test Program* consists of two classes named *Client* and *Phone*. The first implements a loop which creates every 3 milliseconds a new thread of the class *Phone*. The cycle iterates n times according to the number of connections that have to be simulated. Each *Phone*'s thread contains a piece of code needed to establish a new socket connection with the application, used to subsequently send the timestamp of the OJS (simulating the information sent by the OJS when new status or location data comes from the phones). As it can be seen from the picture in Fig. 32, every time the timestamp of the OJS is sent by the *Test Program* to the application, is stored inside an XML file (named *Tests Data*) together with the application time (App-Time). The structure of this XML is shown in the snipped below:

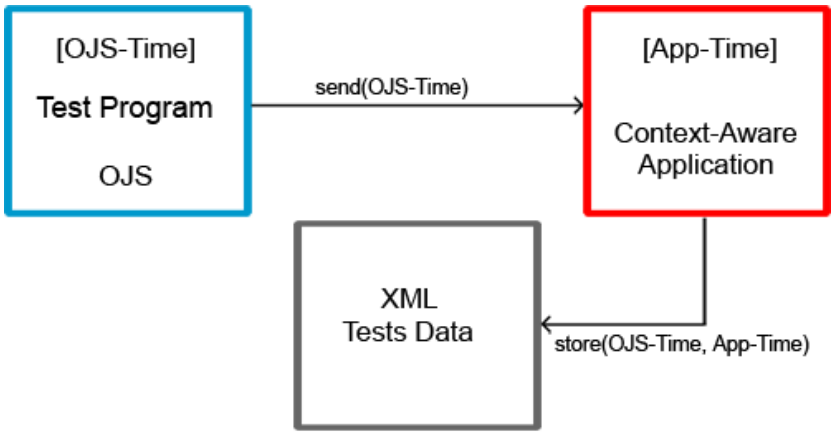

Figure 32: Components involved in the performance test.

```
<info>
  <ojs_time>2011-04-29 16:49:34</ojs_time>
  <server_time>2011-04-29 16:49:38</server_time>
</info>
<info>
  <ojs_time>2011-04-29 16:49:36</ojs_time>
  <server_time>2011-04-29 16:49:38</server_time>
</info>
```

Each $< info >$ node has two childs: the first, labelled $< ojs\_time >$ and the second labelled $< server\_time >$ containing the application time grabbed before storing the data inside the XML file.

In order to analyze the timing pairs collected as described above, a program which scans the XML file has been developed as well. It is very simple: for each $< info >$ node it extracts the ojs_time and the server_time and computes their difference. In order to compute the mean time spent to send the information, the program sums the resulting differences and then divides the result for the number of pairs analyzed. Then, by using the mean just computed, it extracts the variance of the dataset.

It must be highlighted that since location and availability status are strings consisting only by a few characters, the tests were not focused to specifically measure the time required to send a large amount of information, but simply focused to see how the system scale when the number of connections needed to send such little information grows up. This is why we chose to directly send from the *Test Program* the timestamp of the OJS, which approximately contains the same number of characters of the strings describing location and availability status.

The tests have been carried out by gradually increasing the number of threads

created with *Test Program*, starting from 30 up to 3000 and then analyzing the collected timing values. The results of the analysis are reported in the following table:

	Mean	Variance
30 threads	2 sec	1.12
500 threads	1 sec	0.89
1000 threads	1 sec	0.93
2000 threads	1 sec	0.93
3000 threads	1 sec	0.95

As it can be seen, when the number of thread is low the mean time required to send the information is about 2 seconds, with a variance of 1.58. When the number of thread increases, the mean decreases to 1 second and the variance remains constant between 0.89 and 0.95. The motivation of these non-intuitive results comes from the fact that the server machine on which the application runs switches to multithread mode when a large amount of thread is opened, increasing the computational performance. Since the *Test Program* creates a new thread every 3 milliseconds, this means that the application is capable to manage about 300 connection in about 1 second, even when the number of threads grows up.
It must be pointed out that when the tests have been performed by creating new threads in less than 3 milliseconds, some communication error appeared. This is a limitation, and means that in order to guarantee correctness in the transmission, the maximum number of thread manageable by the application is at most 300 for each second.
If we consider that a new socket connection is opened every time a phone changes its location or availability status, 300 is not a very big number especially if we take in consideration the application scenario of the system: hospitals. As already said, hospital's workers are always moving between different areas (changing very often the coverage area between different sensor) or involved in critical activities and therefore this number could be easily reachable. However, it is possible to overcome this problem by sending the data about location on a server and status availability to another, distributing the work load on different machines. Therefore, the scalability of the system can be further improved.

In order to understand the overall picture of the system efficiency, another component that must be analyzed is trixbox. This is a critical component of the architecture because is responsible to manage all the calls between phones. According to the software wiki [92], trixbox is mainly oriented for small offices and 1 server is recommended for about 100 simultaneous calls (this is only a guidance and the exact number of simultaneous calls depends on the characteristics of the machine running the system). If we again consider that the system is targeted to operate inside hospital environment, where a large amount of information is constantly exchanged, even 100 simultaneous calls is not a big number. However, the PRO version of trixbox is capable to handle a greater amount of calls and it is more suitable for the purposes of the application.

Another aspect to be considered concerns the performance of the Ascom Unite system. During the performance evaluation of the platform [77], has been found that there is a bottleneck located between the base station and the phones, due to the limitations of the DECT protocol. In particular, the evaluation highlighed that the time spent to send messages from a phone to another grows up significantly when the total number of messages exchanged over the network raises [77]. The following table (from [77]) reports the time spent to send messages between phones, in relation to the quantity of messages sent at the same time. The messages used during the experiment contained 50 characters.

Num of messages	Time Spent
5	4/5 sec
10	7/8 sec
20	12/13 sec
40	22/23 sec
80	43/44 sec

As it can be seen, when the number of exchanged messages grows, the time spent to send them increases very fast. Since the context-aware application relies on the use of messages in order to send feedback containing availability and location of unreachable recipients and to communicate pending calls, if we sum these messages plus the normal amount of messages that are typically exchanged between users, the performances could be easily drop down.

Summing up, the context aware application can be deployed only in small department, with a reduced number of users. The main motivation is that the application increases significantly the number of messages exchanged inside the Ascom network which, as explained above, is seriously affected by this number. Since feedback messages and pending calls have to be received very quiclky, this aspect could seriously affect the purpose of the application, making pointless the sending of these messages.

7.2.2 Quality

Regarding the simplicity of the application, there are some piece of code particularly difficult to understand especially on the instructions which allow the core application to send messages to the phones. As explained in the implementation chapter, each message request goes through a number of different components and could not be understood at first sight. However, this implementation is strictly necessary in order to guarantee a correct separation of the functionalities and, for example, keep critical parts of the application (such as the policy management) isolated inside the context manager.
The complexity of this part of code is also justified by the necessity to frequently interact with the *client*, which is the key component for enabling messaging between the context-aware application and the phones. Since the *client* is distributed over the network, it was not possible to directly call methods

from it and a simple communication protocol based on text strings containing meaningful tags, has been specifically designed to overcome this difficulty. By interpreting these strings, the *client* running on the OJS is always able to send the right type of message to the phones with the the subject, body and destination number completely specifiable from the core application.

Concerning robustness, the application is able to manage a number of different errors by using specific handlers which allow to capture unexpected conditions. In order to discover potential errors that could occur during the execution of the program, several tests have been carried out. All the discovered problems have been fixed and the stability and robustness of the system has significantly improved. Since the application is only a prototype these tests have not been exhaustively performed, mainly because the system is going to be changed and integrated with new technological solutions in order to add more complex functionalities (e.g the integration with phones based on Android).

7.3 General considerations

As explained during the implementation chapter, Zimbra does not provide an event notification mechanism able to inform the application when a new meeting starts/ends. For this reason, the application is not able to change the display configuration of a device when the owner is involved in such situations. In order to solve this problem, one solution could be to implement a piece of software which every minute contacts Zimbra server to get all the appointments of the users in order to check one by one if a new meeting has begun. This is clearly not a good solution because it consumes time and network resources, especially if we consider that the users enrolled in the system in a real scenario could be hundreds. Another solution could be to develop from scratch a calendar tool tightly integrated with the application. In this way it could be possible to implement a more clean solution able to notify the application only when necessary (i.e when a new meeting is going to start), avoiding the necessity to download every time the appointments from the Zimbra server to constantly look for the beginning of calendar events.

Another important consideration that must be made concerns the limited interaction possibilities offered by the Ascom phones. These devices are not fully programmable and provide only limited possibilities of customization. In addition, they have a small display which can not be used to provide rich information to the users. In [75] authors proposed a redesign of these devices suggesting adjustments aimed at improving the information shown to the users and simplifying their usability.
By using more advanced phones (e.g smart phones) provided with large, customizable and programmable display it could be useful, for example, to develop directly inside them a *panel* listing availability status and location of all the users enrolled in the system, like the one provided by the user interface of the

application. With this feature, all these information would be accessible directly from the devices and not only from the computer hosting the application. Moreover, such panel could be further enriched in order to provide a kind of 'digital whiteboards' from where it would be possible to view several information related to colleagues tipically recorded inside physical whiteboards (e.g on-role shifts). Another interesting feature that can be implemented by using smart phones, is an interface from where users can change the settings of the application in order to personalize the behaviour of the system according to their specific needs.

It must be pointed out that the Ascom phones, along with the Ascom Unite platform, have not been specifically built to support context aware applications. Therefore all the observation previously made are not aimed at highlighting that the Ascom system is a bad product, but for the purpose of the application these limitations affected the final result.

7.4 Considerations about tests

The tests highlighted a number of weaknesses that the context-aware application should fill before being deployed in hospital setting. By using the same hardware and software configuration on which it currently relies, some of the identified problem can be solved, but others not. In the following we are going to discuss some feedback gathered during the test sessions explaining the possible solutions and the major drawback that could be caused by implementing such solutions.

The system can be easily modified to support the routing of a call to three different on-call persons, as proposed by one tester. The implementation of this feature would be straightforward: use the remaining numeric buttons on the phones to let the caller choosing the most appropriate specialist on-call to whom a call must be routed (junior, intermediate doctor or senior consultant). After the user choice, in order to discover the current on-call person, the application will just have to look inside Zimbra the suitable calendar containing the shift schedule for the type of specialist requested by the user. Therefore, this functionality is a simple extension of the use case 5.
It must be highlighted that in this case the shift management could become really complicated: it would require to have for each critical area three distinct calendars each of which containing the shift schedule for a single type of specialist on-call. The alternative to this solution could be to store all the on-call shifts grouped inside one single calendar. However, the drawback of this approach is that since Zimbra has not been specifically thought for this purpose, its calendar does not provide a clear view when it comes to display many working shifts overlapped in time. This is because we adapted the main functionality of this tool (which is store meetings of a single user) for our purposes: store working schedules. If we want to provide a more clean solution able to also support this level of complexity, the schedule management should be redesigned in order to provide users with a more suitable interface which first would be able to pro-

vide an easy way to insert several on-call shifts for the same area with entries overlapped in time, and second able to provide a clear and intuitive view when such on-call shifts are displayed to the users.

Concerning the possibility to leave 'on the fly' the on-call responsibility to someone else directly from the phones, this feature is hardly achievable with the current devices, basically because they do not provide browsing functionality required to access Zimbra. In order to enable the access of the calendar even from the phones, a solution could be to use more technologically advanced devices (e.g palm phones based on Android) provided with a browser.

Speaking about the possibility to choose leaving a pending call message or not to an unreachable recipient, the tester who raised this point was right: the automatic sending of a message when the caller hang up (as currently implemented in the system) could effectively be a source of interruption, mainly because it causes the reception of a large number of messages that could not be relevant anymore when received by the recipient. Therefore, since this feature increases the number of unnecessary interruptions, it must be fixed. The problem can be solved by using the remaining numeric buttons on the devices in order to provide the caller with the following alternatives: hang up without living a message or hang up by living a message.

The use of several numeric buttons on the devices required to implement the suggested features, presents a severe drawback. Consider the scenario where a caller calls a recipient located inside a critical area: if we sum the three numeric buttons required to route the call to the on-call persons, plus the two digits required to send or not a pending call message after a call, plus the digit to force the calls, then the interaction with the phones could become really difficult. Here, the major problem is that when a user calls an unreachable recipient has to listen a very long vocal message explaining the functionality associated to each numeric button. This is clearly not a good solution because users would be forced to hear such vocal message every time an unreachable recipient is called, causing an unpleasant waste of time.

A proposed feature that can be easily implemented without using additional buttons, is the one which allows to send back to the caller a message containing information about until when an unreachable user will be busy. In this case we can again use Interactive Messages in order to program inside them a number of options selectable by the recipient, listing time frames describing the 'busy time'.

Moreover, the feature which allows to view the name or the role of a person who put a page inside page messages, could be easy implementable too. In this case the application will just have to insert inside the body of the page message the name or the role of the person who put the page, by taking in consideration the preferences of the recipient.

There are two suggested feature that cannot be implemented in the system, due to the hardware equipment on which the application relies. The first one is about the possibility to send alerts when a recipient is busy on the phone with another person. Although the Ascom Unite platform provides the possibility to send alerts to the phones from the OJS, this feature could not be implemented due to the substitution of trixbox with the ESS component for managing calls. The second one is related to the possibility to speak with a caller who forced a call through a speaker on the device. Even this feature can not be implemented because the Ascom phones do not provide a speaker integrated inside the devices, mandatory for the implementation of this feature.

8 Conclusion

The presented context-aware application, based on an Ascom Unite communication platform, is aimed at reducing interruptions caused by wireless phones and improving awareness between users carrying these devices. It is specifically thought to support activities of hospital workers and manages interruptions considering contextual information related to users such as location, availability status and personal commitments.

Since clinicians are always moving in different areas, visiting patients and involved in several meetings, with the proposed context-aware solution they do not have to switch off the ringer of their phone every time they are in such critical situations. The system is able to automatically manage the reachabiliy of their devices reducing the number of interruptions and helping them to better focus on their daily activities. Moreover, the system is able to increase awareness sending feedback messages explaining the cause of an unreachable recipient and sending pending call messages collected during the unavailability status only when a critical area or status is left.

The feature which allows to route the calls to the on-call person according to the shift schedule recorded inside Zimbra calendar, provides a useful way to easily get in contact with the on-call doctor on duty, avoiding the need to search every time the number of the person to contact on the whiteboards.

The pager mode, model the behaviour of a phone as a pager. It overcomes one of the major drawback of these devices: the call functionality. With this feature users can be paged on the phone and at the same time call back the person who put the page without the need to search a phone nearby, task wich usually requires an unuseful waste of time.

In order to evaluate the application, a number of tests with some medical doctors have been carried out. The tests highlighted that the application must fill several gaps before being deployed in a real hospital: some of them can be fixed, but others not.

The performance analysis pointed out that the system can be deployed only in small department because it is not highly scalable. The main reason is that it uses a large amount of messages in order to provide awareness to the user, which could cause a loss of performance inside the Ascom platform. However, some strategies have been proposed to improve this aspect.

Future works could be aimed at filling the gaps raised during the tests, integrate the system with different kind of devices providing advanced functionalities and find better strategies to effectively improve the performances in order to deploy the application even in large hospital department.

Appendix A: Ascom Documentation

TD 91026GB Mailgate Function Description
TD 92040GB Open Access Toolkit Programming Guide
TD 92161GB Integrated Message Server Installation & Operation Manual
TD 92185GB Open Java Server Installation & Operation Manual
TD 92187GB GSM/SMS application on OJS Function Description
TD 92198GB Netpage Installation & Operation Manual
TD 92204GB Open Access Server Installation & Operation Manual
TD 92215GB Open Access Protocol Function Description
TD 92230GB Open Java Server Programming Guide
TD 92232GB ELISE2 Installation Guide
TD 92243GB UNITE System Description
TD 92253GB Enhanced System Services Installation & Operation Manual
TD 92258GB UNITE System Planning
TD 92324GB Portable Device Manager Data Sheet
TD 92325GB Portable Device Manager Installation & Operation Manual
TD 92333GB 9d24 User Manual
TD 92341GB Activity Logging in Unite Function Description
TD 92365GB 9dLD Installation Guide
TD 92370GB IP-DECT Base Station (IPBS) Data Sheet
TD 92375GB IP-DECT System Description
TD 92421GB Unite Log Analyzer Installation & Operation Manual
TD 92481GB DC4 Installation & Operation Manual
TD 92579GB IP-DECT Base Station and IP-DECT Gateway Inst. & Op. Man.
TD 92584GB d62 Quick Reference Guide
TD 92585GB IMS2 Data sheet
TD 92586GB IMS2 Installation & Operation Manual
TD 92622GB d62 Protector Data Sheet
TD 92629GB DECT Location Function Description
TD 92639GB d62 Configuration Manual

Appendix B: Configuration of the Ascom system

In the following will be described the steps required to configure the IMS module in order to route data coming from the phones connected to the DECT protocol, to the OJS-GSM module.

1. Open a browser and insert as URL the IP-Address of the IMS (in our case http://193.157.81.62).
2. Open the advanced configuration panel of the module.
3. Under the DECT interface section (Step 1), select 'Message Distribution' (Step 2): the displayed page should be the same as that one illustrated in the figure below:

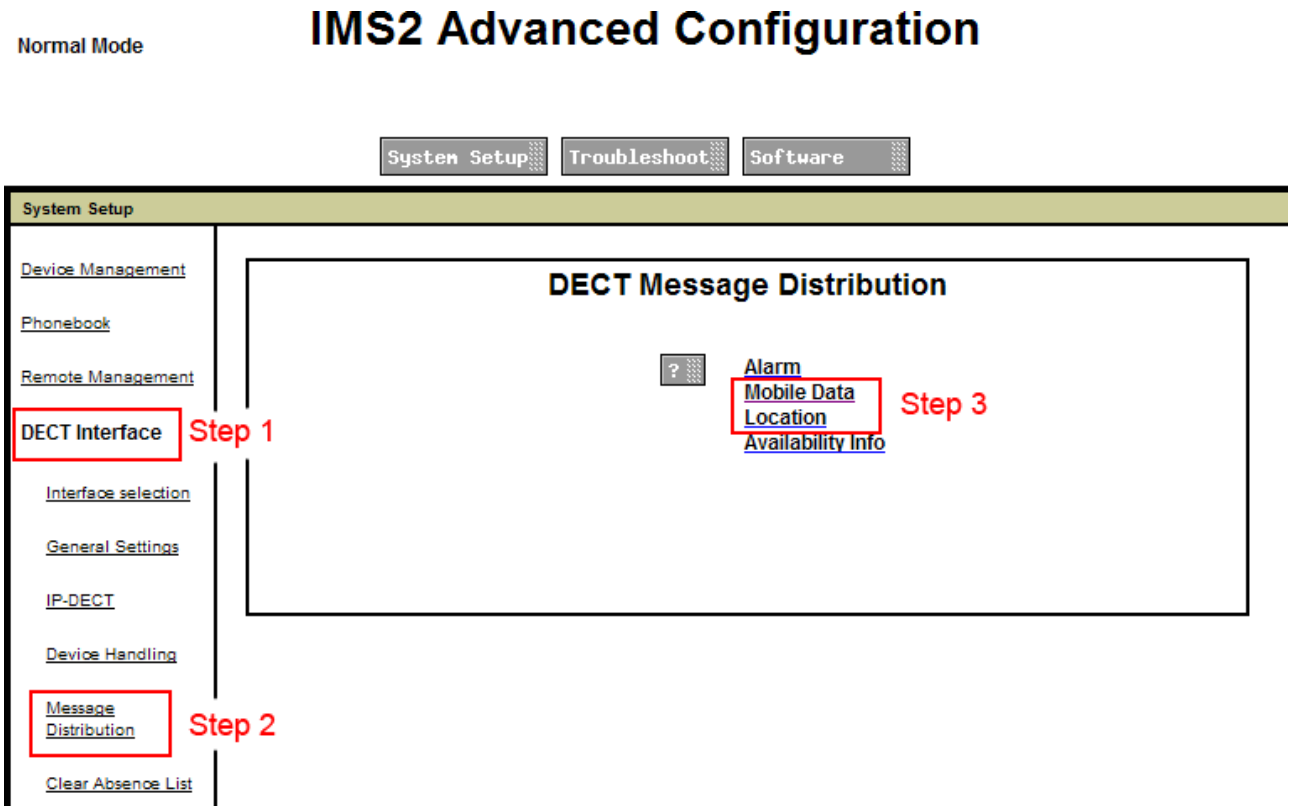

Figure 33: IMS configuration

4. Since we are interested to route both location of the devices and data sent when the soft key buttons are pushed, we have to configure the entries 'Mobile Data' and 'Location' (Step 3). By selecting the 'Mobile Data' link, the page illustrated in Fig. 34 should appear.

5. Insert inside the Destinations list the IP-Address of the OJS-GSM server followed by the name of the service responsible to handle incoming data coming from external modules: in this case the service is named OAJ, by default. Fig. 34 illustrates this step.

6. After this, come back to the Message Distribution list in the previous page and repeat the point 5 for the 'Location' link.

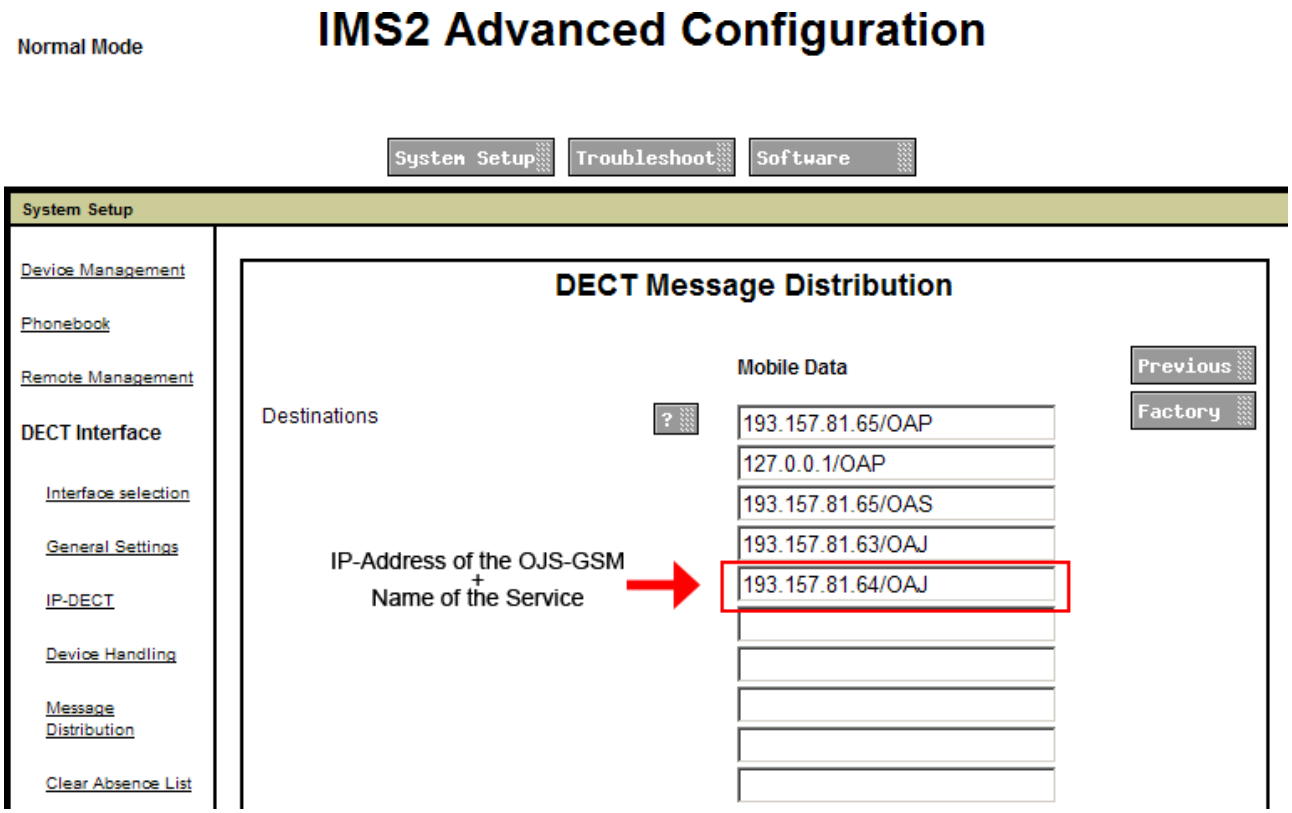

Figure 34: IMS configuration

From now on, all the information about location and user data coming from the phones will be routed by the IMS to the OJS-GSM module.

References

[1] Aziz O., Panesar S. S., Netuveli G, et al. Computers and the 21st century surgical team: A pilot study. BMC Med Inform Decision Making 2005;5:28.

[2] Ammenwerth E., Buchauer A., Bludau B., Haux R. Mobile information and communication tools in the hospital. Int J Med Inf 2000; 57: 21−40.

[3] Bisgaard J.J., Heise M. and Steffensen C., How is Context and Context-awareness Defined and Applied? A Survey of Context-awareness. 2004, Department of Computer Science, Aalborg University.

[4] Bardram JE, Hansen TR, Soegaard M, AwareMedia - A Shared Interactive Display Supporting Social, Temporal and Spatial Awareness in Surgery, Proceedings CSCW 2006, pp 109−118.

[5] Bisgaard J.J, Heise M. and Steffensen C., How is Context and Context-awareness Defined and Applied? A Survey of Context-awareness. 2004, Department of Computer Science, Aalborg University.

[6] Bardram J.E. Applications of ContextAware Computing in Hospital Work Examples and Design Principles. In Proceedings of the 2004 ACM Symposium on Applied Computing, pages 1574−1579. ACM Press, 2004.

[7] Bradner E., Kellog, W. A., And Erickson, T. 1999. The adoption and use of Babble: A field study of chat in the workplace. In Proceedings of the 6th European Conference on Computer-Supported Cooperative Work (ECSCW 99, Copenhagen, Denmark, Sept.12−16), Kluwer Academic, Dordrecht, Netherlands, 139−158.

[8] Botsis T., et al. Context-Aware Systems for Mobile Communication in Healthcare - A User Oriented Approach. in Proceedings of the 7th WSEAS International Conference on Applied Informatics and Communications. 2007. Athens, Greece.

[9] Boehm B.W, A Spiral Model of Software Development and Enhancement, 1985, TRW Technical Report 21−371−85, TRW, Inc., 1 Space Park, Redondo Beach, Calif. 90278.

[10] Bardram J, Kjr TAK, Nielsen C (2003) Supporting local mobility in healthcare by application roaming among heterogeneous devices. In: Proceedings of the 5th conference of mobile human-computer interaction. Springer, Berlin Heidelberg New york, LNCS 2795, pp 161−176.

[11] Bardram JE (2000) Temporal coordination. Comput Support Coop Work 9:157−187.

[12] Brown B., Randell R., Building a context sensitive telephone: some hopes and pitfalls for context sensitive computing, in Proceedings of the Joint Colloquia: Building Bridges Interdisciplinary Context Sensitive Computing and Mobile Computing in Medical Context, Glasgow, September 910,2002.

[13] Bricon-Souf N., Newman C.R., Context awareness in healthcare: a review, Int. J. Med. Inf., in press, accessed online on April 2006.

[14] Coiera E, Tombs V. Communication behaviours in a hospital setting: an observational study. Br Med J 1998;316:673−676.

[15] Cheverst K., Mitchell K. , Davies N., and Smith G. Exploiting Context to Support Social Awareness and Social Navigation. SIGGROUP Bull., 21(3):43−48, 2000.

[16] Cugola G., DiNitto E., and Fugetta A., Exploiting an event-based infrastructure to develop complex distributed systems, Proc. ICSE98, pages 261−270, 1998.

[17] Chen G., Kotz D. A survey of context-aware mobile research. Technical Report TR2000381, Department of Computer Science, Dartmouth College (2000)

[18] Context sensitive systems for mobile communication in hospitals, project description.

[19] Dey AK, Abowd GD. Towards a better understanding of context and context-awareness. CHI2000 Workshop on the What, Who, Where, When, and How of ContextAwareness, 2000.

[20] Dourish P., Seeking a foundation for context aware computing, Hum. Comput. Interact. 16 (2) (2001) 229−241.

[21] Eisenstadt SA, Wagner MM, Hogan WR, Prankaskie MC, Tsui FC, Willbright W. Mobile workers in healthcare and their information needs: are 2-way pagers the answer? Proc AMIA Symp 1998:135−9.

[22] ETSI, Radio Equipment and Systems Digital European Cordless Telecommunications (DECT). 1992.

[23] Favela J., Rodriguez M., Preciado A., and Gonzalez V. M. Integrating context-aware public displays into a mobile hospital information system. Information Technology in Biomedicine, IEEE Transactions on, 8(3):279−286, Sept. 2004.

[24] Favela J., Tentori M., Castro L.A., Gonzalez V.M., Moran E.B., Martinez Garcia A.I.: Activity recognition for context-aware hospital applications: issues and opportunities for the deployment of pervasive networks. Mob. Netw. Appl. 12(2−3), 155−171 (2007).

[25] Fogarty J. , Lai J., Christensen J., Presence versus availability: the design and evaluation of a context-aware communication client, Int. J. Hum. Comput. Stud. 61 (3) (2004) 299−317.

[26] Fasani S. Evaluation of the Ascom/trixbox System for Context Sensitive Communication in Hospitals, Norwegian Centre for Integrated Care and Telemedicine, Internal Report.

[27] Garlan D., Siewiorek D. P., Smailagic A., and P. Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. IEEE Pervasive Computing, 1(2):22−31, Apr. 2002.

[28] Gamma E., Helm R., Johnson R., and Vlissides J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley, Reading (MA), 1994.

[29] Gilb, T. 1988. Principles of Software Engineering Management. Reading, MA.: Addison-Wesley.

[30] Garrison K., trixbox CE 2.6.2009: Packt Publishing. pag. 7-28.

[31] Helal S., Giraldo C., Kaddoura Y., C. LEE, H. El Zabadani, W. Mann, Smart phone based cognitive assistant, in: Ubihealth,12 international journal of medical informatics 76(2007) 2−12. 2003.

[32] Hersh W, Helfand M, Wallace J, et al. A systematic review of the efficacy of telemedicine for making diagnostic and management decisions. J Telemed Tele ca re 2002; 8: 197−209.

[33] Hoegh RT, Skov MB (2004b) Exploring context-awareness as mean for supporting mobile work at a hospital ward. In: CDProceedings of the 7th international conference on work with computing systems (WWCS 2004).

[34] Heath C., Vom Lehn D., Hindmarsh J., Svensson M., Sanchez, and P. Luff. Configuring Awareness. Computer Supported Cooperative Work. An International Journal, 11(3-4):317−347, 2002.

[35] Huang E. M., D. M. Russell, and A. E. Sue. Im here: public instant messaging on large, shared displays for workgroup interactions. In CHI 04: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 279−286, New York, NY, USA, 2004. ACM Press.

[36] Hansen T. R., J. E. Bardram, and M. Soegaard. Moving Out of the Lab: Deploying Pervasive Technologies in a Hospital. IEEE Pervasive Computing, 5(3):24−31, July-September 2006.

[37] Hristova A., Conceptualization and Design of a Context-aware platform for user-centric Applications, Master's thesis in Security and Mobile Computing, Norwegian University of Science and Technology, 2008.

[38] Indulska, J., Sutton, P. (2003). Location management in pervasive systems. In Proceedings of the Australasian information security workshop conference on ACSW frontiers (Vol. 21, pp. 143−151).

[39] Intille S.S., Rondoni J., Kukla C., Iacono I., Bao L., A context-aware experience sampling tool, in: Proceedings of the Conference on Human Factors and Computing Systems, 2003.

[40] Intille S.S., Bao L., Munguia Tapia E., Rondoni J., Acquiring in situ training data for context aware ubiquitous computing applications, in: Proceedings of CHI 2004 Connect: Conference on Human Factors in Computing Systems, ACM Press, April 2004.

[41] Jones G. J. F. and Brown P. J. Context-aware retrieval for ubiquitous computing environments. In Mobile HCI Workshop on Mobile and Ubiquitous Information Access, pages 227−243, 2003.

[42] Kidd, A.G., Sharratt, C. and Coleman, J. Mobile communication regulations updated: how safely are doctors telephones used? Quality and Safety in Healthcare, 13, 6 (2004), 478.

[43] Korel, B.T., Kao, S.: Addressing context awareness techniques in body sensor networks. In: 21st International Conference on Advanced Information Networking and Applications Workshops 2007, pp. 798−803 (2007).

[44] Korhonen, I., Paavilainen, P., Srela, A.: Application of ubiquitous computing technologies for support of independent living of the elderly in real life settings. Proc. UbiHealth 2003, The 2nd International Workshop on Ubiquituos Computing for Pervasive Healthcare Applications (2003).

[45] Korkea-aho M. Context-aware applications survey. Technical Report Internetworking Seminar (Tik-110.551), Helsinki University of Technology, 2000.

[46] Kjeldskov J, Skov MB, Als BS, Hegh RT (2004) Is it worth the hassle? Exploring the added value of evaluating the usability of context-aware mobile systems in the field. In Proceedings of the 6th international conference on human computer interaction with mobile devices and services (MobileHCI 2004), Springer, Berlin Heidelberg New york, LNCS 3160, pp 61−73.

[47] Lieberman H., Selker T., Out of context: Computer systems that adapt to, and learn from, context, IBM Systems Journal; 2000, 39 3/4, pg. 617.

[48] Mizzaro S., Nazzi E., Vassena L., Retrieval of context-aware applications on mobile devices: How to evaluate?, Proceedings of the second international symposium on Information interaction in context, USA, 2008.

[49] Matthias Baldauf, A survey on context-aware systems, Int. J. Ad Hoc and Ubiquitous Computing, Vol. 2, No. 4, 2007, pp. 263−277.

[50] Mitchell S., Spiteri Mark D., Bates J., Coulouris G., Context-aware multimedia computing in the intelligent hospital, Proceeding 9th workshop on ACM SIGOPS, ACM New York.

[51] Munoz M. A., Rodriguez M., Center J. F., A. I. Martinez-Garcia, and V. M. Gonzalez. Context-Aware Mobile Communication in Hospitals. IEEE Computer, 36(9):38−46, 2003.

[52] Myerson SG, Mitchell AR. Mobile phone in hospitals. British Medical Journal 2003;326:460−1.

[53] Mihailidis A., Carmichael B., Boger J., Fernie G., An intelligent environment to support aging-in-place, safety, and independence of older adults with dementia, in: Proceedings of the Second International Workshop on Ubiquitous Computing for Pervasive Healthcare Applications (UbiHealth2003), Seattle, 2003.

[54] MacIntyre B., Mynatt E. D., Vodia S., Hansen K. M., Tullio J., and Corso G. M. Support for Multitasking and Background Awareness Using Interactive Peripheral Displays. In Proceeding of ACM User Interface Software and Technology 2001 (UIST01), pages 1114, Orlando, Florida, USA, Nov. 2001.

[55] Moran T, Dourish P (2001) Introduction to this special issue on context-aware computing. Hum Comput Interact 16:87−97.

[56] Meggelen J.V., L. Madsen, J. Smith, Asterisk: The future of telephony, Second edition, O'Reilly Media.

[57] Nardi B., Whittaker, S., and Bradner, E.. Interaction and Outeraction: Instant Messaging in Action. Proceedings CSCW 2000. (December, 2000), 79−88. New York: ACM Press.

[58] Pascoe, J., Ryan, N., and Morse, D. Using while moving: HCI issues in fieldwork environments. ACM Transactions on Human-Computer Interaction 7, 3 (Sept. 2000), 417−437.

[59] Pascoe J. Adding generic contextual capabilities to wearable computers. In: Proceedings of 2nd International Symposium on Wearable Computers, 1998; 92−99.

[60] Pascoe, J., Ryan, N.S., Morse, D.R. Human-Computer-Giraffe Interaction HCI in the Field. Workshop on Human Computer Interaction with Mobile Devices (1998).

[61] Patrice A. Spurck, Mary L.Mobr, Martha Stoner, The Impact of a Wireless Telecommunication System on Time Efficiency, Vol.25, No.6 JONA,1995.

[62] Prekop P., Burnett M., Activities, context and ubiquitous computing, Comput. Commun. 26 (2003) 1168−1176.

[63] Rouncefield M., Viller S., Hughes J., and Rodden T. Working with constant Interruption: CSCW and the Small Office. The Information Society, 11(4):173−188, 1995.

[64] Rodriguez M., Favela J., Martinez E. A., and Muoz M. A., Location aware access to hospital information and services, IEEE Trans. Inform. Technol. Biomed., vol. 8, no. 4, pp. 448−455, Dec. 2004.

[65] Row Want and Andy Hopper. Active badges and personal interactive computing objects. IEEE Transactions on Consumer Electronics, 38(1):10−20, Feb 1992.

[66] Reddy M., Pratt W., Dourish P., Shabot M., Sociotechnical requirements analysis for clinical system methods, Inf. Med.4 (2003) 437−444.

[67] Reddy M., Dourish P., and Pratt W. Coordinating heterogeneous work: Information and representation in medical care. In Prinz et al. [15], pages 239−258.

[68] Rodden T., Cheverst K., Davies N., Dix A (1998) Exploiting context in HCI design for mobile systems. In: Proceedings of the 1st workshop on human computer interaction with mobile devices.

[69] Rodrguez M. and Favela J., Autonomous Agents to Support Interoperability and Physical Integration in Pervasive Environments, Proc. Atlantic Web Intelligence Conf., Springer-Verlag, 2003, pp. 278−287.

[70] Rindfleish TC (1997) Privacy, information technology, and health care. Commun ACM 40(8):93−100.

[71] Schilit B. and Theimer M., Disseminating Active Map Information to Mobile Hosts. IEEE Network, 1994.

[72] Schilit B., Adams N. and Want R., Context-Aware Computing Applications, in 1st International Workshop on Mobile Computing Systems and Applications. 1994.

[73] Skov M, Hoegh R (2006) Supporting information access in a hospital ward by a context-aware mobile electronic patient record. Journal of Perv. and Ubiq. Computing 10:205−214.

[74] Spurck PA, Mohr ML, Seroka AM, Stoner M. The impact of a wireless telecommunication system on time efficiency. Journal of Nursing Administration 1995 Jun, 25(6):21−6.

[75] Solvoll T., Hartvigsen G., Tiemersma A., Kerbage E., Fasani S., Ravuri A. B., Context-sensitive Communication in hospitals: a user interface evaluation and redesign of Ascom wireless IP-DECT phones, eTELEMED 2011: 37−46.

[76] Solvoll T., Scholl J.: Strategies to reduce interruptions from mobile communication systems in surgical wards. J Telemed Telecare 2008, 14(7):389−92.

[77] Solvoll T, S. Fasani, A. B. Ravuri, A. Tiemersma, G. Hartvigsen, Evaluation of an Ascom/trixbox system for context sensitive communication in hospitals, in Scandinavian Conference on Health Informatics, 2010: 49−53.

[78] Scholl J., Hasvold P., Henriksen E., Ellingsen G., Managing Communication Availability and Interruptions: A Study of Mobile Communication in an Oncology Department, in: A. LaMarca, M. Langheinrich, K.N. Truong (Eds.), Pervasive Computing, Springer, Berlin, 2007, Heidelberg.

[79] Siewiorek D., et al. SenSay: A Context-Aware Mobile Phone. in Proceedings of the 7th IEEE International Symposium on Wearable Computers. 2003.

[80] Swanson E., Galvao A., Sato K., A framework for understanding contexts in interactive systems development, in: 7th World Multi-Conference on Systemics, Cybernetics and Informatics, Orlando, FL, July 2003.

[81] Spreitzer M. and Theimer M. Scalable, secure, mobile computing with location information. CACM, 36(7):27, July 1993. In Special Issue, Computer-Augmented Environments.

[82] Sohn T., K. A. Li, W. G. Griswold, and J. D. Hollan. A diary study of mobile information needs. In CHI 2008 Proceedings, pages 433442, Florence, Italy, Apr. 2008.

[83] Schmidt K, Heath C, Rodden T (2002) Preface to special issue on awareness. Comput Support Coop Work 11:iii-iv.

[84] Sousa J. P. and D. Garlan. Aura: an Architectural Framework for User Mobility in Ubiquitous Computing Environments. In Proceeding of the 3rd Working IEEE/IFIP Conference on Software Architecture, Montreal, 2002.

[85] Van Lieshout EJ, van der Veer SN, Hensbroek R, et al. Interference by new-generation mobile phones on critical care medical equipment. Crit Care. 2007;11(5):165.

[86] Van Bemmel JH, Musen MA (1997) Handbook of medical informatics. Springer, Berlin Heidelberg New york.

[87] Whittaker, S., Frohlich, D. Daly-Jones, W. (1994). Informal Workplace Communication: What is it Like and How Might We Support It? Proceedings of CHI94 Conference on Human Factors in Computing Systems, 131−137, ACM Press: New York.

[88] Winograd, Terry, Architectures for Context, Human-Computer Interaction, 16:2−3, 2001.

[89] wirelesscommunication.nl, Digital Enhanced Cordless Telecommunications (DECT), [Accessed 10/12/2009]: http://www.wirelesscommunication.nl/reference/chaptr01/telephon/dect.htm.

[90] Xiao Y., C. Lasome, J. Moss, C. Mackenzie, and S. Faraj. Cognitive properties of a whiteboard: A case study in a trauma centre. In Prinz et al. [24], pages 259−278.

[91] Yee S. and K. S. Park. Studiobridge: using group, location, and event information to bridge online and offline encounters for co-located learning groups. In CHI 05: Proceedings of the SIGCHI conference on Human factors in computing systems, pages 551−560, New York, NY, USA, 2005. ACM Press.

[92] http://fonality.com/trixbox/wiki/how-big [Accessed 02 may 2011]

[93] http://en.wikipedia.org/wiki/Private_branch_exchange#Private_branch_exchange [Accessed 02 may 2011]

[94] http://www.cisco.com/web/DK/assets/docs/StOlavsHospitalCiscocasestudy.pdf [Accessed 24 may 2011]

[95] http://asterisk-java.org/ [Accessed 24 may 2011]

[96] http://en.wikipedia.org/wiki/SOAP [Accessed 24 may 2011]