INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

MOBI-COLLAB: Aplicações de Colaboração para Dispositivos Móveis

MOBI-COLLAB: Collaborative Applications for Mobile Devices

João Filipe Vieira de Sousa

Dissertação para obtenção do Grau de Mestre em

Engenharia de Redes de Comunicações

Júri

Presidente:	Prof. Rui Jorge Morais Tomaz Valadas
Orientador:	Prof. Luís Manuel Antunes Veiga
Vogais:	Prof. João Dias Pereira
	Eng. Paulo Chainho

Outubro 2010

Acknowledgements

I would like to express my deepest gratitude to my teacher advisor, Prof. Luís Veiga for his invaluable support and guidance throughout the research and during the writing of this document.

I also would like to thank Eng. Paulo Chainho, my advisor from PT Inovação, from whom I extend my sincere thanks for his tips and advices, during the design and development of this work.

Much of the success of this work, I owe to my colleagues from PT Inovação: Mauro, Neutel, Pedro Taborda, Hugo, Bernardo and Pedro Gomes. Their help during my learning of the technologies, and the development phase of this work was invaluable.

Finally, I would like to thank my friends and family. Without their friendship and patience, it would be impossible to carry out this work. And my special thanks, go for my girlfriend, Sandra. Her love and affection inspired me during my darkest and uninspired days.

Lisboa, October 17, 2010
João Sousa

Resumo

Hoje em dia, os dispositivos móveis são dotados de uma capacidade de processamento cada vez maior. Ainda para mais, os níveis de conectividade cada vez mais crescentes neste tipo de dispositivos, motivam-nos a desenvolver novas soluções que tirem partido das tecnologias móveis.

A pesquisa na área de *groupware*, ou colaboração, detem uma longa história no desenvolvimento de frameworks que suportam o trabalho colaborativo através de infra-estruturas fixas. Novas soluções a serem desenvolvidas neste âmbito, devem tirar partido das capacidades dos ambientes móveis, relativamente ao acesso à informação em qualquer tempo, e em qualquer lugar.

A PT Inovação, encontra-se a desenvolver neste momento, a plataforma PUC (Plataforma Unificada de Colaboração), com o intuito de fornecer uma arquitectura unificada de serviços, que podem ser usados por aplicações clientes externas, que queiram fornecer funcionalidades colaborativas, aos seus utilizadores.

O projecto Mobi-Collab é também parte desta iniciativa. Ademais, o seu intuito é de forma geral o mesmo, e integra-se com a própria plataforma PUC. No entanto, o seu foco é no tratamento dos problemas inerentes ao desenvolvimento de soluções de groupware para dispositivos móveis.

A nossa pesquisa centra-se sobretudo, nas áreas de groupware, mobilidade e middleware. Propomos uma arquitectura que incluí uma Gateway que se integra com a plataforma PUC e que media a comunicação entre clientes móveis e os serviços da plataforma. Propomos também, uma camada de middleware, a ser executada nos sistemas operativos alvo dos terminais móveis que fundamentalmente oferece um conjunto de APIs para os desenvolvedores de aplicações móveis poderem criar aplicações colaborativas multi-plataforma.

A nossa implementação segue uma abordagem de desenvolvimento baseada em tecnologias Web (AJAX, JavaScript, JSON) nas várias camadas de software constituintes, bem como, nos mecanismos de transporte na rede, de forma a alcançar um elevado nível de portabilidade.

Consideramos que os resultados obtidos durante a fase de avaliação da nossa solução são bastante encorajadores, quer a níveis de performance, como qualitativos. Fomos bem sucedidos em fornecer uma solução multi-plataforma que satisfaz tanto os utilizadores de aplicações colaborativas, como os desenvolvedores de aplicações. A nossa solução introduz um atraso na execução que é pequeno o suficiente para não causar impacto nas aplicações que a usem, e é portável ao ponto de termos conseguido uma compatibilidade quase total e sem mudanças, na maioria dos sistemas operativos alvo.

Abstract

Today, mobile devices are becoming more powerful, and the ever increasing connectivity of this type of devices motivate us to look into what can be achievable by mobile technologies.

The groupware research topic has a long history in the development of frameworks which supported collaborative group work, through fixed infrastructures. New collaborative groupware solutions must take advantage of the capability for anytime, anywhere, information access provided by mobile environments.

PT Inovação is developing PUC (Plataforma Unificada de Colaboração) - a collaboration platform, in an effort of providing an unified service architecture which may be used by external client applications, who want to provide collaborative functionality to the end-user, without worrying about the lower-level issues.

The Mobi-Collab project is also part of this initiative as it follows the same principle and is tightly integrated with PUC itself, but, with the focus of addressing the issues of *mobile groupware*.

We survey topics related with groupware functionality, mobility issues and middleware, and propose a system architecture that comprises a Gateway that integrates with the PUC Platform and mediates the communication of accessing mobile terminals to the platform's services, and a middleware layer to be executed in targeted mobile operating systems, that integrates with supporting technologies and provides a set of APIs for developers to easily develop cross-platform collaborative applications.

Our implementation takes a Web development approach, capitalizing on the ubiquity of Web technologies (AJAX, Javascript, JSON) in both the software components and network transport mechanisms in order to achieve a high level of system portability.

We feel that we have obtained very encouraging results, both performance-wise and qualitatively, and succeeded in delivering a cross-platform solution that satisfies both users and developers of mobile groupware applications that result in a small overhead for client applications and a portable solution that offers almost full compatibility without changes for most of the targeted mobile operating systems.

Palavras Chave
Keywords

Palavras Chave

Colaboração
Groupware
Mobilidade
Middleware
Portabilidade
Widgets

Keywords

Collaboration
Groupware
Mobility
Middleware
Portability
Widgets

Table of Contents

List of Figures

List of Tables

1 Introduction

"The most profound technologies are those that disappear: they weave themselves into fabric of everyday life until are indistinguishable from it."
– Mark Weiser, The Founder of Ubiquitous Computing. 1952-1999

1.1 Background and Motivation

Today, to find someone who does not own and use a mobile device (from cellphones, to PDA's and laptops), in a daily usage basis is a rare occurrence. The inherent mobility of this type of devices make us detach from fixed and constrained working environments, making these devices a kind of extension to ourselves.

Recent developments in mobile technology, related not only to greater and greater processing and functional capabilities of current mobile devices, but also, the always increasing connectivity provided by current mobile networks (GPRS, 3G, WiMax) motivate us to look into what can be achievable by mobile technologies.

Considering these factors, the concept of ubiquitous computing, described by Mark Weiser [44], in his early research,is today, more of a reality as it was years ago.

People commonly collaborate with each other, in their work life or simple daily social routines. Collaborative software aims to support this kind of interaction. Collaboration itself implies **group interaction**. Which is commonly addressed by *groupware* applications.

Fixed groupware solutions constrain us to fixed work environments. Which by itself, is a hindrance to our everyday work and social life, where most of the time is passed on face-to-face interactions in multiple locations.

Therefore, it is desirable, as it is natural, that, new collaborative groupware solutions take advantage of the theoretical capability for anytime, anywhere, information access of mobile environments, to provide long distance on-the-move teamwork.

PT Inovação is developing a collaboration platform in an effort of providing an unified service architecture which may be used by external client applications, who want to provide collaborative functionality to the end-user, relieving developers from the intricacies of common challenges in the development of collaborative applications, such as: access control; concurrency control; session management and access control to collaborative resources. This platform is called PUC (Plataforma Unificada de Colaboração).

The Mobi-Collab project is also part of this initiative as it follows the same principle and is tightly integrated with PUC itself. Mobi-Collab however, is focused on the delivery of a common framework for the development of collaborative applications in mobile devices, using PUC as a supporting platform. Although the core groupware challenges are also subject of study in our work, we are mostly interested in addressing the issues of *mobile groupware* and deliver a solution that explores the exciting and increasing capabilities of today's mobile devices in order to improve on-the-move teamwork to its users and ease the developers lives by delivering a cross-platform solution that relieves them from the common mobile groupware and mobility issues.

1.2 Limitations of Current Solutions

Although current solutions that have been developed in mobile groupware research, have been successful in delivering frameworks that properly mitigate the issues found in mobility and mobile environments [5, 20, 40, 37, 29, 43]. One issue still needs to be properly addressed by current solutions. This issue, is the issue of **portability**.

Portability is one of the main requirements found in middleware for the development of mobile applications [31, 2]. It is a crucial requirement if we want to develop and deploy our services faster and expect a wider adoption by users of mobile applications.

Current mobile groupware solutions have been developed in specific programming languages which are tightly integrated with the native APIs of the targeted operating systems. In these solutions, the portability of the developed applications has not been an object of validation due to its difficulty or impossibility. This limitation is addressed in our solution by exploring the Mobile Web Development approach that will properly be described in our State of the Art chapter.

The **dependence on customised protocols** in the communication of the mobile devices with external services is still evident in most mobile groupware solutions. This hinders the re-usability of the developed components and the integration with services from other solutions. Our solution also addresses this issue, by capitalising on the usage of lightweight Web services and asynchronous transport mechanisms that are widely used on the Web.

1.3 Desired Properties

In order to comply with our motivation and address the above mentioned limitations, our solution must have the following properties:

- **Be Developer Friendly:** by abstracting both, groupware functionality and supporting technologies with a well defined Public API, that relieves developers from knowing how things work in below layers and focus only on the application's business logic.
- **Be Extendable:** by providing a library structure and development environment that is easily extendable with new functionality.
- **Be Portable:** by exploring available tools and development approaches in order for application developers to easily port applications that use our framework.
- **Be Lightweight:** by minimising the amount of transmitted data over mobile networks; battery consumption levels and execution times in order for the middleware layer to not become a significant overhead for the application.

1.4 Objectives and Contributions

Our work comprises the following objectives:

- Development of a server-side component (Gateway) that integrates with the PUC Platform and mediates the communication of accessing mobile terminals to the platform's services. Additional features which are essential for mobile clients, must be supported, including data conversion for a lightweight data model and delivery of collaborative resources according to the accessing device capabilities.

- Development of a middleware layer to be executed in targeted mobile operating systems, that integrates with supporting technologies and provides a set of APIs for developers to easily develop cross-platform collaborative applications.
- Development of a prototype application, that validates the above mentioned server-side and middleware components, and that must also be easily extended and re-usable for future implementations.

As for the prototype application, we also propose the accomplishment of the following set of functional requirements:

- **Group management:** group creation, deletion and edition;
- **Session management:** collaborative session creation; deletion; configuration; group association; management of user roles and policies.
- **Awareness features:** sharing of presence information; enriched user profiles; notifications and device capabilities management.
- **Synchronisation:** synchronisation with internal contacts(phone's address book) and synchronisation with external contacts (social networks, search services).
- **Data management:** includes mechanisms for local data persistence and caching; data consistency control when synchronising with external entities.
- **Access to collaborative resources:** includes collaborative text edition; audio-video conferences; collaborative slideshows; whiteboard and desktop sharing.

We hope that our solution contributes to a standardisation in the development of multiple collaborative applications, inside or outside PT Inovação's context and that the research and development during the course of this work, results in a valuable contribution to future implementations that execute in different devices and environments, being set-top boxes an example.

The PUC platform is still in a on-development phase. Therefore we hope that our solution also contributes to validate its core infrastructure and build an indispensable knowledge base, in order to improve it even more.

1.5 Document Structure

This document is constituted by five more chapters next to this introductory chapter.

In chapter 2 we will present our State of The Art analysis. It includes an extensive survey in topics that comprise the foundations of CSCW and groupware; mobile environments and mobile groupware. This chapter also includes a study about the features of currently available and most used mobile operating systems and the development approaches that are currently being followed in the development community in order to maximise the portability of mobile applications.

In chapter 3 we address our work's conception phase, by presenting the architectural designs of our solution's components. Here, we detail the communication processes between the components and address each of them individually, going from the PUC Gateway to its integration with the PUC Platform and the mobile terminal middleware layer.

Next, in chapter 4 we will address the challenges and relevant details of our implementation phase, in order to know how we turned our architectural designs into software implementations that fulfil our objectives.

Our implementations will be put to the test in chapter 5, in an evaluation based in quantitative metrics for both the server-side and client-side implementations. We also evaluate our client-side implementation qualitatively in order to produce a solution that properly satisfies both users and application developers.

Finally, in chapter 6 we will review all the phases of this work and this document's chapters and present our final conclusions; assert if the proposed objectives were accomplished and propose possible future work.

Additionally, this document also includes appendices that support our evaluation chapter with additional test results (apendix chapter A); a view of the developed widgets in our prototype application (apendix chapter B) and photos taken during tests of our application running on real mobile devices (apendix chapter C).

2 State of The Art

"In 20 or 30 years, you'll be able to hold in your hand as much computing knowledge as exists now in the whole city, or even the whole world."
– Douglas C. Engelbart, early computer pioneer, inventor of the computer mouse, 1997.

2.1 CSCW or Groupware?

The concept of *Computer Supported Cooperative Work* (CSCW), was first mentioned in Douglas C. Engelbart's and William K. English's work [17]. CSCW Research efforts in the coming years were based on the foundations of the concepts here introduced:

- **Work stations**
- **Shared file systems**
- **Conferencing tools**

To put it simply, work stations gave users the possibility of working in a computer-assisted manner, with the help of keyboards, mouses and monitors - hence *computer-supported*. Shared file systems and conferencing tools (like cameras and CRT displays), gave users the ability to cooperate - hence *cooperative-work*.

Past efforts in CSCW technology after early ones, focused in building efficient frameworks that could integrate existing conferencing hardware with applications that could enhance: interaction, coordination, distribution, visualisation, flexibility, and human and social factors in collaborative work[32, 34, 18].

The *Groupware* concept commonly appears as a synonym of CSCW. However, this concept does not take into account the intricacies of work station technologies like CSCW does. Tough, it does take computer and network based technology. Ellis et al. [16] stated that groupware can be viewed as:

"computer based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment"

Work station design is not our concern. However, our concern is to provide means that give users the capability of working cooperatively in a shared environment,and also, take advantage of the existing communications and computer technology.

It is important to note that groupware applications may or may not specifically support collaborative work. We are particularly interested in the ones that support it. From now on, we assume that our work is in the same context as groupware. Consequently, related issues to this topic, may also apply to our work.

2.2 General Groupware Support

Due to the sharing and dissemination of information in its applications, groupware research has been overlapped with many disciplines. In [16], there is mention to at least, five key disciplines or perspectives for successful groupware: distributed systems, communications, human-computer interaction, artificial intelligence and social theory.

We will thereby present the key perspectives in groupware development.

2.2.1 Key Perspectives

The **distributed systems perspective** comes naturally due to the distribution of users in time and/or space in groupware systems. Decentralisation in data and control occurs. Therefore we need to explore means to maintain consistency of a global system state and properties. That is, common distributed systems research topics, such as, access control, concurrency control and session and data management appear as essential groupware requirements.

Communications emphasise the exchange of information between remote agents. Thus being a primary concern in development of groupware applications that support distributed interactions. The challenge here is to achieve the same level of effectiveness achieved by face-to-face interaction. Thus, the study of network protocols, how to deal with variable bandwidth and connectivity issues, are some of the concerns we must deal with in the development of groupware systems.

Human-computer interaction and social theory deal with ways to improve interaction within the context of multi-user or group interfaces. Adapting common single-user interfaces is not enough. Factors such as organisational structure and group dynamics influence the design of groupware applications. Systems must be built around the social principles accepted by what is perceived by humans common sense in their everyday work. For example, Moran and Anderson defined a paradigm [25], consisted by a taxonomy that described interaction from the point of view of the actor immersed in what is called "The Workaday World" to help clarify and motivate future design decisions.

The **Artificial Intelligence Perspective** takes an augmentative approach, allowing information to be processed by intelligent agents in ways that enhance group interaction. This is an extremely complex perspective that goes beyond the scope of work, thus it will not be a focus of our state of the art analysis.

Taking these key perspectives at hand, we will now make an introduction to groupware systems and present general concerns that have been the focus of the groupware research efforts along these years.

2.2.2 Groupware Systems Overview

Groupware systems have been conceived according to two taxonomies. One is *time space taxonomy*, the other is *application-level taxonomy* [16].

Time space taxonomy can be divided in:

- **Same place/same time:** corresponds to face-to-face interaction - this is where meeting room tools would be.
- **Same place/different times:** local asynchronous interaction, an example is a physical bulletin board.
- **Different places/same time:** synchronous distributed interaction, where a real-time network based document editor would be included.

- **Different places/different times:** asynchronous distributed interaction, being an electronic mail system an example.

According to our objectives, our work is situated in distributed interactions. As such, **our focus will be on this type of interactions in the respective synchronous and asynchronous modes.**

Application-level taxonomy distinguishes groupware systems according to application-level functionality. Examples include collaborative document editors, Group Decision Support Systems, Conferencing tools, etc.

The architectural design of a groupware system commonly takes two approaches [32]:

- **Centralised approach:** the *client-server* model is followed. All system components responsible for cooperation management and resource sharing run as server processes on dedicated server machines. Local terminals, then communicate with the central components to perform each one of their tasks, even if they could be performed locally. Main advantage is that data is always kept in a consistent state. However, network traffic is high since all data must be uploaded to the centralised server. Not only it becomes the communication's bottleneck, but also, a single point of failure.
- **Distributed approach:** the *peer-to-peer* model is followed. All system components run in local terminals, with each one, having the same functionality as the other. Here consistency issues arise, given that, each agent has a local replica of the application's data and there is not a centralised server which holds the master replica. Thus, agent communication protocols must be built to propagate data updates and maintain consistency in the system. This approach may be more complex, but, the network load is reduced considerably, because it allows for *offline work* by local terminals, and communication is only necessary when data has to be updated in the whole system. This approach is then ideal for the heterogeneity seen in mobile environments [30].

A recent trend, specially in mobile collaborative applications has been to use an **Hybrid P2P - Client/Server approach**, as seen in [45]. Systems developed with this architecture explore the scalability associated with the underlying P2P networks and can, simultaneously take advantage of the better management and security of centralised services.

2.2.3 Groupware Challenges

We designing a Groupware application we see ourselves facing challenging issues. These issues are complex and require attention to a myriad of investigation areas that have been subject of many studies in years past. In this section we present a survey of common issues found in the development of Groupware frameworks over the years.

Access Control

Collaborative groupware makes for potentially chaotic environments. Just the presence of other users with their own goals and experiences introduces uncertainty, unpredictability and surprise in our system.

The purpose of access control is to limit chaos in collaboration. The first research efforts for access control in collaborative systems focused on the adaptation of general methods used in multi-user systems, being the access matrix model one example [22]. In this model a protection state is defined by the triple (S, O, A), where *S* is a set of *subjects* (entities that access data), *O* corresponds to *objects* (data units that may be accessed). The *access rights matrix A[s,o]* , defines for each correspondence of subject with object the respective access rights. Access rights may be based in a set of privileges that are verified by an access checking rule that verifies if a certain privilege is right in a given A[s,o]. An example of a privilege may be the right to read or write a given data object.

Although this type of methods might work to guarantee some form of access control, they simply are not enough to govern collaborative environments, given that the exact nature of the subjects, objects and access rights is not supported. Over time, groupware research efforts concentrated in creating methods that could provide a more dynamic access control scheme. Users in collaborative systems take multiple and dynamic roles, as such, access control methods that could infer access rights from user roles in the system were needed. Collaborative groupware had grown in scale and complexity. As such, the needs for flexibility, easier specification and automation had to be considered.

Shen and Dewan [38] presented an access control model that extended the previous access matrix model to a 4-tuple (S,O,A,F), where F represented an inference function that allowed inference of rights that had not been explicitly specified in A[s,o]. Variables in this system, including subjects and even access rights, had to support an inheritance relationship. Access checking rules were based on the inference of the inheritance relationships of a given variable. Put simply, suppose that access rights: "ReadR" and "WriteR" are children of "DataR". If a given subject *s* wants to access object *o* in a "ReadR" operation, and it has the right "DataR" explicitly specified in its access matrix, the right "ReadR" will be inferred by the inference function, thus giving *s* the right to perform operation *o*. This model introduced a new level of complexity in access control for collaborative systems, though it contributed to create a system that offered a good level of flexibility, allowing for the definition of collaboration based rights and multiple and dynamic user roles.

The most common approach is the one based on *policies and roles* [15]. Here *policy* is defined as intelligence about how to respond to particular occurrences in collaborative situations. As an example of a common workaday world policy, we might have:

"Don't let anyone bother me when i'm working on my thesis. Unless it's my advisor of course."

Although Shen and Dewan's access control method could support the notion of the "advisor" role in this example, it could not support the policy behind this example. The goal with the more recent access control methods consists in adopting the model of access control perceived in human workaday activities. Thus, collaborative systems tend to provide simple user interfaces or even specification languages for the definition of user roles and the associated policies based on access rights that will exist for the set of resources and attributes available. An example is the Intermezzo framework [13].

Awareness

Another critical factor to achieve success in group interaction is awareness. Defined in [11] by Dourish and Bellotti as:

"an understanding of the activities of others, which provides a context for your own activity"

These authors also proposed, two dimensions for awareness. A high-level awareness of the *character* of a subject's actions permits participants to structure their activities, avoiding duplication of work. Other, a low-level awareness of the *content* of a subject's actions allows for fine tuning in group work division.

There are various approaches to the provision of awareness information. This information can be *explicitly generated*, that is, the information is directed and separated from the shared work object. One way of doing this is by *explicit roles*, take for example, the INTERMEZZO framework based on the previously mentioned notion of *policies and roles* [15, 13], that provides an activity-based awareness monitor constructed on top of an object model that distinguishes users by their roles in a given activity. The other approach is by *passively collected and distributed information*, that is, the awareness information is presented in the same shared workspace as the object of collaboration. MMConf [6] used this approach, by providing conference-awareness mechanisms using state queries, floor control policies and multicast functions, all applied in the same conference and participating users.

Apart from operations related to the creation, modification and presentation of information, awareness operations are among the most utilised by groupware users, as stated in [1].

Coordination

Since the beginning of groupware research, coordination has been a primary concern. The need to avoid conflicting access to shared resources appeared during synchronous collaborative sessions, when the use of classical concurrency control methods appeared inappropriate because they hindered tightly coupled teamwork [16], due to the small granularity and high frequency of the concurrent actions.

One way to perform coordination in collaborative systems is by the use of *floor control*. Generally, an exclusive access right is granted to a given user for a shared resource, that is, the user winning the *"floor"* has the right among all other competing users to perform the available operations on the competed resource. Why, and how the user wins, depends on the underlying floor control mechanism.

Dommel and Garcia-Luna-Aceves [9, 10] presented a floor control model supported by a series of floor control protocols that fitted in large-scale Internet conferencing systems. Their floor control model consisted in a *collaboration environment* CE = (S,U,R,F), being respectively *sessions*, *users*, *resources* and *floors*. Also, *collaborative sessions* were defined by CS = (sid, D,U,R, F), with *sid* being a session identifier with a duration of *D* and respective session users, resources and floors. A given user *U* can be the floor originator *FO* of a resource *r*, that is, the node that injects r into a session and initiates floor control for r; the floor coordinator *FC*, that is, an arbiter for a resource r, or a session moderator who grants or denies the floor for r during session time to the *FH*, who exclusively has the floor on r for a finite time period. A floor *f* is then defined by:

$$f(r,T,FC,FH,ta,l,QoS) = st$$

Being *r* the single resource where the floor applies, *T* the resource type (can be video, audio, text), *ta* a timestamp that shows when the floor has been allocated. *l* corresponds to the time during which the adjunct resource is consumed by FH and inaccessible for others.The *QoS* field may contain a directive on the service quality for using the resource. Finally, *st* corresponds to the floor's current state, and it can be: free, busy, idle, requested or nil. This model layed the foundation by which *floor control protocols* were built on. *Random-access Group Coordination* (RGC) protocols are based on the concept of sensing the status of a remote resource, which can be accomplished either by users tracking each others activities through the user interface, or, by the system. *Scheduled Group Coordination* (SGC) protocols are based on mechanisms such as: token passing, reservations or polling.

It is not only with floor control mechanisms by which we can achieve coordination. Li and Muntz presented COCA [23], a collaborative objects coordination architecture based on *coordination policies*. COCA is a decentralised groupware system that provides a dual-bus collaboration architecture. Consisted by a *conference bus* where computation related to application data takes place, and a *collaboration bus* where the interpretation of coordination policies takes place. These policies are defined using a determined specification language. Participants are explicitly divided into different roles, with individual behaviours and constraints specified in sets of logic rules. Participants are connected via the previously mentioned collaboration bus with each one having a gate where message processing occurs. When a message arrives at one gate, a given participant either: i) does some processing; ii) forwards the same message or new ones; or iii) blocks it. The processing or not of a message is done according to the rules specified in a given user role. Thus coordination, and also, access control is achieved. However, this type of architecture presented some performance problems, due to the simple fact that all messages to and from each participant would run through every participant's COCA virtual machine.

Concurrency Control

If we cannot avoid conflict by coordination, then, we must deal with it. This is where concurrency control comes into place. Research efforts on this topic are commonly found in the context of database systems and parallel and distributed systems. However, concurrency control in collaborative systems takes a completely different approach. Here we must support tightly coupled teamwork with highly

dependent, conflicting actions with very small granularity. Traditional independent database transactions, simply do not allow for this type of interaction. Collaborative applications require transactions that may be of long duration; unstructured or unplanned and more sensitive to response-time performance than total system throughput. As such, research efforts in concurrency control for collaborative applications have been trying to support longer transactions, user control over transaction execution, relaxed consistency criteria, and integration with access control mechanisms.

Munson and Dewan [27] presented a concurrency control framework for collaborative systems that met the requirements mentioned above, but also, permitted high-level definition of application-specific consistency criteria; concurrency policy specification at collaboration time; merge and coupling policies and implicit "read" operations for data dependency awareness.

Session Management

A primary thought in collaborative systems is about how we organise interaction with multiple users, their objectives and group work relationships? In our workaday world, when we think of *session*, we might think of formal or informal meetings where we work cooperatively. In groupware this concept is translated almost literally, being *session management* [14]:

"the process of starting, stopping, joining, leaving and browsing collaborative situations across the network"

Edwards [14] also states that session management takes two forms. One is *explicit session management*, that is, participants in the collaboration are required to take some action when joining a session. This joining action can take two approaches. One, *initiator-based*, consists in a sequence of dialogs that an initiating user performs to invite other users to the collaborative session, with the invited users accepting or rejecting the invitation. Other, is *joiner-based*, with the initiating user creating a new session, making other users that want join him to browse the list of current active sessions. Here, participants use other means to notify each other that they have initiated a collaborative session.

However, there are situations in which a more lightweight form of session management would be desirable. Edwards also defined an *implicit* form of session management. This form is based on *activity information*, that is, the system must be capable of knowing what are the current activities running across the network; the users and the applications that they are currently engaged. The system would simply detect the potential for collaboration when multiple users are working on the same object, establishing a session without the need for an explicit rendezvous process. This form is indicated for serendipitous and transient collaborations as opposed to planned and long-term formal collaborative sessions that the explicit management form permits.

Tailorability

A newer concept in groupware, is the one of *tailorability*. defined in [39] as:

"the activity of modifying a computer application within the context of its use"

This is a key property that allows for modifications of a groupware system, as its use evolves. Users are given the possibility to adjust the software to their personal preferences or according to their tasks.

Tailorability can assume different levels. Mørch [26] defined three levels of tailoring. Tailoring by *customisation* consists of a user selection of pre-defined configuration properties. By configuration properties we might include user interface preferences in display of a user's contacts for example. Another level is tailoring by *integration*, that states that users are able to select functions from a list of available functions. In this way, a user is given the possibility to utilise only functions or services that suit his needs or a given group context. The last one, is a more extreme form of tailoring, called *radical tailoring*. Here the system can extend its own functions, or even new ones, by adding new blocks to the system. The last level requires a system architecture with a greater level of extensibility.

2.3 Mobile Groupware

The greater unpredictability and heterogeneity in the context of mobile work introduces requirements for an unmatched level of **flexibility** and **adaptability** not seen in classical groupware that is dependable on fixed infrastructures.

Luff and Heath [24], revealed the importance of *micro-mobility* in the context of mobile collaboration in their study for the implementation of a mobile collaborative system in a London's Underground construction site. Here, micro-mobility is defined as:

> *"the way in which an artifact can by mobilized and manipulated for various purposes around a relatively circumscribed, or at hand, domain"*

Take for example, a piece of paper. Paper has an extreme level of micro-mobility. It can be read and written in a myriad of spaces and in the most extreme conditions. Today, mobile devices are small, connectible and powerful in such levels that accommodate with this kind of micro-mobility.

Mobility by itself implies that a theoretical ideal of *access, anytime, anywhere* is achieved. This ideal was debunked in a study of mobile workers behaviours by Perry et al. [28]. From a mobile worker's point of view, the ideal assumes the following form:

- **Information access:** how information is used and whether it is in the appropriate form for viewing and interaction.
- **Access anytime:** the cost in time units when accessing information.
- **Access anyplace:** levels of granularity in terms of how artifacts are moved to support work (in this case the mobile device itself), this is also related with the previous mentioned micro-mobility.

By taking this perspective into account we can see that achieving a groupware experience in mobile devices comparable with the desktop counterpart is not an easy task. Mobile devices still present some serious challenges in interaction and display of information (due to their small size) tough they are much easier to move and use in smaller units of time, thus being ideal to accomplish the *access anytime, anywhere* goal than a laptop for example.

Immediate and easier access to information is an always pressing need for us, human beings. Thus, the shift to mobile collaborative work is natural, albeit, a challenging one.

In section 2.3.1 we will see the challenges imposed by mobile environments in the development of mobile groupware. Next, in section 2.3.2 we will look into how mobile middleware can help us overcome these challenges. Finally, we end this section by mentioning examples of mobile groupware frameworks and platforms, where specific middleware solutions have been deployed (see section 2.3.3).

2.3.1 Mobile Groupware Challenges

As stated in the section above, the mobility of participants offers new opportunities in collaboration. However, we must consider challenging issues [12, 35, 36]:

- **Communication issues:** Mobile networks bring more issues to the table than fixed networks due to the mobility of participants. Here, we must consider ad-hoc routing, mobility management, and service discovery, mainly if we are considering a completely distributed design in our groupware architecture. If issues arise we must consider how our network infrastructure integrates with solutions like Mobile-IP or SDP. Also, mobile networks are not as reliable as their fixed counterparts. Connection may be lost at any moment. Thus, the system must be capable of a *disconnected mode*

of operation. The mobile user must be able to continue his/her work, even with limited functionality. Considering bandwidth, current mobile networks are still very limited. Special care is needed when migrating features seen in current Web based applications, mainly when looking at the introduced overheads when using Web services technologies.

- **Data distribution and consistency:** if a mobile user is in *disconnected mode* and then switches to *connected mode* the system must be able to use the changes made locally by the user while he was disconnected. When synchronisation is performed, concurrency and consistency issues arise. For this, systems that deal with weakly consistent replicated data could be used, one example is the session guarantees used in the Bayou system [42, 8]. A newer and more collaborative-environment oriented system, is the one proposed by Suleiman et al. [41], that explicitly models procedures for the disconnection and reconnection of mobile terminals in a distributed groupware architecture.

- **User interfaces:** Obviously, mobile devices are very different from desktop computers. We must consider small screens, new input devices and new usage paradigms. We cannot expect to use the same user interface of a desktop groupware application and expect it to accomplish the same level of usability on a mobile device. As Rekimoto says in [33]: *"the most significant difference between personal computers and handheld computers is not the computation power, but the size of the screen"*. Mobile application interfaces must be designed with new techniques and evaluated with new usability criteria [3]. Special care is needed to design an interface that does not frustrate the user while providing effective awareness and interaction mechanisms.

- **Diversity of devices and platforms:** The sheer number of devices with different technological capabilities, computational power and multiple development platforms require that mobile groupware application fulfil difficult adaptability requirements. Krebs and Marsic [21] proposed a framework for the development of adaptive collaboration applications that aimed to solve these challenges by specifying shared data views and user interfaces in independent XML manifest documents. Then, these documents were transformed in device-dependant interface graphs for individual client devices, using predefined device specific mapping rules. In this way it was possible for the application developer to deliver device specific implementations without needing to change code on the application level.

2.3.2 The Middleware's Role

Today, the most significant trend is the requirement of ever **faster service development and deployment**. Thus recently, service and application frameworks and platforms have been developed adopting the definition of middleware, that commonly denotes a set of generic services operating above the operating system. According to [31, 2], mobile systems middleware must provide the following features:

- **Execution support layer:** an architecture that encapsulates the middleware functions to future applications that use it; supporting fast service development and deployment and that easily divides the application logic from the system logic.

- **Persistence support:** mainly to provide functions to application developers to easily use the mobile device's file system for data persistence during disconnected operation.

- **Adaptability:** ability to automatically cope with connection quality changes and even disconnection; explicit user actions or from the monitored environment.

- **Mobile distributed information base:** provisioning of a consistent, efficiently accessible, reliable and highly available information base. Thus implying distribution and replication of data, that must be intelligently synchronised after disconnection.

- **Extensibility:** the deployed middleware must expect that new devices and services will appear over its lifetime, thus its components must be prepared to be modified. Commonly, middleware development takes an object-oriented approach that eases this requirement.

- **Portability:** the middleware system architecture and the approach in development must support its portability to different operating systems (even the ones that were not considered during middleware's development) reducing as much as possible future porting efforts.
- **Fault tolerance:** if the middleware is going to operate in mobile networks, it is desirable that the system provides fault tolerance mechanisms to deal with the inherent unreliability of these type of networks.

2.3.3 Examples of Middleware in Mobile Groupware

The mobile groupware area is an example where the development of middleware has been applied to achieve the objectives mentioned in the previous section. To illustrate this, we will look now at examples of mobile groupware frameworks and platforms.

YCab

YCab [5], is a decentralised groupware framework (see 2.2.2) for mobile collaboration. It is designed for *ad-hoc* wireless networks where there may be no possibility of using a fixed network. YCab's usage goes for collaboration in scenarios such as: rescue efforts, military intelligence and strike teams, field survey operations, construction and staff and security of large events.

It provides a Java based API for developers to quickly create collaborative applications with a minimal learning curve. Along with a set of services provided by the API which an application developer can call by method invocations, there are also customisation options available. An interesting feature is that certain service features, such as service optimisations or state recovery can be enabled or disabled without the need to redesign the services. A developer can take a modular approach in development, creating customisable modules that can be plugged into any application. The system thus offer the possibility to create tailorable groupware (see 2.2.3). YCab separates the framework API and the services provided. The framework takes the responsibility of sharing information from the services in execution with other members in a collaborative session. Each user then runs a local YCab client that consists of the following components:

- **Communication and service managers:** includes a *communication manager* that uses a specific message format and contains incoming and outgoing message queues. It also provides a *client manager* that instantiates the services currently in execution and messages are processed by a *message router* inside the client manager component, that routes all messages to their respective services and also to the communication manager, if messages are to be delivered across the network.
- **Session and election services:** communication protocols that perform group membership management (joining and leaving sessions, leader election) in a multicast environment (see also 2.2.3).
- **State recovery manager:** provides mechanisms that enable clients to be brought to a state that is consistent with other session participants.
- **GUI components:** API methods that provide the instantiation of already built-in GUI components.

MOTION

Kirda et al. [20], presented a service architecture for distributed mobile collaboration called MOTION. It supports a configuration consisting of users in desktop computers, laptops, PDA's and WAP-enabled mobile phones. Offered services include: peer-to-peer file sharing, distributed searches and advanced information subscription and notification.

MOTION's system architecture is composed of the following layers:

- **Communication middleware layer:** offers communication services such as peer-to-peer file sharing through distributed searches, and publish/subscribe mechanisms to the upper layers.
- **Teamwork services layer:** situated directly above the communication middleware layer, this one integrates the data repository, access control (see also 2.2.3), messaging, document management and distributed search service modules while providing a Java based API to the above layer.
- **Presentation layer:** an application developer can then build specific services on top of the Teamwork services API.

Of special interest in this framework is the *publish/subscribe* messaging services. Message delivering is driven by the message content rather than explicit user addresses that users specify. Put simply, a sender *publishes* messages and receivers *subscribe* to message types that are of interest. This is specially useful though, in distributed environments when specifying the addresses of each of all the available users in a given session is not desirable.

Multimedia Mobile Collaborative System

Su et al. [40] concentrated their research in *adaptive content generation and delivery* justifying that most of the collaborative systems in existence focus on the implementation of the collaborative logic engine, that is, content representation in a heterogeneous environment is not clearly addressed. Thus they proposed a multimedia mobile collaborative system based on a content, device and connection aware framework.

An interesting aspect of this work is that to accomplish the goal mentioned above, a single and unified format for data representation had to be developed. As such, an Unified Media Description Language (UMDL) based on XML was presented, embedding multimedia information into a single unified file format (UFF).

The system takes a centralised approach (see 2.2.2), implying the existence of a server process that plays the role of a *content provider* where a *data warehouse* is maintained. Content visualisation is performed by the system clients (desktop, PDA's, mobile phones) which communicate with the centralised content provider via the UMDL described earlier.

Therefore, the proposed system architecture consists of the following layers:

- **Content generation layer:** the content server generates the unified content based on client request. If the data is available, unified content is then generated and delivered.
- **Communication layer:** performs session management and delivers messages between client and server. An interesting aspect is that it can detect network status and decide if a message is stored in a message buffer when no connectivity is available.
- **Content consuming and regeneration layer:** messages are assembled and sent to the content visualisation layer.
- **Content visualisation layer:** this layer is implemented on the client side software. The object-parsing engine parses the multimedia object to a Document Object Model (DOM) structure.

MoCA

MoCA [37], is a middleware that aims for easier developing and deploying of context-aware (see 2.2.3) mobile collaborative applications. The system infrastructure consists of client and server APIs, basic collaborative services and a framework for the implementation of application proxies.

The system assumes the existence of two types of nodes on a static network. Those nodes are servers and proxies. The client processes run on mobile devices. A particular aspect of this framework

is that, proxies act as mediators in all communication between the application servers and the clients, their tasks include: data compression; protocol conversion; encryption; user authentication; context processing; service discovery; handover management and others.

The system architecture is particularly interesting. Application servers and clients are developed using the MoCA APIs that support the creation of collaborative applications in synchronous and asynchronous modes (see 2.2.2). When the application developer wants to perform specific, heavy processing and adaptability operations, it relies on a proxy node to perform them, using the provided functions of the Proxy Framework. In this way, is possible to develop client and server applications while not concerning about portability issues, leaving that for the proxy to handle.

SyD

Prasad et al. [29] presented SyD, a middleware platform with a centralised architecture that aimed to ease mobile collaborative application development by addressing the key heterogeneity issues in mobility.

The key feature of SyD is the SyD Deviceware component. It presents a uniform and persistent object view of the device data and methods. Thus, a given user device is hosted in the centralised server in a uniform world-view to all SyD applications.

SyD's system architecture consists of the following layers:

- **SyD Deviceware layer:** consists of a listener module to register objects and to execute local methods in response to remote invocations and an engine module to invoke methods on remote objects.Thus mediating remote executions between the devices objects. The SyD Application Object Server implements this layer, and its is situated in the system's centralised server.
- **SyD Groupware layer:** a logically coherent set of services, APIs, and objects that eases the execution of collaborative applications. It is implemented on the centralised server's SyD Kernel.
- **SyD application layer:** developed applications rely on the above layers and are independent of the device characteristics, data representations, and network protocols. Applications interact with other devices objects by instantiating the correspondent SyD Deviceware objects.

Vimoware

Vimoware [43], is a toolkit for the development of collaborative applications in mobile ad-hoc networks, in the same way as YCab (see 2.3.3).

However, it distinguishes itself from the previous examples in a way that it relies on a Web Services based model in the deployment and use of its services. Thus, exploiting the interoperability, flexibility,and re-usability of web services based software. This also eliminates the effort of developing specific communication protocols and models.

Some of the components included in Vimoware's system architecture are:

- **Lightweight Web services Middleware:** SOAP-based Web services are hosted and advertised. The kSOAP2 engine is used to access SOAP based web services. Communications are via HTTP or via direct TCP socket communication.
- **Device and location sensors:** device sensors (CPU, network, memory, and device profile) and location sensors are wrapped in Web services. Client applications can then send requests to obtain context information. Thus, providing awareness (see 2.2.3).
- **User and team management:** this component manages user profiles and a list of teams the user belongs to. The profile includes basic information, such as name, ID and skills, used to associate users with services deployed in the device.

2.4 Development in Mobile Platforms

A serious hurdle in mobile development is the lack of an unified development platform, that largely restricts developers in the adoption of a *write once, run everywhere* philosophy. The variety of mobile operating systems, each with different development environments and features and all with considerable adoption in the industry, is the main contributing factor for this situation.

Therefore, we need to evaluate the available features of current mobile operating systems and development frameworks in ways that best address the functional requirements of our system, the issues in mobility (see 2.3.1) and middleware objectives (see 2.3.2).

We would like to take a special emphasis on the issue of **system portability** because we feel that this has not been a primary concern of the studied middleware examples (see 2.3.3). According to our objective of deploying collaborative applications to the greatest number of devices, this is an issue of extreme importance.

We will start by looking into the features that are natively supported by currently available mobile operating systems that can be useful to in accomplishing our application requirements, in section 2.4.1. Next, in section 2.4.2 we analyse web service technologies for use in mobile applications, in order to build an application without the need to specify specific network protocols in the communication with external data sources. Last but not least, we end this section with an analysis about Mobile Web development, which is an attractive alternative in order to accomplish our portability requirements, in section 2.4.3.

2.4.1 Mobile Operating Systems

Android

Android[1] is an open-source mobile operating system developed by the Open Handset Alliance[2] - a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. The Android's execution runtime consists of a set of core libraries that provide most of the functionality available in the core libraries of the Java programming language and the Dalvik virtual machine, running on top of a customised linux kernel.

Development is Java based, but, Dalvik[3] is not a Java virtual machine. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is, optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by a special tool included in the Android SDK.

An application developer makes use of the provided Android SDK that includes a Java based application framework that interacts with built-in C/C++ libraries including: media libraries; SQLite[4] libraries; 3D libraries; a Web engine based on WebKit[5], among others.

Android's strengths lie in:

- **Native SQLite support:** the Android API contains native support for creating and using SQLite databases. This is extremely useful for local data persistence during operations in disconnected mode.

[1] Android Website - http://www.android.com/
[2] Open Handset Alliance - http://www.openhandsetalliance.com/
[3] Dalvik Virtual Machine - http://www.dalvikvm.com/
[4] SQLite - http://www.sqlite.org/
[5] WebKit - http://webkit.org/

- **Media framework:** offers built-in encoding/decoding for a variety of common media types; plays audio and video from several types of data sources, including local media files or from data streams arriving over a network connection.
- **Contact synchronisation features:** Account manager and Sync adapters Api provided since Android V2.0, support contact aggregation with multiple accounts and synchronisation features which provide full two-way contact sync with multiple backend services and the device's address book.
- **WebKit engine:** The WebKit version provided with Android since version 1.5 has HTML/CSS/Javascript compatibility comparable with the best desktop web browsers, scoring a 93/100 score in the Acid3[6] tests. Version 2.0 added HTML 5 support. Also, the API provides the WebView class which permits the creation of embedded browser windows within an application's interface.

iPhone OS

iPhone OS[7] is the operating system of the iPhone device by Apple.

An application developer makes use of the iPhone SDK to develop iPhone applications. Development is based on Apple's Objective-C Language, which can be considered a main drawback, severely hindering the portability of the developed applications to other systems, due to the fact, that this is not a commonly used language in the vast majority of applications besides Apple's own operating systems. However, the vast world-wide adoption of this device makes us consider it in our study.

The SDK provides an extensive set of APIs that interact with the four iPhone system architecture layers:

- **Cocoa Touch layer:** includes the Apple Push Notification Service useful for awareness functions; the Address Book UI Framework that permits direct access to the user's contacts and consequent integration in the developed application; Map Kit Framework that gives the developer the possibility to use location services on the developed applications and the UIKit Framework that contains Objective-C programming interfaces that provide the key infrastructure for implementing graphical, event-driven applications and provides access to the device's capabilities.
- **Media layer:** deserving mention are the AV Foundation and Media Player frameworks that support playback and recording functions of audio and video in a vast quantity of codecs. There is no media network stream manipulation though, as offered by the Android system (see the section above).
- **Core Services layer:** includes the Address Book framework which provides programmatic access to the contacts stored on a user's device; the SQLite library and XML support.
- **Core OS layer:** includes the CFNetwork framework which is a set of high-performance, C-based interfaces that provide object-oriented abstractions for working with network protocols (BSD sockets, encrypted connections using SSL or TLS, HTTP, authenticating HTTP, and HTTPS servers).
- **WebKit based browser:** Apple's own Mobile Safari browser, present in the iPhone OS ranks as the best browser implementation in mobile devices scoring a 100/100 score on the Acid3 tests.

[6] Acid3 tests - http://www.webstandards.org/action/acid3/
[7] iPhone OS developer center - http://developer.apple.com/iphone/

Windows Mobile

Development on the Windows Mobile[8] system is based on the .NET Compact Framework in C++ or C# programming languages.

Although Windows Mobile is not as powerful as Android and Iphone OS in relation to contact synchronisation, multimedia and Web integration capabilities, its main advantages lie in:

- **Advanced telephony services:** Telephony API provides functions for easy GSM call establishment, SMS messages.
- **Compatibility with ASP.NET Web Services[9]:** permits direct access to and easy deployment of SOAP based Web services.
- **Compatibility with SQL Server Compact[10]:** easy deployment of embedded databases and automatic synchronisation features with centralized SQL Servers guarantees data consistency.

Symbian and BlackBerry

Development on the Symbian[11] platform can be C/C++ based or like BlackBerry[12], Java ME based.

Development for these systems benefits from the Java ME compatibility of both. Considering the extremely high market penetration of Symbian and Java ME devices, we can get an high level of portability by developing on the Java ME platform.

We thereby describe some worth mentioning features we can get by investing in Java based development on these two systems:

- **SQLite Support (only on Blackberry):** the BlackBerry system provides native SQLite functions.
- **Multimedia capabilities:** compatibility with the JSR-135 Mobile Media API[13] specification, supports recording and manipulation of media network streams.
- **Event handling:** MIDP compatibility supports event handling functions and notification services.
- **Telephony:** GSM calls are supported through MIDP2.0 plattformRequest() method.
- **SOAP based Web services:** soap Web services are supported due to kSOAP2 and JSR-172 compatibility (refer to 2.4.2).
- **Web Integration (only on BlackBerry):** the BlackBerry Java Development Environment, provides the BrowserField which is an equivalent of Android's WebView.

2.4.2 Web Services in Mobile Devices

As it was seen in Vimoware's implementation (refer to 2.3.3) a web-services based approach in service deployment was used to avoid the creation of customised communication protocols and explore the better interoperability, flexibility and re-usability of web services based software. For us, the web services approach in the deployment of some system components is of particular interest to achieve the high level of portability we desire.

[8] Windows Mobile development - http://msdn.microsoft.com/pt-pt/library/bb158483(en-us).aspx
[9] **XML Web Services Using ASP.NET - http://msdn.microsoft.com/en-us/library/ba0z6a33.aspx**
[10] **SQL Server Compact - http://www.microsoft.com/Sqlserver/2005/en/us/compact.aspx**
[11] Symbian OS - http://www.symbian.org/
[12] BlackBerry Development - http://na.blackberry.com/eng/developers/
[13] JSR-135 specification - http://jcp.org/en/jsr/detail?id=135

SOAP Web Services

As it happens with desktop web applications, we can use the SOAP[14] message protocol to support the communication between local mobile terminals with external web services over HTTP. However the significant overhead introduced might be too much for the most constrained mobile devices and it is also not desirable in a developer's point of view to implement the code necessary for SOAP message creation and parsing. The alternative is to use lightweight SOAP client libraries for this effect.

One of these, is kSOAP2:[15] a third-party Java client library designed for constrained Java environments, such as, applets or Java ME[16] applications. It is recommended for devices that are not compatible with the JSR-172[17] specification, a J2ME client library for SOAP web services. Nevertheless, kSOAP2 requires a Java ME compatible device, given that, the library itself uses Java ME's own HTTP communication framework.

JSON

A good alternative to SOAP is JavaScript Object Notation[18] (JSON). It is a lightweight data-interchange format, ideal for mobile devices to process and generate. It is based on a subset of Javacript and is completely independent from the application's programming language.

There is a vast community support for this notation, spanning from a vast quantity of third-party open source libraries for almost any programming language available. Thus, making it (like SOAP) a viable option to guarantee system portability.

Some may argue that there is a disadvantage in implementing web services in JSON, due to the difficult management, security and discovery of these type of services given that no contract is established, as it happens with SOAP and the WSDL[19] format. But, simplicity and the much reduced overhead introduced in communication makes it a very popular option in the Web community when used in conjunction with REST, which is described in the next section.

REST

Another alternative format is Representational State Transfer[20] (REST). It is an architectural style where each of the system's available resources are represented in an unique URL. The purpose is to use standard HTTP functions (Get, Post, Put, Delete), to access and modify those resources.

It suffers from the same disadvantages as JSON but it is the most lightweight and portable alternative.

2.4.3 Mobile Web based Development

A recent trend in mobile development is to explore the inherent high levels of portability of Web based development.

The mobile web development community strives for the possibility to apply a *write once, run everywhere* philosophy in the development of their mobile applications. This need has motivated vendors

[14] SOAP specification - http://www.w3.org/TR/soap12-part1/
[15] kSOAP2 Website - http://ksoap2.sourceforge.net/
[16] Java ME Apis - http://java.sun.com/javame/reference/apis.jsp
[17] JSR-172 specification - http://jcp.org/en/jsr/detail?id=172
[18] JSON Website - http://json.org/
[19] WSDL specification - http://www.w3.org/TR/wsdl
[20] Building Web services the REST way - http://www.xfront.com/REST-Web-Services.html

and developers to seek alternatives like Mobile widgets (see section 2.4.3.1) or unified development frameworks (refer to 2.4.3.3).

This is of special interest to us, because alternatives like these give us an unparallelled level of portability that was simply not achievable when focusing the development on a single mobile operating system.

2.4.3.1 Introduction to Widgets

With the advent of the Web 2.0 a new kind of application has gained significant popularity. Generally known as widgets, these applications have the particularity of being lightweight web applications designed for single specific functions and quick access. The World Wide Web Consortium (W3C)[21], in an effort to standardise widget development, defines widgets as:

"interactive single purpose applications for displaying and/or updating local data or data on the Web, packaged in a way to allow a single download and installation on a user's machine or mobile device."

The most common examples of widgets are clocks, weather forecasters, news aggregators and photo albums, and are developed by using standard web technologies such as HTML, CSS, JavaScript and XML. The application code is usually downloaded only once (if the user downloads the application from the web), and the application code is then deployed locally on the device, with subsequent communication with external data sources being accomplished with an AJAX development approach. The code must be interpreted and executed by a special execution runtime, which is generally known as *widget engine*.

Widgets can be divided in three groups [19]:

- **Desktop widgets:** generally seen in desktop operating systems of major vendors, like Microsoft's Windows Vista Sidebar or Apple's dashboard in MacOS X. These products are the widget engines that give users the possibility of keeping useful gadgets (as they are called) on their desktop. An important aspect is that these widgets are executed locally and they do not need to necessarily serve web content, as many of them only provide local data. As examples: the local clock, a simple calculator or a CPU monitor.
- **Mobile widgets:** in a similar way as desktop widgets, these widgets run locally on the mobile device in the execution runtime of a specific widget engine which in turn, must be integrated with the host mobile operating system. Differences with desktop widgets lie more in development issues than how widgets are deployed or executed. Of course a developer accustomed with desktop widget development must take into account the common mobile device limitations and development challenges that we have seen before in this document.
- **Web widgets:** are commonly reusable components of a web site which can be embedded in other web pages. The difference in relation with desktop and mobile widgets is that they are not packaged and need not to be downloaded and installed locally on a client.

Only desktop and mobile widgets satisfy the W3C widget stack architecture recommendation [4, 19, 7], as seen in Fig.2.1.

The recommendation states that a widget must be composed by one or more HTML, CSS and ECMAScript files. Image and sound files can also be included. Additionally, a widget must have a manifest file, which contains metadata about the widget configuration. Common metadata fields are: widget

[21] W3C - Widgets 1.0 Requirements - http://www.w3.org/TR/2009/WD-widgets-reqs-20090430

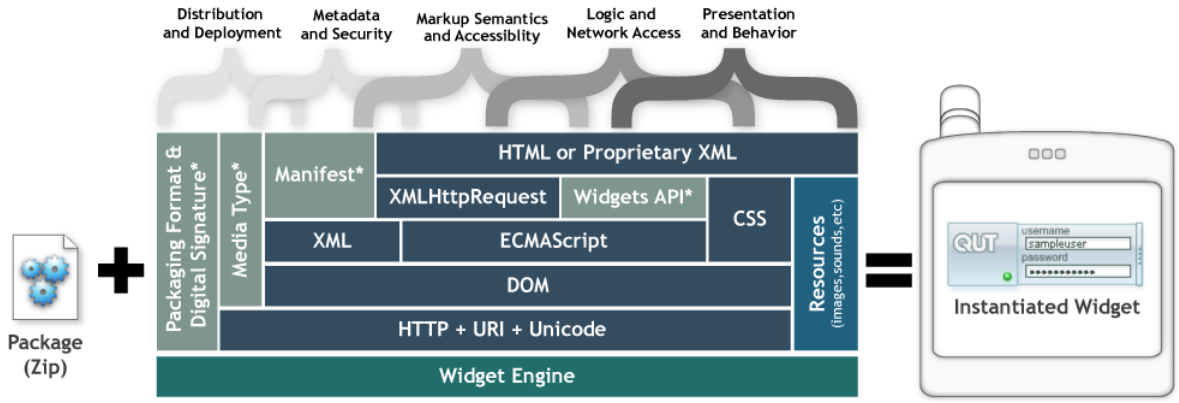


Figure 2.1: W3C's Recommendation Widget Technology Stack

name, version and the obligatory main HTML file, as all widgets must have only one main HTML file which is the entry point of the application. The developer must be aware of the widget engine capabilities, as some do not support multiple HTML files, thus, navigation between different web pages might not be possible due to issues related with browser history support or memory constraints. Widget developers are then encouraged to develop applications around a single main HTML file. All of the widget files are packaged into a single file using a standard packaging format, like ZIP. Depending on the widget engine, the extension of the resulting package file must be renamed in order to be recognised.

The recommendation also gives room for vendors to provide specific functions which need to be mapped in specific APIs, accessible to the web coded application.

Although W3C's effort is to standardise widget development, the recommendation still gives room for many incompatibilities, making the process of porting a widget developed in a specific widget engine to another a not instantaneous and issue free process. Main incompatibility issues are [19]:

- **Different manifest documents:** the recommendation specifies that widgets need an obligatory manifest document, but does not standardises manifest data files or file names. As a result, each widget engine vendor has its own type of manifest documents.
- **Packaging:** again, the packaging format is not standardised resulting in packages with different file extensions and internal package structure.
- **Platform specific functions:** as has been told before, widget engines support different platform features and the API to access these features is not standardised. This issue is commonly the one which causes more impact for the end developer.

In hopes of increasing application portability, the mobile application development community has been looking into widgets as an easy way for the rapid deployment of applications, by reusing code from existing desktop web applications and maximise their user base by deploying applications for the maximum number of platforms.

The urge for mobile widgets haunts the community as the paradoxical increase of fragmentation in mobile operating systems and the need for simple and reusable applications is more and more evident. As such, mobile widgets come as an option for us too and must be considered in our work's scope. In the next section, we present some of the currently available widget engines for the development and management of widgets in today's mobile devices and evaluate if they are viable choices in the development of our solution and accomplishment of our objectives.

2.4.3.2 Mobile Widget Engines

In order to accomplish our portability objective, we are particularly interested in what mobile widget engines have to offer. Code reuse, rapid and easy application deployment are our main interests, but we are also interested in the capabilities offered by the specific functions that the different widget engines might support. For example, it could be interesting to use a widget engine that could provide the option to load the mobile device's contacts through a provided JavaScript API that we could use in web application code developed by us. As such we did a little survey of currently available mobile widget engines and present it below.

BONDI

BONDI[22] is an initiative mainly intended to address the problem of non-interoperability between multiple mobile platforms. It was founded in mid 2008 by the OPEN Mobile Terminal Platform[23] (OMTP), which is a forum of international mobile operators and device manufacturers. By pushing mobile Web application development, BONDI hopes to make the development process easier for new mobile applications.

BONDI proposes a set of JavaScript interfaces for typical telecommunication and device specific services, that can be used in a normal website, but mainly, in widgets. The idea is to expose native functionality (otherwise inaccessible) to applications developed by Web technologies in mobile devices. Version 1.1 of BONDI interfaces have been approved since February 2010.[24]. BONDI's architecture is represented in Fig.2.2

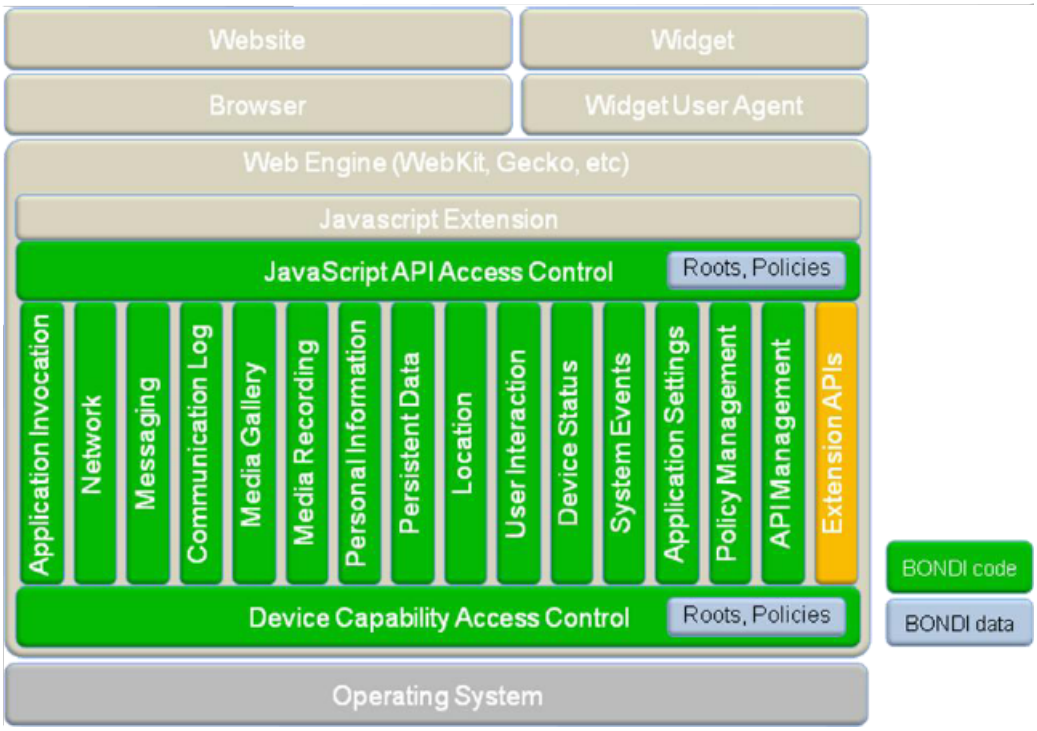


Figure 2.2: BONDI Architecture Overview

[22]BONDI Web site - http://bondi.omtp.org/default.aspx
[23]OPEN Mobile TErminal Platform Web site - http://omtp.org/
[24]BONDI 1.1 Approved Release - http://bondi.omtp.org/1.1/

BONDI's JavaScript API is separated into 16 groups such as Messaging, Media Gallery and Personal Information (which gives information about Calendar and Contacts). Of course, having sole JavaScript sources simply does not work. Thus, BONDI started to integrate their JavaScript extensions into the browser engine of Windows Mobile to make them accessible to end-user applications, and, to the Operating System layer in order for the extensions to access native functionality. As a downside, in order to execute device specific BONDI commands the JavaScript interpreter of the host operating system must be rewritten. A specific operating system vendor needs to indulge in changes that may be undesirable if it wants to support the architecture.

BONDI also follows the W3C widgets recommendation, as BONDI widgets are also based on a structure of HTML, CSS, JavaScript and image files, a main html file and a manifest xml document "config.xml".

As of today, BONDI has gained the support of more vendors and is also now compatible with Samsung Widget SDK 1.2 and LG SDK 2.01 for Mobile Widgets[25], and it provides it's own BONDI SDK.[26]

Nokia Web Runtime

Nokia was the first company to introduce support for widgets as they are defined in the W3C recommendation since its S60 3rd edition Feature Pack 2 platform [27] with Nokia Web Runtime 1.0 (WRT).

Like BONDI, WRT also introduced a set JavaScript APIs to provide access to device specific functionality. In WRT it came with version 1.1 of the runtime through support for the Symbian Platform Services. Latest versions like 7.1 and 7.2 added support for widgets to be executed on the Symbian S60 5th edition home screen and Flash Lite 3.0 while retaining backward compatibility with widgets developed in prior versions.

WRT is an execution environment that is part of the Nokia S60 Webkit based browser, thus being perfectly integrated with the operating system. As mentioned above, the application developer can use the provided APIs and combine information from the Internet with data stored on the device. It is possible, for example, to utilise location information from the GPS hardware, or the contacts locally stored on the device.

WRT provides the following API sets[28]:

- **Platform Services 1.0 JavaScript API:** provides access to a wide range of device and user information, such as Calendar and Contacts application data. This API set is the most available of the three, being available for selected Symbian OS and S60 3.2 and all Symbian OS 5.0 devices from Nokia.
- **Nokia Platform Services 2.0 JavaScript API:** similar to the first one, this one adds camera support and a simplified access to device information.
- **APIBridge:** complements the Platform Services APIs with features for file activities, image manipulation, and multimedia capture. It also provides an alternative mechanism for using multimedia, logging, and location services. One differentiating factor is that this set is available for use in Java applications and Adobe Flash Lite content.

Nokia does not offer a SDK for the development of WRT widgets, because the developer only needs text edition and packaging tools to make a widget and deploy it on the device. Obviously a developer

[25]Developing BONDI widgets - http://bondi.omtp.org/usebondi/Webpages/devtools.aspx
[26]BONDI SDK 1.0 - http://bondisdk.limofoundation.org/
[27]Nokia S60 Widgets - http://sw.nokia.com/id/e67c057a-7549-4edb-b9f1-193ab27cce71/S60 Web Run Time and Widgets.pdf
[28]Nokia WRT Overview - http://www.forum.nokia.com/Develop/Web/Webruntime.xhtml

needs to meet the requirements for the widget packaging defined by Nokia, but if all goes well, a widget is executed as any other S60 application locally on the device in all Symbian S60 devices since S60 3.2, making the development and deployment of mobile applications easy and with little to no cost for accustomed web developers.

BlackBerry Widget SDK

RIM also introduced a widget engine on version 5.0 of their BlackBerry OS. Very similar to Nokia WRT, this widget engine also supports the execution of packaged widgets without usage of special development tools. Nevertheless, and as opposed to Nokia WRT, RIM provides a SDK that eases the process of widget packaging for the end developer and provides some special development tools. The SDK's currently available features are[29]:

- **Widget packager:** allows web developers to package their web assets into BlackBerry Widgets automatically, so the user does not need to create the packaged widget manually.
- **Gears API[30] Support:** gives web applications storage, location, multi-threading and more features.
- **Support for JavaScript Extensions:** the end developer has the possibility to add their own or third party JavaScript extensions to BlackBerry widgets and leverage them in the Blackberry core API functionality by the creation of custom links.
- **Security Features:** includes digital signing of widgets, usage of wildcards for the definition of domain white or black lists, and all security features of native BlackBerry applications concerning the packaging and distribution of applications, via websites or BlackBerry App World.

2.4.3.3 Mobile Web Development Frameworks

In the previous section we talked about widget engines and how they give developers the possibility to easily build and deploy mobile applications using Web technologies to minimise development costs and maximise portability. In this section we present two frameworks which give developers the same possibilities as widget engines but in a slightly different way.

The main difference between these frameworks and widget engines is that as opposed to widgets that need not to be compiled to be executed (they just need to be properly packed and configured). Applications that are built around these frameworks need to be compiled for the target operating systems. These frameworks generally offer a set of components built in the operating system native languages, which provide an interface accessible by JavaScript interfaces that can be used by the end application. In order to execute the developed code and JavaScript APIs, operating system components like Webviews which permit full screen presentation of Web content in an application and bridging of JavaScript functions with native functions are used.

In the next section we will present two of these frameworks.

PhoneGap

PhoneGap[31] is a mobile cross-platform development framework created by Nitobi Software.

PhoneGap provides a set of packages built for each target operating system which are basically composed by source files built for the target programming language ready to be compiled on a specific

[29] BlackBerry Widget SDK Overview - http://na.blackberry.com/eng/developers/browserdev/widgetsdk.jsp
[30] **Google Gears API - http://code.google.com/intl/pt-PT/apis/gears/**
[31] PhoneGap Web page - http://phonegap.com/

SDK. The developer just needs to move his developed web assets to a predefined directory where web content must reside and compile the application. When the user executes the application in his mobile device the PhoneGap developed code instantiates a Webview in full screen where the application code is interpreted. Furthermore, the underlying PhoneGap code instantiates a set of interfaces that access native functionality, which is properly mapped by a JavaScript API that the developer can use in his/her web applications.

In comparison with widget engines, that are built around a specific operating system and provide non-normalised APIs, PhoneGap appears as a more powerful option considering the portability issue. Even that PhoneGap gives a different implementation for each system, the provided JavaScript APIs are normalised, being generally the same for each implementation minus for a very small number of specific functions. The main drawback is that a developer needs to be familiarised with multiple SDKs and know how to setup and compile applications for each one.

As of today, PhoneGap is compatible with almost all smartphone operating systems available. It is compatible with: Android 1.5+, all versions of iPhoneOS, Symbian 5th edition, Windows Mobile 6.5 and Blackberry OS 4.6+, Palm and Maemo.

Another major drawback of this framework is that not all features are supported in the same way in every system, due to some system limitations or by lack of implementation. In Table.2.1 we present a summary of PhoneGap's supported features[32] on each operating system.

	iPhone	Android	Blackberry OS	Symbian	Windows Mobile	Palm	Maemo
Geolocation	✓	✓	✓	✓	×	✓	×
Accelerometer	✓	×	✓	✓	×	✓	✓
Camera	✓	✓	✓	✓	×	×	×
Vibration	✓	✓	✓	✓	×	✓	×
Contacts API	✓	✓	✓	✓	×	×	×
SQLite Functionality	✓	✓	×	×	×	✓	×
File System IO	×	✓	×	×	×	×	×
Gesture/Multitouch	✓	×	×	×	×	×	×
SMS API	×	×	✓	✓	×	✓	×
Telephone API	×	×	✓	×	×	✓	×
Copy/Paste	×	✓	×	×	×	×	×
Sounds (Play)	✓	✓	✓	✓	✓	✓	×
Sounds (Record)	×	✓	×	×	×	×	×
Maps	✓	×	×	×	×	✓	×
Orientation change	✓	×	×	✓	×	✓	×
Network availability	✓	×	✓	✓	×	✓	×
Magnetometer	✓	×	×	×	×	×	×

Table 2.1: PhoneGap Feature Support Map

As the reader may notice by observing the table mentioned above, there is a large discrepancy in the full support of specific functions on target operating systems. Even using PhoneGap, a developer needs to evaluate his application's requirements in terms of feature support and portability and make certain compromises. Imagine that we are developing an application which needs to load the device's contacts and send SMS messages. Well, the contacts part is possible on four systems, but, the SMS part is only possible on two of the four systems. Now it's up to the developer to drop the SMS functionality or have two distinct applications. This sole aspect serves to alert the reader that even with a framework like this, problems still exist concerning application portability.

[32] PhoneGap wiki - Feature Roadmap - http://wiki.phonegap.com/Roadmap

Titanium Mobile

Appcelerator's Titanium Mobile[33] appears in the same way as PhoneGap by augmenting mobile applications portability by development based in (HTML/CSS/Javascript).

Figure 2.3: Titanium Mobile Architecture Overview

As Phonegap, Titanium provides a set of JavaScript APIs that can be used by the end application, properly bridged with code built in the native operating system language. This framework distinguishes itself by offering a complete and unified development environment independent from any target operating system's SDK. When deployment is made, it creates binary files for the corresponding operating system.

Titanium only supports two operating systems, namely iPhone and Android. Concerning the number of supported systems it loses for PhoneGap, nevertheless, Titanium offers a more consolidated API with no major discrepancies in feature support between the two platforms.

Titanium's differentiating features include:

- **JavaScript User Inteface API:** offers an API for the creation of user interfaces that mimic native UI elements of each system, thus applications developed with Titanium tend to look more "native".
- **Local database functions:** the provided JavaScript libraries give local access to embedded databases out of the box.
- **Social Networking and Location Based Services:** provides APIs to easily access social networking services like Facebook, Yahoo, Twitter and Geo-location services.
- **Cloud services:** includes also a full SOAP client for making SOAP API requests and APIs that support RESTful web services.

2.5 Chapter Summary

In this chapter, we did an extensive survey in topics and available technologies that we determined as essential for the conception and development of our own solution.

As the objective of this work is to develop a common framework that comprises collaborative functionality and support to develop collaborative applications, we started by, to study the foundations of Collaboration and how it was first supported by computational tools, which is the concept of *Computer Supported Cooperative Work* (CSCW), and how it later evolved to the concept of *Groupware* (sections 2.1 and 2.2).

[33] Titanium Mobile architecture - http://www.appcelerator.com/appcelerator-platform/titanium-architecture/

Research in Groupware has been overlapped with many disciplines. It comprises several key perspectives (section 2.2.1) that go from the traditional problems of distributed systems, such as, access control, concurrency control and data consistency, to network protocols; human-computer interaction; social theory and artificial intelligence.

We focused on the distributed systems and communication topics and made an overview in Groupware Systems (section 2.2.2), by studying them according to a *time-space taxonomy* that distinguishes systems according to the interactions they support in space and time. This permitted us to focus our interests in distributed interactions in the synchronous (real-time document edition, videoconference, for example), and asynchronous (group management, mail systems) modes. Concerning the communications side, we distinguished systems by their network architectural designs, that commonly follow, a centralised approach in the *client-server* model, with the advantages of data consistency guarantee and the disadvantages of centralised communications; a distributed approach, in a *peer-to-peer* model which offers better performance but brings the challenges of maintaining data consistency.

Albeit differences in the architectural designs and time-space interactions, development of groupware systems brings several challenges to the table (section 2.2.3). These challenges go from providing Access Control to collaborative resources according to the user that is accessing them; Awereness to provide understanding of the activities of others to a given user; Coordination to avoid conflicting access to shared resources, which if conflict cannot be avoided we need Concurrency Control; collaborative Session Management and Tailorability.

Our objective is to deliver a Groupware solution to mobile devices. Therefore, our survey focused mainly in the development of these solutions in mobile devices (section 2.3). Mobile Groupware brings new and difficult challenges (section 2.3.1) that include: service discovery and mobility management in networks; frequent disconnections and provision of a disconnected mode of operation; user interfaces problems due to different sized and small screens and the diversity of devices and development platforms.

To mitigate these issues, middleware solutions are commonly deployed. In section 2.3.2 we identified a set of requirements that are accomplished by mobile middleware. These requirements comprise the provision of an execution support layer; persistence support; resource adaptability; extensibility; portability and fault tolerance. In order to acknowledge how these requirements are accomplished today, we studied several middleware solutions, that are presented in section 2.3.3.

It became evident that the portability issue, which is one of our main requirements, was not properly addressed in the studied middleware examples, as they were all deployed and developed in specific mobile platforms with the portability to other platforms not being validated. In order to accomplish our requirement we had to explore the currently available mobile operating systems (section 2.4.1) and study their relevant features and how they correlate in order to produce a portable solution.

A recent trend in mobile development in order to maximise portability is to develop mobile applications in the form of mobile widgets (section 2.4.3.1) developed in HTML/JavaScript/CSS that can be executed across different platforms that provide a bridging component with the Web code and the native operating system APIs, as is the case with the studied Widget Engines (section 2.4.3.2) and Mobile Web Development Frameworks (section 2.4.3.3). We have followed this alternative and in the next chapter we will present our architectural design where many of the studied topics and technologies were applied.

3 Solution's Architecture

"A self does not amount to much, but no self is an island; each exists in a fabric of relations that is now more complex and mobile than ever before."
– Jean-Francois Lyotard - Philosopher

The objective of this chapter is to present the architectural design model of our solution. In the related work chapter (see chapter 2) we did an extensive survey of the concepts and challenges about groupware, mobile environments and development of mobile applications. This survey, and its analysis, serves as the basis of our design model where many of the studied concepts and tools have been applied.

As the reader may recall from this document's introduction (see chapter 1), our solution aims to be a portable and reusable middleware for the development of groupware solutions for mobile devices, while complying with existing infrastructure. The mentioned infrastructure is PT Inovação's own unified collaboration platform: Plataforma Unificada de Colaboração (PUC).

As a collaboration platform, PUC already accomplishes access control, concurrency control and session management requirements (refer to section 2.2.2), and offers a persistent information base. It also provides a core of collaborative functionality through a set of application interfaces and resource management controllers. By integrating our solution with PUC, we gain immediate support for the requirements mentioned earlier. Our proposed architectural design is then focused on offering:

- **Code reuse and portability:** benefit from usage of open Web standards and cross platform frameworks, concerning both communication protocols and application development.
- **Adaptability:** a system component that delivers resources according to the capabilities of the client device.
- **Data synchronisation:** support for synchronisation of relevant information stored in the mobile device with the server side platform.
- **Offline work**: support for local storage of relevant data on the mobile device.
- **Effective user awareness**: minimum application response time to events, capitalising on usage of communication technologies based in *publish/subscribe* models with *server-push*[34] delivery of messages instead of standard polling mechanisms.
- **Extensibility:** object oriented design in both server and client side architectures that is easily extended with new components.

The architectural choices and development of PUC is not in this work's scope. Nevertheless, we present a brief overview of PUC's architecture in section 3.1, in order for the reader to better understand some of the choices made in our design, which is tightly coupled with PUC's model.

Next, in section 3.2 we present our solution's architecture, with a single detailed analysis for each of our proposed system entities, the PUC Gateway (refer to 3.2.2) and the Mobile Terminal (refer to 3.2.3).

[34]Comet Programming - http://www.webreference.com/programming/javascript/rg28/

3.1 PUC Architecture Overview

PUC is born from the need of converging different collaboration services that were once separated and part of different and non interoperable platforms. PUC aims to solve this issue by providing an abstract and unified framework of *collaboration service enablers* accessible to third party applications.

The framework presents itself as a layered architecture (see Fig.3.1):

- **Application layer:** in-house or third-party applications that want to use PUC's service enablers through the framework's public API.
- **Management layer:** provides the core business logic of the available collaboration services (conversation and session Management, management of collaborative resources, etc.).
- **Persistence layer:** contains the necessary code for database management. It runs as a service accessed by the Manager entities.
- **Resource layer**: holds the resource connectors which are responsible for establishing a network connection with the resource servers while abstracting their own network protocols from the resource manager's business logic.

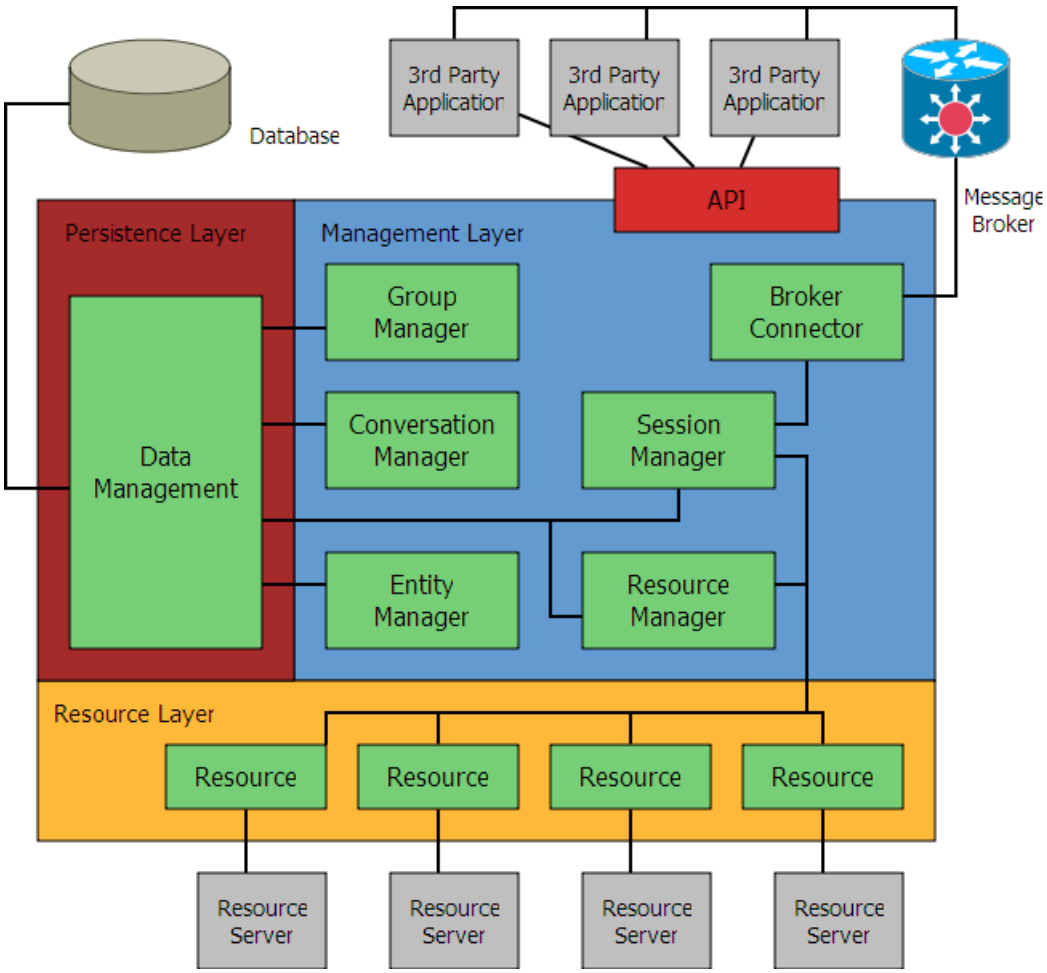


Figure 3.1: PUC Architecture Overview

There is also the *Broker Connector* component that connects to a message delivery service which publicises the received events launched by the platform in order to be consumed by client applications.

With an unified public API and extensible core and resource layers, new resource servers with new network protocols can be added with minimal impact for the platform developer and no cost for the client application developer.

The PUC Gateway which is proposed in our solution acts as an in-house client of PUC. For the next sections we will describe how this integration is accomplished.

3.2 Mobi-Collab Architecture Overview

We begin by stating that in order to maximise re-usability and portability in our solution, all of our architectural design choices follow an approach based in open standards and a Web based development environment on the client side. Regarding the later, this option was evaluated versus a native development approach and decided as the best course of action, considering that the inaccessible native functionality does not interfere with the accomplishment of our functional requirements.

Mobi-Collab's architecture takes a semi-centralised approach with each mobile terminal having its own information base and local interfaces.

We have adopted a layered design, which will be described in the following paragraphs. Note that some components are already developed, with some being usable third-party implementations that support our own components.

We start by presenting an architecture overview describing the different parties involved in the communication process and the residing system components (Fig.3.2).

System Entities

Five different entities are involved in the communication process:

- **PUC Platform (in-house developed):** Described in section 3.1, this is the set of collaboration service enablers, exposed by a Public API that gives support for our groupware functionality and it can be hosted in an application server of choice.
- **Mobile Terminal:** this is the mobile device itself. All application code is locally deployed in the device, as are, our middleware components, which means that, a download process does not occur, when application components are loaded. All are independent of the provided operating system's execution runtime.
- **PUC Gateway:** acts as a proxy server for the mobile terminal, mediating access to the provided core groupware functionality exposed by the PUC's application server. The objective is to allow support of multiple client-server communication protocols for the terminal side with the collaboration core, while preserving the implementation of the latter. Another purpose is to accomplish our *adaptability* requirement, by providing resources according the client device's capabilities.
- **Bayeux Event Broker (Third Party):** listens for events launched by PUC and forwards them to the mobile terminal in the form of asynchronous messages over HTTP. Clients can listen to events in specific channels which they subscribe to, or publish their own events to a channel they wish, which are then processed by the event broker and forwarded to the platform.
- **Resource Engine Servers (Third Party):** these are the actual servers which host resources that can be accessed by our groupware applications. Examples could be audio and video streaming servers or file repositories.

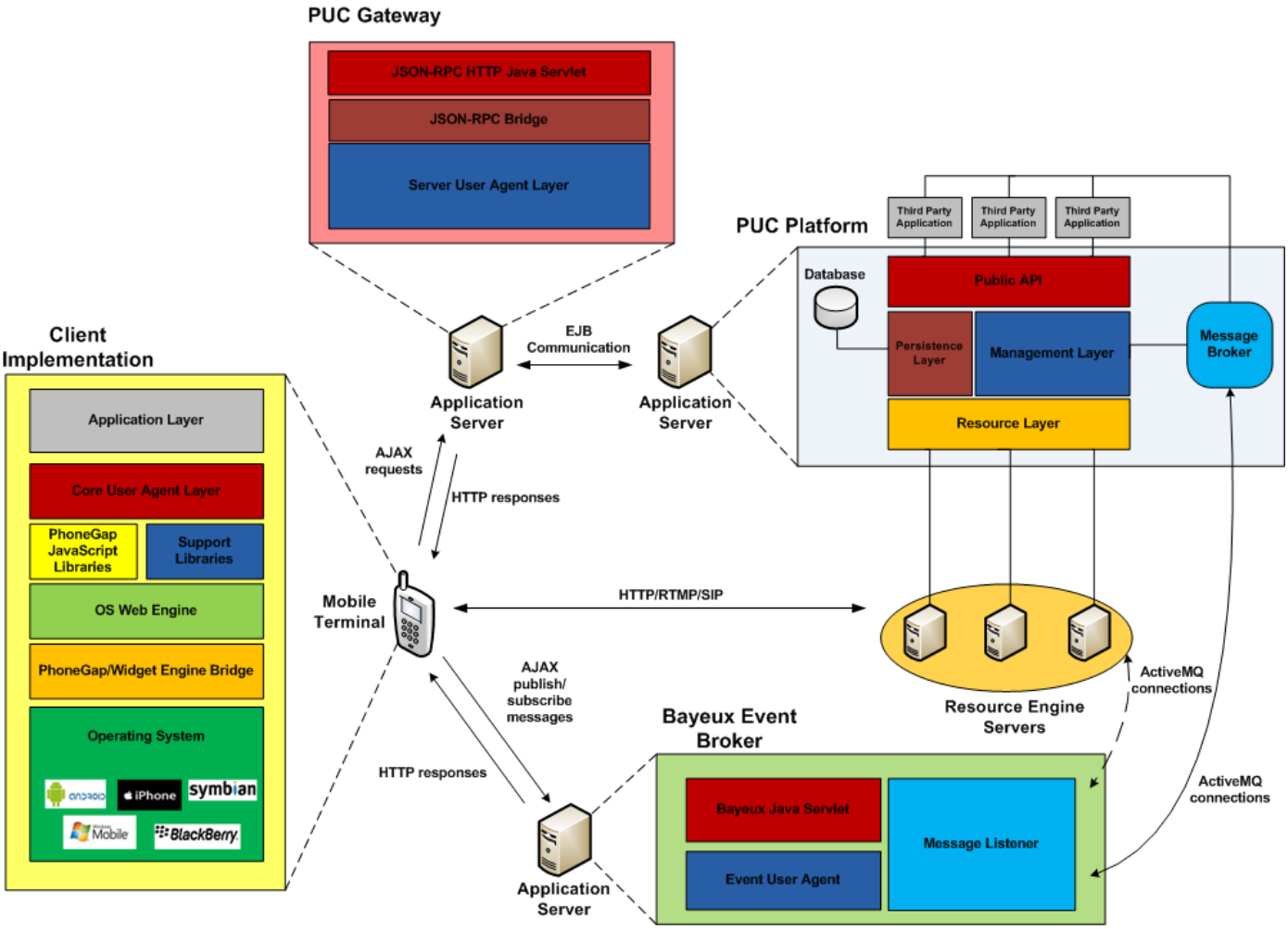


Figure 3.2: Mobi-Collab Architecture Overview

3.2.1 Communication

Considering our objectives, it is crucial that our design does not depend on custom designed network protocols which could hinder the re-usability and portability of our solution (if specific platforms fail to be compatible with our protocol).

Albeit the remarkable architectural differences of multiple platforms, one thing is certain, browser engine implementations on today's devices (refer to 2.4.1), are becoming as capable as their Desktop counterparts. As such, we decided to design a communication process based in protocols which are widely used today on the Web.

Communication with PUC Gateway Communication of the mobile terminal with the PUC Gateway is made through AJAX HTTP Requests invoked by the in-device deployed Javascript, through the XMLHttpRequest Javascript API. The HTTP Servlets on the application gateway side process these requests and generate specific HTTP responses.

HTTP POST requests and responses in the communication between the Mobile Terminal and PUC Gateway follow the JSON-RPC[35] over HTTP protocol. It is basically, a lightweight remote procedure

[35]JSON-RPC Specification - http://json-rpc.org/wiki/specification

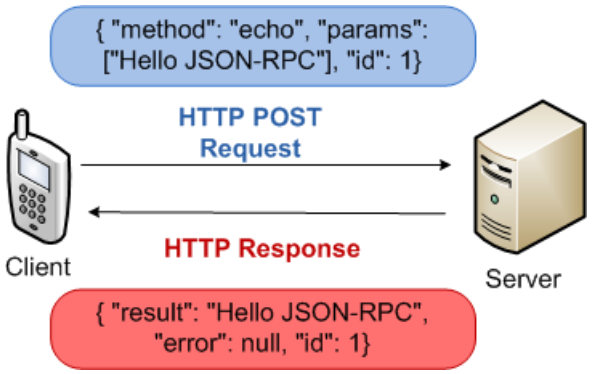


Figure 3.3: JSON-RPC Communication Example

call protocol that uses the JSON message format, which we presented in our related work chapter (see 2.4.2).

Returned JSON strings in HTTP responses are unmarshalled on the client-side JavaScript code to create data objects of the returned information.

Communication with Event Broker Communication with Event Broker follows the Bayeux message transportation protocol.[36] In order to increase user awareness, minimum user response to events is needed, thus, to minimise latency in server-client message delivery, we opted to use Bayeux's *long-polling* transport variant. In a *standard polling* scheme, servers send and terminate responses to requests immediately, even when there are no events to deliver, and worst-case latency is the polling delay between each client request. In our chosen *long-polling* variant, the server implementation attempts to hold open each request until there are events to deliver; the goal is to always have a pending request available to use for delivering events as they occur, thereby minimising the latency in message delivery.

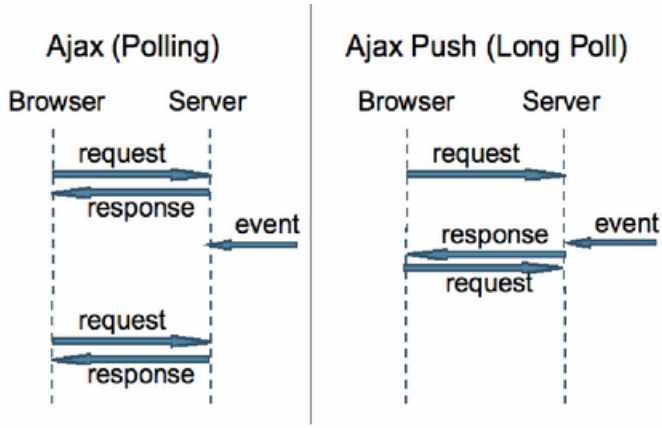


Figure 3.4: Standard Polling versus Long Polling Transportation

The downside is the need for a proper server implementation that handles the high number of open connections when the number of connected users raises, which is not needed in a standard poling mechanism where the implementation of looped requests remains only on the client side. Increased server load and resource starvation are commonly addressed by using the reconnect and interval advice fields to throttle clients, which only in the worst-case degenerates to a standard polling behaviour.

[36] Bayeux Protocol - http://svn.cometd.com/trunk/bayeux/bayeux.html

Gateway Communication with PUC To finalise, communication between the PUC gateway and PUC's Public API is accomplished through Enterprise Java Beans[37] remote method invocations to exposed remote interfaces.

3.2.2 PUC Gateway Architecture

PUC Gateway has a 3-tier architecture (Fig.3.5):

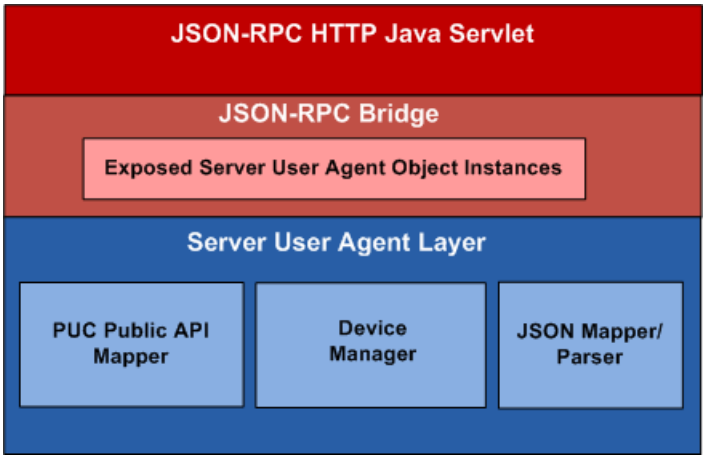


Figure 3.5: View of PUC Gateway's Architecture

- **JSON-RPC HTTP Java Servlet:** responsible for handling all remote invocations made by the mobile terminal via JSON-RPC.
- **JSON-RPC Bridge:** establishes an interface between the HTTP Java Servlet and the application server's hosted Java objects. Any Java Object can be registered on the bridge, rendering the exposure of Java Objects to Web browser clients possible.
- **Server User Agent Layer:** composed by *PUC API Mapper* objects, which are a direct mapping of PUC's own Public API with it's data model adapted to the constraints of mobile devices; a *device manager* component that loads predefined device profiles and delivers resources according to the client supported features; and a *JSON Mapper/Parser* that maps Java objects to JSON strings for the client to handle, and parses JSON strings to Java objects to be manipulated by the platform.

PUC Gateway also has its own data model, designed with the objective of providing a simpler data model for resource constrained devices, and it will be described in the next section (see 3.2.2.1). For a better understanding of how the PUC Gateway process really works, we also present a detailed description of the PUC Gateway's system design which explains the role of each system component and how each component is integrated with inner and external components, such as the PUC Platform itself (see section 3.2.2.2).

3.2.2.1 User Agent Data Model

PUC's data model is heavy on data structure complexity and recursion, thus the only way to use the same data model as the PUC Platform in our mobile client, would be by serialising all its Java object information, generating JSON strings with an intolerable size for restricted devices.

[37] Enterprise Java Beans - http://java.sun.com/products/ejb/

Therefore we decided to design PUC Gateway's, very own data model. We call it by *User Agent Data Model* or *UAData*.

UAData objects are basically a subset of PUC's own data model that directly maps obligatory PUC data objects for the client application, without complex data types and circular references in order to produce JSON string conversions as clean and small as possible. Additionally, it provides a set of objects not available in PUC which are related with device management and device-specific loading of collaborative resources.

Summarising, UAData objects (Fig.3.6) are divided in four main groups:

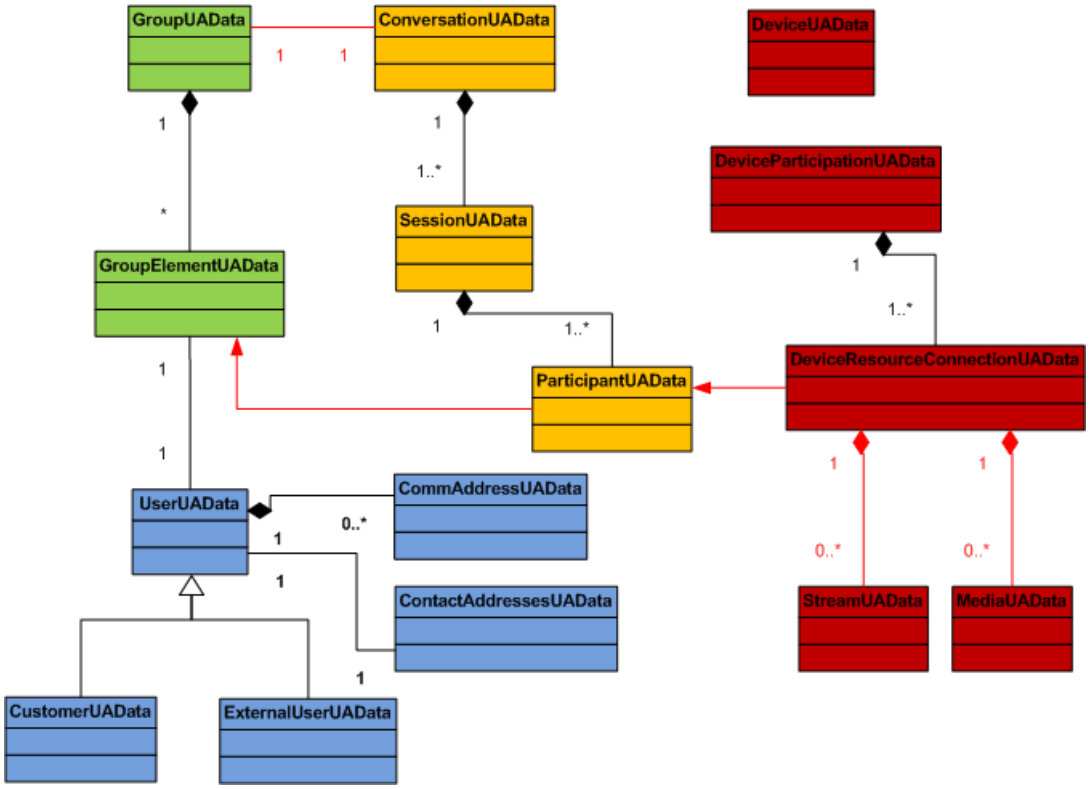


Figure 3.6: User Agent Data Model

- **Conversation Entities (in Yellow):** holds all the relevant information of a collaborative session, which in PUC has one additional level known as *Conversation* which is a group of sessions. Here resides information like topics, timing information and participant details.
- **User Entities (in Blue):** holds user data, which is composed by a user's account details, and registered communication addresses (E-Mail, phone numbers, IM addresses, etc...). An user object can be of the type *Customer* which is an user that has a provisioned account in the platform, or it can be a *External User* which acts as a guest user that does not yet have an account. This distinction exists in order to save contacts that get synchronised from a user's mobile device which are not yet registered in any customer account.
- **Group Entities (in Green):** holds the user membership of user-defined groups, which are part of the created conversations.
- **Device Participation Entities (in Red):** holds device profiles that basically map a set of collaborative features according to each device model capabilities, in the form of *DeviceUAData* objects; and *DeviceParticipation* objects which correspond to an actual user participation in a given collaborative session through a specific device.

Some data objects have *direct relationships* (which are marked in black), and *indirect relationships* (marked in red). The difference is that on the former relationships, objects reference full object instances of the aggregated entity's class, which results in a JSON string conversion that maps aggregated objects in a tree-like structure; while on the latter, objects just reference aggregated entities by their id in order to avoid recursion in the JSON string mapping (see Fig.3.7).

The data model has been designed to minimise the returned JSON string sizes. As such, only relevant data is aggregated in order to minimise the communication with the Gateway through direct references. In a indirect reference situation, where the client receives only the object's id, the client can use the provided *Server User Agents* (which are described in the next section) in order to obtain the correspondent full data object.

Figure 3.7: Example of Direct and Indirect Object References in a Resulting JSON String

3.2.2.2 PUC Gateway System Design

Knowing our data model, we are now in position of addressing the general system design of our PUC Gateway.

We present our design in Fig.3.8 which is composed by several components that are integrated with PUC.

The *Server User Agent* component corresponds to our actual Gateway process, which is composed by several objects, known as *Server User Agents*, that implement different functions of PUC's own Public API:

- **DeviceManagerUA:** not available in PUC, this agent configures and loads device profiles and provides a function for the retrieval of a specific client's participation in a given collaborative

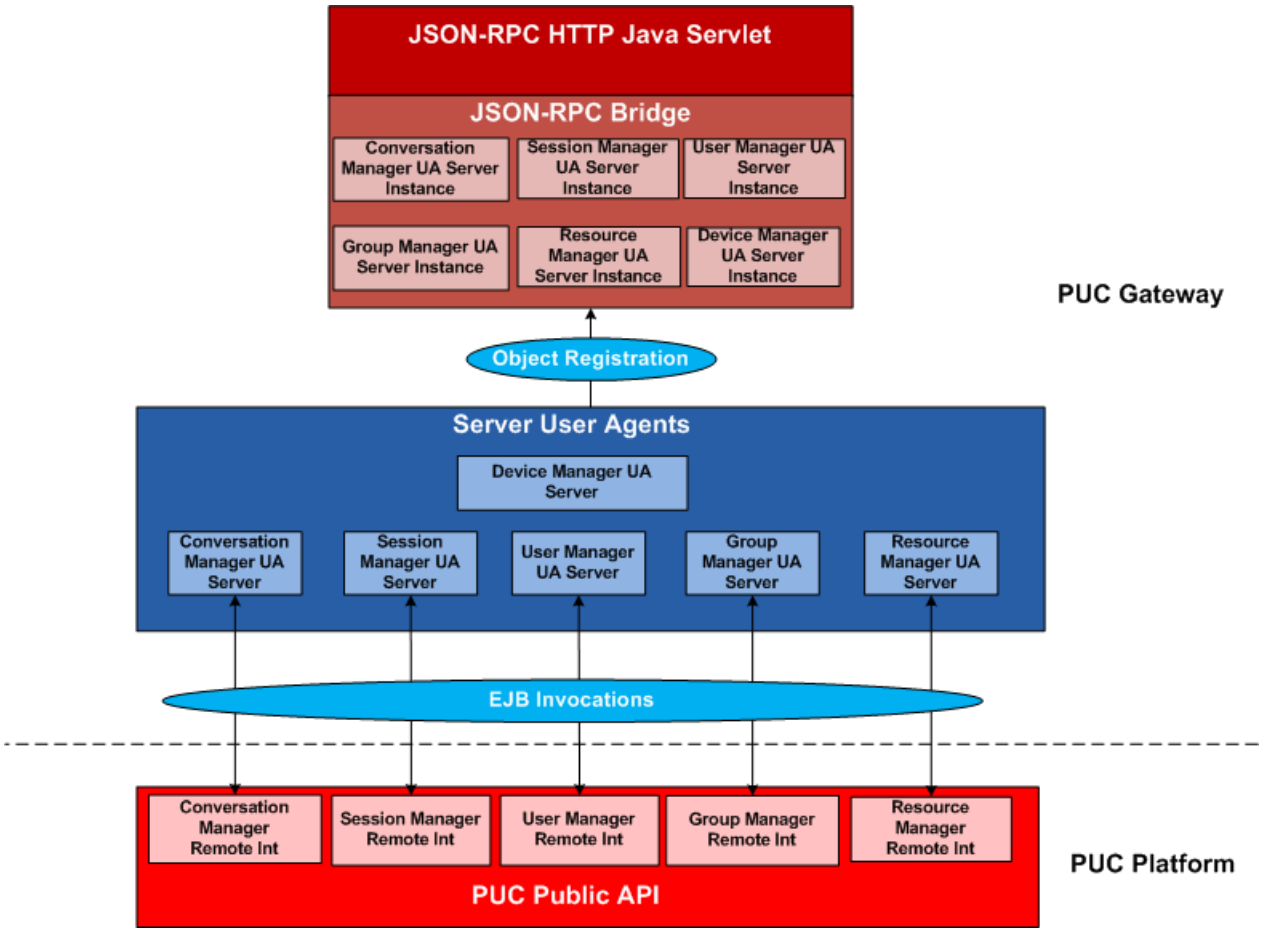


Figure 3.8: PUC Gateway's System Design

session, according to its client device capabilities. This component will be subject of a detailed description in section 4.2.2.

- **Conversation Manager UA Server:** provides functions for the creation of new conversations; conversation retrieval; removal and conversation status updates.
- **Session Manager UA Server:** gives session setup; session status update and session participant management functions.
- **User Manager UA Server:** provides contact synchronisation; account creation and login functions.
- **Group Manager UA Server:** gives group creation; group retrieval and removal functions.
- **Resource Manager UA Server:** provides download functions for network stream or media objects which belong in a given collaborative session.

All Server User Agents are registered in the JSON-RPC Bridge as soon as the application gets deployed in the application server and communicate with the respective PUC Public API interfaces through EJB method invocations.

3.2.3 Mobile Terminal Architecture

In this section we will describe the mobile terminal's system components (represented in Fig.3.9) which are also presented as a layered architecture. The upper layers which are closer to the application side, are

developed by us. The third-party components are properly designated as such in the next description.

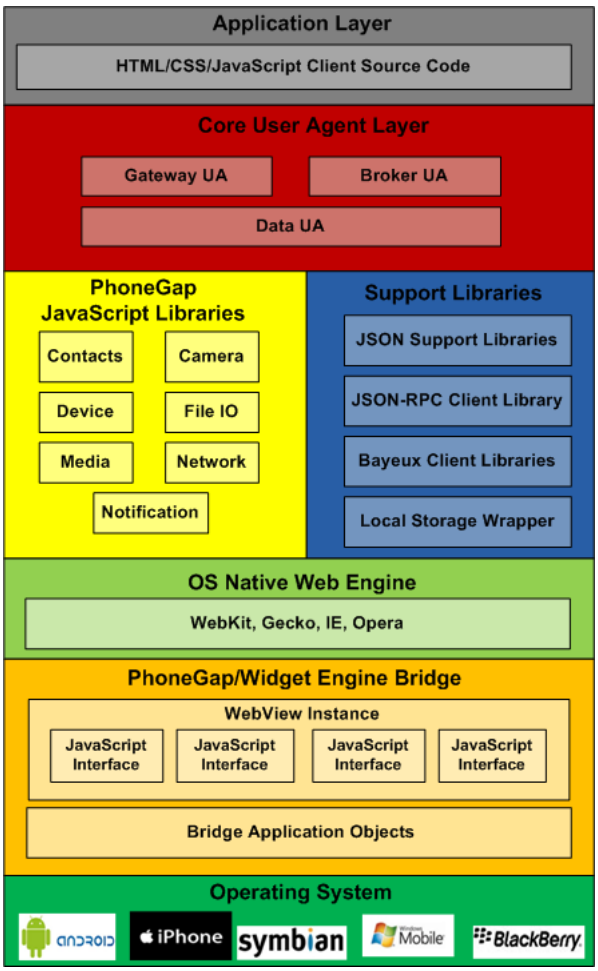


Figure 3.9: View of Mobile Terminal's Architecture

- **Application Layer:** the developed groupware client application (local HTML/CSS/Javascript files) that uses our middleware solution.
- **Core User Agent Layer:** our own set of Javascript libraries, providing an abstraction level to the application developers, who want to create groupware applications. It consists of a *gateway user agent* which is a set of libraries that support groupware functionality (Session Management, Group Management, etc.) and abstracts the JSON-RPC based communication with PUC Gateway to the developer. In this same domain, is the *data user agent* which provides synchronisation functionality with the back-end database and local persistence functions, allowing for a disconnected operation mode. This layer also provides a *broker user agent* which abstracts the communication with the Event Broker, allowing easy registration of event listener routines and event publishing for the end developer.
- **Phonegap JavaScript Libraries (Third party) :** provides the interface with the mobile operating system native functions through JavaScript functions accessible from the application (refer to 2.4.3.3).
- **Support Libraries (Third party) :** The JSON and JSON-RPC *support libraries* help with the creation and parsing of JSON messages and implement the JSON-RPC protocol, respectively. They are used by the *core user agent* in the communication process with PUC Gateway (described in section

3.2.1).The *bayeux client libraries* abstract the Bayeux message transportation protocol to us and finally, the *local storage wrapper* provides embedded database manipulation functions to the upper layer.

- **OS Native Web Engine (Third party) :** provides the Web browser execution runtime, where all the code from the upper layers is interpreted. Implementation is dependent on the application's target operating system.
- **Phonegap/Widget Engine Bridge (Third party) :** it is the framework that makes an interface between the Web code and the operating system. Generally this implementation instantiates native WebView instances, which are presented in full screen mode and it is here that the Web code is interpreted. The communication with the native application objects is accomplished with a set of publicly accessible JavaScript interfaces which are published in the WebView itself, by the instantiated *bridge application objects* built in the native programming language. Depending on the operating system, the Phonegap native sources, or, a compliant widget engine, instantiates these objects . Regarding Phonegap and widget engines, we encourage the reader to review sections 2.4.3.2 and 2.4.3.3.
- **Operating System (Third party) :** provides the execution runtime and the device's native functionality.

We will present a detailed description of the *Core User Agent Layer* in section (3.2.3.1), where all its inner components will be detailed and how each component is represented in the real implementation that is deployed in the mobile terminal.

3.2.3.1 Core User Agent Layer Design

This component corresponds to the middleware layer that is developed by us and is supported by a set of third-party libraries. It is constituted by a set of properly organised JavaScript source files which can be included in any mobile web client which runs on top of PhoneGap or a supported widget engine. It can also run on a Desktop browser, since device dependent functions do not get included (loading device contacts for example).

The design model is hereby presented in Fig.3.10.

In the presented figure, the reader can observe how the different middleware components are organised and how they are mapped to the real JavaScript source files. In our *Core User Agent Layer*, we have:

- **Gateway User Agents:** a direct map of PUC Gateway's Server User Agent functions, but, for Web application developers that make local JavaScript calls through their client application. Each API is mapped by an independent JavaScript source of which all reference the master Gateway script that provides access to an object that holds all method signatures that are hosted by the Gateway and directly uses Jabsorb's JSON-RPC library in order to properly make the remote invocations.
- **Broker User Agent:** in the form of a single Public API in a single JavaScript source, provides routines for end-developers to register their application's callback routines which are properly executed when their associated event is launched in the server-side platform. This user agent uses the CometD Dojo API in order to implement a event routing logic based in Bayeux protocol which is totally abstracted to the end developer.
- **Data User Agent:** holds functions to create JavaScript objects that conform with the *UAData* model (which was described in section 3.2.2.1). Additionally, it also supports local object persistence by providing object saving and loading functions that are properly implemented by the Lawnchair libraries, but abstracted from the end developer.

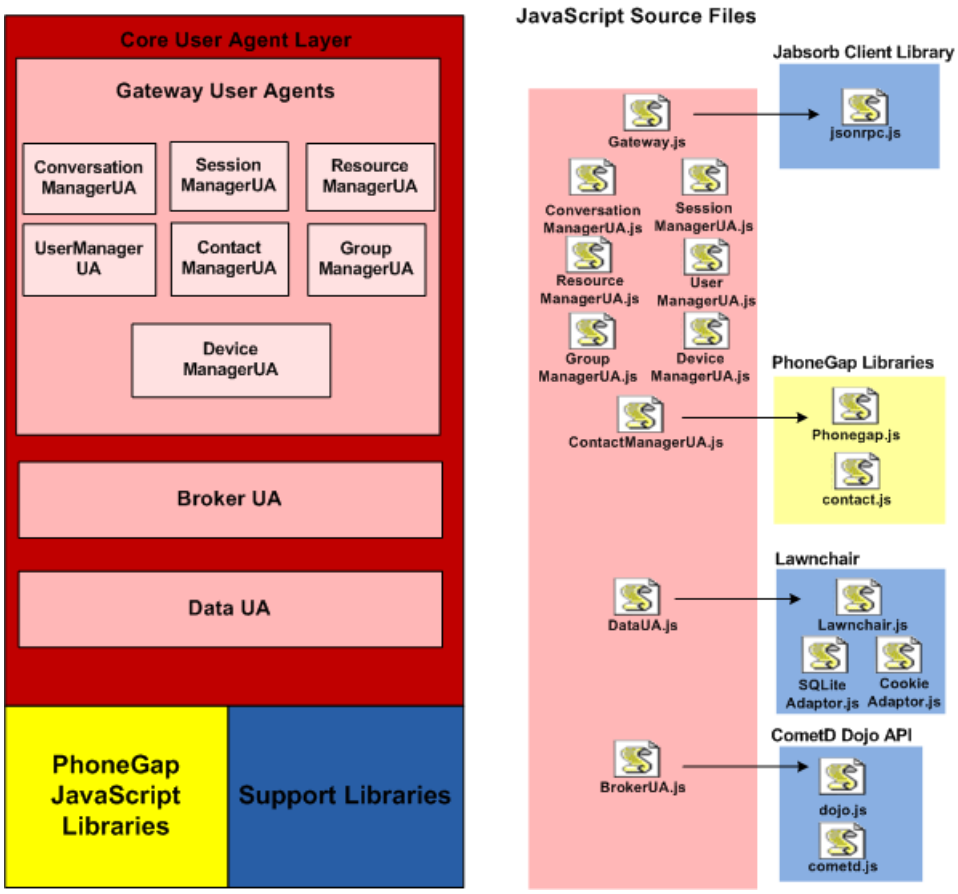


Figure 3.10: Detailed View of The Mobile Terminal Middleware Component

3.3 Chapter Summary

During the course of this chapter, we have presented the architectural overview of all the developed components and the communication processes between them (see section 3.1). We have also detailed each system component individually, going from the PUC Gateway (section 3.2.2.2) to its integration with the PUC Platform which provides a unified framework of collaboration service enablers, making the Gateway a in-house client of PUC, and all the components hosted in the mobile terminal itself (both our middleware layer and supporting third party libraries), in section 3.2.3.

Our PUC Gateway gains immediate support to requirements, such as: access control, concurrency control; session management requirements and the provision of a persistent information base, by integrating with the PUC Platform, which already has support for these requirements. Therefore, our architectural design is focused on offering the additional requirements of code reuse and portability; adaptability; data synchronisation; Offline work; Effective user awareness and extensibility.

All of our architectural designs follow a layered design that properly makes use of already available third-party solutions in order to support the above mentioned requirements.

Concerning the communication processes (see section 3.2.1), we capitalise on the use of the JSON-RPC message protocol over HTTP in order to provide a lightweight RPC solution to mobile resource constrained devices. In order to improve user awareness by minimising latency in the delivery of application events to the mobile client, we also make use of the Bayeux message transportation protocol in its *long-polling* variant which makes use on long-lasting HTTP Get requests that only get responded when events occur.

The inherent technical limitations of mobile devices and the additional requirements concerning resource adaptability, forced us to also design a simpler data model for the PUC Gateway component, the *User Agent Data Model*, which is basically a subset of PUC's own data model that directly maps obligatory PUC data objects for the client application, without complex data types and circular references in order to produce JSON string conversions as clean and small as possible (see section 3.2.2.1).

All the hereby described components were designed in hopes of resulting in a good implementation that properly fulfils the accomplishment of the above mentioned requirements. As for the actual implementation details and challenges, we will be describing them in the next chapter.

4 Solution's Implementation

"If I have seen further than others, it is by standing upon the shoulders of giants."
– Isaac Newton, English Mathematician and Physicist, "father of the modern science"

In the previous section we presented the architectural design model of our solution which is composed by multiple and tightly integrated entities each with distinct in-house developed and third party components. Now, we will be delving in the description of our development process and followed implementation models, in order to turn our earlier proposed design from a concept, into a coherent software implementation.

Several practises have been followed in order to minimise effort and development time, such as:

- **Incremental approach:** by favouring a bottom-up development process that guarantees that components which serve another, are completed first in an incremental fashion.
- **Safeguard extensibility:** by always guaranteeing that new components can be easily added to our implementation on every development phases.
- **Do not reinvent the wheel:** by leveraging available technologies for the development of our solution. Preferably, technologies that are widely used today.

Object Oriented programming is a widely followed model in PT Inovação of which it's in-house development teams have a solid knowledge base. Therefore, in order to exploit this knowledge base and comply with the above mentioned practises, we favour a Java based object oriented implementation where possible, which we were able to carry through.

The design model of our mobile client assumes a Web based implementation on the application and middleware layers, which on a first impression, goes against a real object oriented approach that commonly serves middleware components. Even so, we tried to push object oriented programming even in this layer and managed to produce object oriented JavaScript code on the middleware layer, and a totally Java based Web application built in Google Web Toolkit (described in section 4.1.5) that serves as our mobile groupware app prototype.

Putting this matter aside, we are now in position to present this chapter's roadmap, which starts by describing the technological tools that we chose to support our own in-house developed components in section 4.1. The reader can refer to the implementation details of our solution's components, in sections: 4.2 for the PUC Gateway implementation; 4.3 for our client oriented middleware layer, and finally 4.4, which describes the implementation of our prototype mobile app that makes use of our middleware components, and server-side implementation, in order to support groupware functionality.

4.1 Used Technologies

We have managed to leverage several technologies that support our own components whether we are talking about the server side or the client side implementations. Here, we present an overview of these technologies. They are used on the server side, client side or even in both sides (see Table.4.1).

Note: in this section, we will not present a detailed description of Phonegap because we have already done so in our related work section (refer to 2.4.3.3).

	Server Side Implementation	Client Side Implementation
Java EE	✓	
JBOSS AS	✓	
Jabsorb	✓	✓
JSON Tools	✓	✓
CometD	✓	✓
GWT		✓
Phonegap		✓
Lawnchair		✓

Table 4.1: Technology Usage Map

4.1.1 Java EE and JBoss AS

Java EE or Java Platform Enterprise Edition[38] is a platform whose objective is to reduce the development complexity of enterprise oriented Java applications, by providing an extensive set of API's to developers that wish to deploy distributed and modular multi-tier services.

The platform's version 5.0 (Java EE 5) focused on offering easier application development by improving some of the features introduced in previous versions, by providing annotations that ease the definition of Web services, Java to XML serialisation, etc.

Applications developed with Java EE are supported by an *application server* which is responsible for hosting and executing the several modules of which they are composed of.

For the development of our server side implementation, we chose *JBoss Application Server*[39] (JBoss AS). It is an open-source implementation developed by Red Hat. Our choice is due to its extensive functionality and great support offered by developers and community. Another reason is due to PT Inovação's long history of JBoss AS usage, of which we can take benefit from.

4.1.2 Jabsorb Framework

Jabsorb[40] is a lightweight framework that allows any Web application to call Java methods through JavaScript code. Its main advantage lies in the abstraction of the JSON-RPC transport protocol (which we talked about in section 3.2.1) to the end developer.

The framework provides both server and client side implementations. A developer who wishes to expose a set of Java methods, just needs to build the provided Java libraries and the JSON-RPC Servlet, and simply, add a servlet mapping in the configuration file of the server application (which is hosted in an application server of choice) that holds the corresponding Java objects.

As soon as the JSON-RPC Servlet gets deployed in conjunction with the developer's server application, a JSON-RPC bridge is also installed and can be manipulated through Jabsorb's Java libraries. Here, a developer can register any Java object in the bridge in order to expose it to client Web applications.

The client Web application's developer, just needs to include the provided JSON-RPC JavaScript library, which abstracts all the details of marshalling/unmarshalling objects and the JSON-RPC messages, and makes remote calls of Java objects as if they were local objects residing directly in the browser.

We have identified the JSON message format and the JSON-RPC transport mechanism as the best alternatives in order to deploy lightweight Web services to constrained mobile devices. But, in the beginning, this alternative revealed itself to be troublesome and time-consuming due to the lack of available

[38] B. Shannon, JSR 244: Java Platform, Enterprise Edition 5 (Java EE 5) Specification - http://jcp.org/aboutJava/communityprocess/final/jsr244/index.html
[39] *JBoss AS Project - http://jboss.org/jbossas*
[40] Jabsorb Framework - http://jabsorb.org/

tools that could help us in implementing a solution based on these mechanisms, when compared with SOAP based tools which are more popular and widely supported. Jabsorb's appeared as our number one choice, because simply, there was no another choice that served our requirements with the same level of simplicity and effectiveness as this one and with the communication mechanisms that we were seeking.

Its main drawbacks are the lack of security mechanisms (messages get sent and received in clear) and service discovery (client is assumed to know both server URL and function names).

4.1.3 JSON Tools

JSON Tools[41] is a handling tool set for JSON encoded messages (refer to section 2.4.2). It is provided in a Java library set that can be used on any Java application.

Several tools are offered by this tool set. However, we only use two of these tools in our implementation, namely:

- **JSON Mapper:** maps a Java POJO to JSON, and maintains the resulting JSON string as clean as possible in a human perceivable form. This tool is adequate for the migration of Java objects to client applications programmed in other programming languages, examples being JavaScript, PHP and ActionScript. Although it gives cleaner and smaller JSON strings which are ideal for resource constrained devices, not all original data structures are maintained, resulting in information data loss.
- **JSON Serialiser:** serialises Java Objects to a JSON representation. Here, all original data structures are maintained in the text representation preserving recursion, references and primitive types. Thus, this tool is ideal to migrate objects from server to client back and forth, and still, maintaining all the original Java object structure. The major drawback is the resulting JSON string size, which can be multiplied by the number of recursions a Java object has, thus, turning this tool not ideal for object migration whose destinations are constrained clients such as mobile devices for example, or networks with slow data rate connections.

Java object migration is essential to standardise our data model in both client and server implementations. If we had to implement a customised Java object migration to JavaScript that complied with our data model, it would be a extremely complex and time consuming effort. By using JSON Tools we instantly convert Java objects from server to client, or JavaScript objects to Java objects while automatically complying with an uniform data model on both sides.

4.1.4 CometD

The CometD[42] Project by Dojo Foundation aims to produce the *bayeux specification* (which we described in section 3.2.1).

The name *Comet* refers to a message transmission mode which is purely based in *server-push* style interactions, which are commonly achieved by *long-polling* techniques. Our solution's architectural model proposes this interaction in communication with the Event Broker, thus the reader is encouraged to review section 3.2.1 in order to understand how this mechanism works.

Aside from producing the Bayeux specification, CometD also provides a set of implementations of this specification in several programming languages, both on server side (Java) and on client Web Applications developed with JavaScript.

[41] JSON Tools Project Page - http://jsontools.berlios.de/
[42] http://cometd.org/documentation

The client side implementation comes in the form of a common JavaScript API and implementation that has bindings for the Dojo toolkit and for the jQuery toolkit.

The JavaScript API also provides a *publish/subscribe* event logic, which enables the subscribing of events in specific channels which can be defined freely by the developer on the server side implementation. Servers or clients can also publish their events on a specific channel that each and every subscribed entity receive (see Fig.4.1) .

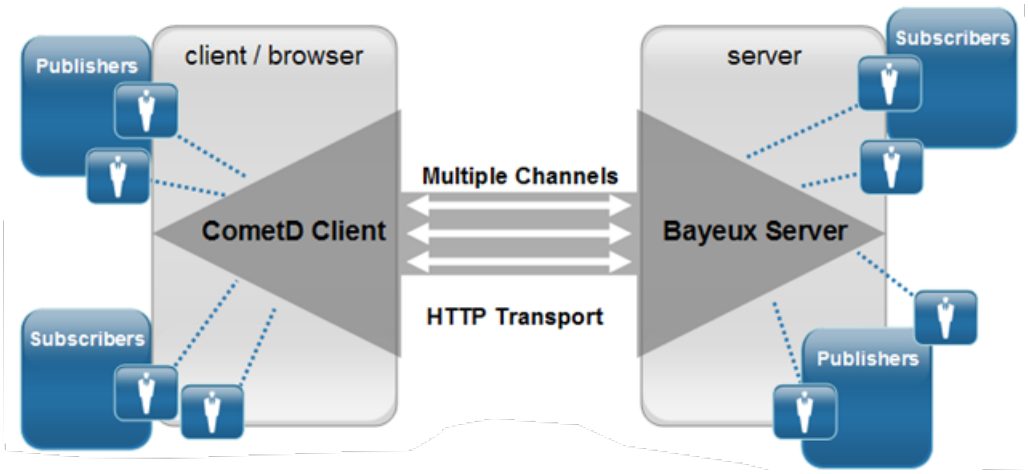


Figure 4.1: CometD Architecture

By easily allowing the implementation of a *server-push* event router for our Web developed client application, that minimises latency through a *long-polling* transport mechanism and *publish/subscribe*-based event routing, this framework offered tremendous value in our development phase. If we consumed the same development time as by using this technology, we would only have managed to implement a traditional polling based event broker without publish/subscribe logic, which in comparison, would have been extremely stale.

4.1.5 Google Web Toolkit

Google Web Toolkit[43] (GWT), is a toolkit for building complex Web applications. Its main singularity is that applications are developed with a set of core Java APIs and pre-defined Widgets making it possible to write AJAX applications in Java and then compile the produced source code to a highly optimised JavaScript that runs across all modern Desktop browsers, including even, some mobile browsers.

GWT also makes the interaction with handwritten JavaScript possible by using its JavaScript Native Interface[44] (JSNI).

In order to support communication with back-end servers in its applications, GWT also provides the GWT RPC framework that transparently makes calls to GWT Java servlets through a customised RPC protocol, and takes care of object serialisation and other details.

The GWT RPC framework was not the feature that we sought, because for that we have Jabsorb, and we want our core middleware layer client code to be as raw JavaScript as possible. If we developed our core layer in GWT we would be forcing developers to use our core layer in GWT too or using JSNI which in our foresight, was not acceptable, considering that not all developers have expertise in building GWT applications. In the Web development universe, we can cover a higher number of developers using raw JavaScript as it is an older and more available standard.

[43]GWT Overview - http://code.google.com/intl/pt-PT/webtoolkit/overview.html
[44]GWT JSNI Overview - http://code.google.com/intl/pt-PT/webtoolkit/doc/latest/DevGuideCodingBasicsJSNI.html

What really interested us in GWT was Java based development and ulterior compilation to optimised JavaScript. This sole feature allows an object oriented approach in the development of our client application, relieving us from browser quirks and DOM manipulation while permitting re-usage of already developed user interface elements.

4.1.6 Lawnchair

Lawnchair[45] is a client side JavaScript library that enables JavaScript object saving in a browser's local storage adaptor.

JavaScript objects are converted to JSON strings and stored in database tables, or cookies depending on the developer's chosen adaptor. The JSON strings are mapped by a user defined or generated string.

Besides JSON string saving, Lawnchair also offers JSON string retrieval, removal and iterators for local saved data.

What is interesting about this framework, is that it offers an unified API for local storage implementation, allowing for object persistence in a Web developed mobile application, which is one of our main requirements.

With Lawnchair we can always use the same local storage implementation even if the target operating system browser does not support HTML5 Local Storage, or even if a specific Phonegap implementation does not offer an SQLite Adaptor. That is due to Lawnchair wide browser support, which the reader can observe in the below table (Table.4.2).

	Operating System					
Adaptor	Android 1.5, 1.6	Android 2.0+	iPhone OS	Symbian	BlackBerry	Windows Mobile
Webkit SQLite		✓	✓			
Gears SQLite	✓					
DOM Storage		✓	✓			
Cookie	✓	✓	✓	✓	✓	✓

Table 4.2: Lawnchair's Mobile Operating System Support

Better yet, we have the possibility to support local storage in all of our target operating systems. We just need to use the adaptor according to the operating system capabilities.

4.2 PUC Gateway

In this section we will present all the relevant implementation details of the system component that makes the connection of our mobile client with the PUC platform, the PUC Gateway.

We will begin by explaining how we leveraged the chosen and above described technologies for this component's implementation, in section 4.2.1,. The technologies that were used in this implementation are: Java EE + JBoss AS (section 4.1.1); Jabsorb (section 4.1.2) and JSON Tools (section 4.1.3).

Next (in section 4.2.2), we will detail the implementation of the PUC Gateway's *Device Manager* component, in order to ackowledge how we implemented a system component that accomplishes resource delivering according to the capabilities of the accessing client devices.

[45] Lawnchair Home Page - http://blog.westcoastlogic.com/lawnchair/

4.2.1 Leveraging Technology

PUC Gateway's main implementation challenges lied in how to leverage available technology in order to minimise the development time of otherwise complex and time consuming tasks, if technology that could help us would not be available at all. These complex tasks are essentially related with the Web hosting and initialisation of application components (sections 4.2.1.1; 4.2.1.2 and 4.2.1.3) and object migration between the server side and client side implementations (section 4.1.3).

4.2.1.1 Deploying the JSON-RPC HTTP Servlet

In order to turn our Gateway functions accessible to applications executing in Web browser based clients, we deployed the JSON-RPC servlet provided by the Jabsorb framework. We just needed to deploy the servlet in our JBoss application server and add the respective servlet configuration and mapping in our XML application descriptor file, which is loaded when the application is deployed (see Fig.4.2).

```
<servlet>
  <servlet-name>JSONRPCServlet</servlet-name>
  <servlet-class>org.jabsorb.JSONRPCServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>JSONRPCServlet</servlet-name>
  <url-pattern>/JSON-RPC</url-pattern>
</servlet-mapping>
```

Figure 4.2: JSON-RPC Servlet Configuration

Clients are then allowed to make JSON-RPC calls to the gateway, as long as they send their requests to the configured url, which in our case corresponds to: **http://gatewayAddress:port/JSON-RPC**.

4.2.1.2 Hosting the Application

We needed to know how to host our PUC Gateway's application objects. For this, we used the JavaEE specification in conjunction with the JBoss Application Server (described in section 4.1.1).

Our first challenge was to host a Java object that would be responsible to initialise all the necessary Gateway objects as soon as the application would get deployed on the server, including, the JSON-RPC bridge, the Device Manager component and all the Server User Agent objects to be exposed remotely to Web clients. To accomplish this, we created an *initializer object* with the @Service EJB annotation (see Fig.4.3).

Services are singleton beans and are not pooled, so only one instance of the bean exists in the server. Additionally, Services implement a **start()** method which is executed when the application gets deployed. Therefore, this implementation serves our purpose adequately.

In order to allow Java based clients to access our Gateway Application objects, mainly for testing and validation purposes, we opted to deploy our PUC Public API Mapper as a set of *Stateless EJBs* mapped by remote interfaces (see Fig.4.4).

When a client invokes the method of a stateless bean, the bean's instance variables may contain a state, but only for the duration of the invocation. When the method is finished, the state is no longer retained. Except during method invocation, all instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client. Thus allowing the creation of a Java based test battery that simulates a large number of clients that simultaneously make requests on the Gateway.

```
@Service
@Local(GatewayInitializerLocal.class)
public class GatewayInitializer implements GatewayInitializerLocal {

    public void start() {

    // initialization code here

        -> JSON-RPC Bridge Creation
        -> Device Manager Initialization
        -> PUC Public API Mapper Initialization
        -> Object Registration in JSON-RPC Bridge

    }

}
```

Figure 4.3: Gateway Initialisation

```
// Object Implementation

@Stateless
@Remote(ConversationManagerUARemote.class)
public class ConversationManagerUAServer implements
    ConversationManagerUARemote, Serializable {

}

// Object Declaration
@EJB
private ConversationManagerUARemote conversationManager ;
```

Figure 4.4: Gateway Application Objects as Stateless EJBs

4.2.1.3 Creating the JSON-RPC Bridge

In order for client requests to be routed for the corresponding Java objects, a JSON-RPC Bridge is needed. In Jabsorb's case it is deployed in conjunction with the JSON-RPC Servlet. The only thing left for the developer to do is to register all Java objects he we wants to expose for Web browser clients.

The JSON-RPC Servlet uses a global singleton JSON-RPC Bridge instance or optionally one in the user's HttpSession (if it exists). We have the option of placing a JSON-RPC Bridge into a user's HttpSession to allow the servlet to make calls on objects that are exported only to that specific user session (which may contain stateful data related to the specific user).

Due to the fact that security issues are, for now, not our concern and specific user treatment is not a requirement of ours, we opted to use a singleton bridge instance in our implementation, being object registration done in the way described in Fig.4.5.

The **registerObject()** function maps an object reference to a string the developer thinks as adequate for the object's remote name. In our case, in order to standardise object reference names to Web clients, we opted to always give the object's class name as it's remote reference name. The JavaScript developer who needs to make a remote call of a specific function of a registered Java object, just needs to type a line code as the following:

JSONRPClient.ObjectClass.functionName(args);

```
//initialization of exposed application objects
GroupManagerUARemote groupManager;
ConversationManagerUARemote conversationManager ;

//gets the  Global JSON-RPC Bridge instance from the JSON-RPC Servlet
JSONRPCBridge bridge = JSONRPCBridge.getGlobalBridge() ;

// adds the correspondent objects in the bridge
bridge.registerObject(GroupManagerUAServer.class.getSimpleName(), groupManager) ;
bridge.registerObject(ConversationManagerUAServer.class.getSimpleName(), conversationManager) ;
```

Figure 4.5: JSON-RPC Bridge Configuration

4.2.1.4 Object Migration Using JSON Tools

To implement object migration between PUC Gateway and the mobile terminal we used the provided conversion tools by the JSON Tools framework (described in section 4.1.3).

JSON Tools offers a comprehensive toolset, that requires more than just one simple call in order to convert an object. Our only objective was to use the tools in order to implement two generic and publicly accessible methods that would simply convert a Java object from the Gateway's data model to a string that complied with the JSON data format and another that would make the opposite, that is, a JSON string to the correspondent Java object of our data model. All this, with just one line of code for each operation (Fig.4.6).

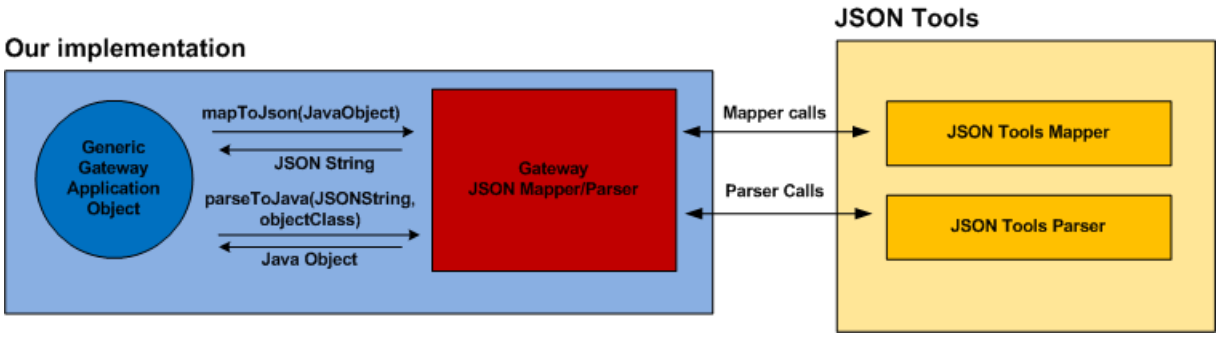


Figure 4.6: PUC Gateway's Data Conversion Using JSON Tools

The usage of the JSON Mapper tool instead of the Serialiser tool was more adequate due to the mobile client's limitations, because JSON strings converted with the Serialiser tool retain all Java data structures, resulting in JSON strings which are ate least, ten times as bigger and not as human perceivable as the ones generated by the JSON Mapper tool. Of course there is a downside here too, because smaller strings result in information loss. Concerning this latter issue, we developed a solution which will be described in the next sections.

4.2.2 The Device Manager Component in Detail

Proliferation of different device models, mainly in the mobile device market, introduces new requirements in resource delivering according to the target device's capabilities.

We have managed to design a Server User Agent component that accomplishes these requirements (see Fig.4.7).

The information regarding a device's capabilities and supported collaborative features is saved in the form of *DeviceUAData* object instances. This information is generated from two manually edited

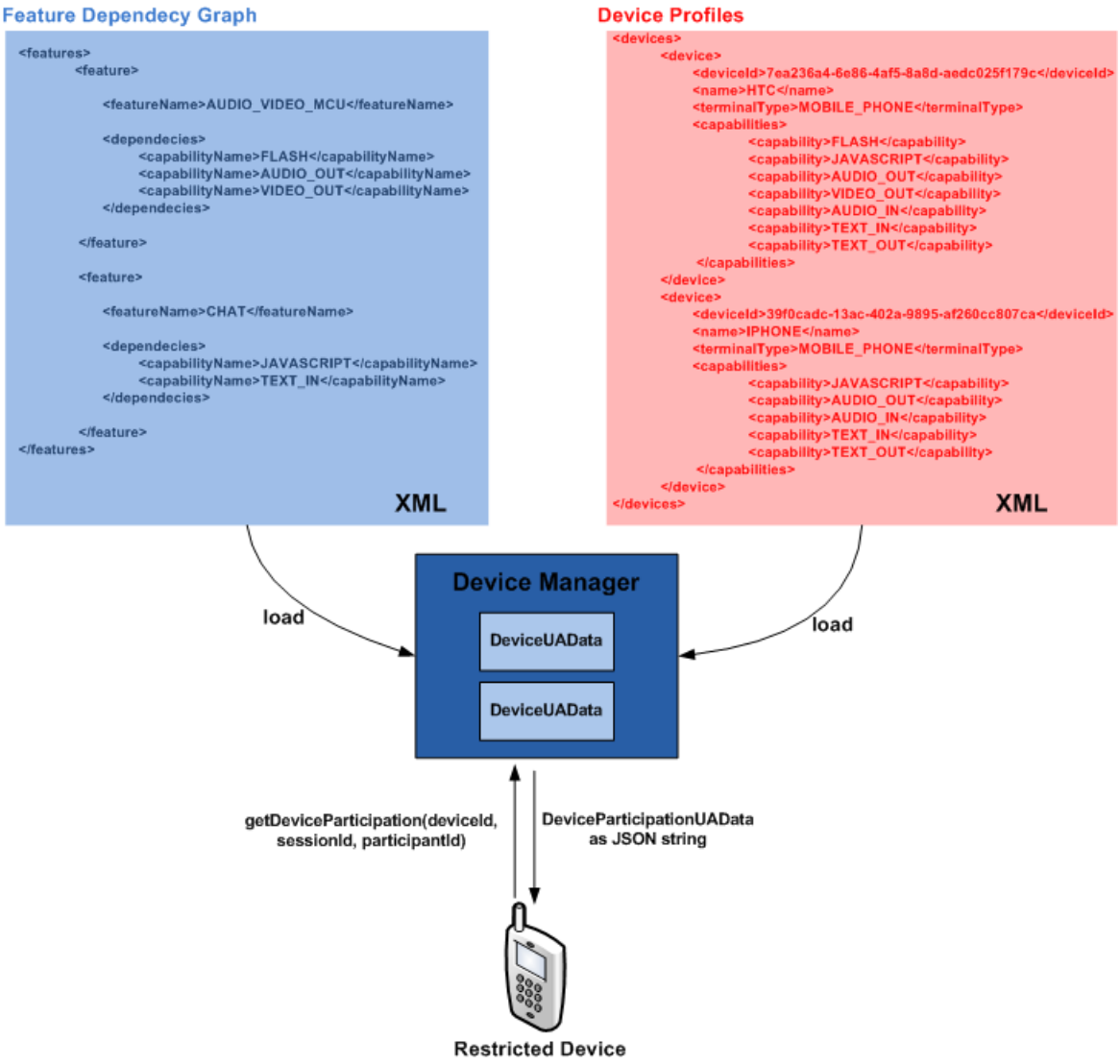


Figure 4.7: Detailed View of PUC Gateway's Device Manager

XML documents, in conformation with a specific XML schema and loaded on the Gateway's application start up:

- **Feature Dependency Graph:** maps a set of needed device capabilities in order to fully support a specific feature.
- **Device Profiles:** defines a device profile which is composed by a set of identification properties and a set of technological capabilities.

In this way, a client only downloads resources that are technically compatible with his/her mobile device model, thus minimising transmitted data and avoiding erratic client application behaviour. An application developer is relieved from implementing this behaviour on the client side, which would be impractical, considering that the mobile device market is always being flooded by new devices with new capabilities. Therefore, in order for the system to support new devices, the PUC Gateway's developer just needs to alter metadata in a centralised source, thus avoiding, forced changes on the application side.

The downside of this approach is that we need to maintain a database that maps a device profile document to a universal model identification number that univocally represents a mobile device model. Worst, a client application developer needs to know this mapping and compile his/her client application with a configuration parameter that makes the application use the correspondent identification when the operation to download the resources gets executed.

An alternative which is not implemented today, would be to make device profiles only based on the operating system of the mobile device, which would make for a less granular device differentiation, but, for a client implementation that would not need any changes at all, because the operating system information, would be fetched by the device's native functions.

This model will be extended in the future, in order to additionally support, physical properties like screen dimensions or input methods (in order to build auto adjustable user interfaces) and user preferences regarding content delivery, with an example being, a user preferring video to be displayed on his/her PC while the audio on his mobile device.

4.3 Mobile Terminal Middleware

Here, we will be looking into our client side middleware, which is actually the component that makes the development of a mobile groupware application that uses our server-side infrastructure, possible.

This implementation is totally JavaScript based in both, our in-house developed libraries and third-party libraries. It assumes the form a groupware-oriented middleware component that is tightly integrated with a specific third-party bridge API that runs o top of the Web engine's execution runtime.

The main implementation challenge of this layer was in how to leverage the available supporting technologies in a way that we would become completely transparent to end developers that would take advantage of our middleware layer.These challenges are thereby described in sections 4.3.1 and 4.3.2.

4.3.1 Getting Device Contact Information with Phonegap

An end developer that wants to load contacts from a mobile device through our middleware layer desires that all the operation's technical details, including the communication with the operating system and data model conversions, to be as transparent as possible. Well, PhoneGap Contacts API already solves the first problem, but not the latter.

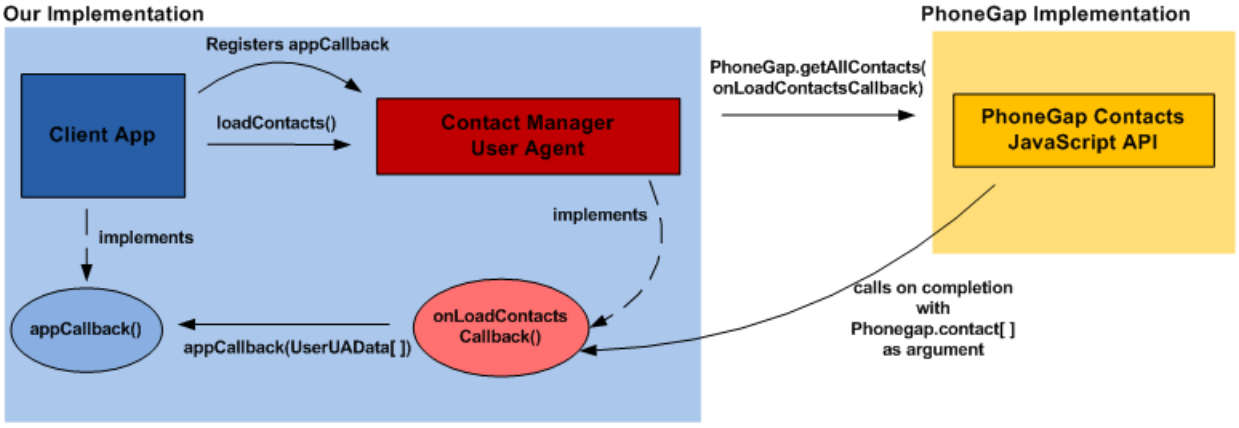


Figure 4.8: Abstracting Phonegap Contact Loading to the End Developer

That is due to fact that PhoneGap loads contacts in a data presentation that is utterly different from our *UAData* model.

Our solution is represented in Fig.4.8. The client application just needs to register it's own callback function that gets executed when contact loading is done, and call the **loadcontacts()** operation. What happens on the inside is that, the *Contact Manager* component uses PhoneGap's own **getAllContacts()** function that returns contacts in PhoneGap's data model in a Contact Manager's own callback routine that is sent to the PhoneGap function. Our component then converts this contact information to our UA-Data model and returns it by calling the previously registered application callback routine that handles the data in a way that is customly defined by the client application developer.

4.3.2 Implementing the User Agent Event Broker via CometD

Another challenge was to transparently give end developers, the possibility to register any application callback routine in response to certain platform launched events.

Obviously, it is not desirable for the end-developer to worry about the intricacies of developing a CometD based application. So, we had to turn our *Broker User Agent* as transparent as possible.

In order to best illustrate our implementation, we present an interaction example of the involved parties in Fig.4.9:

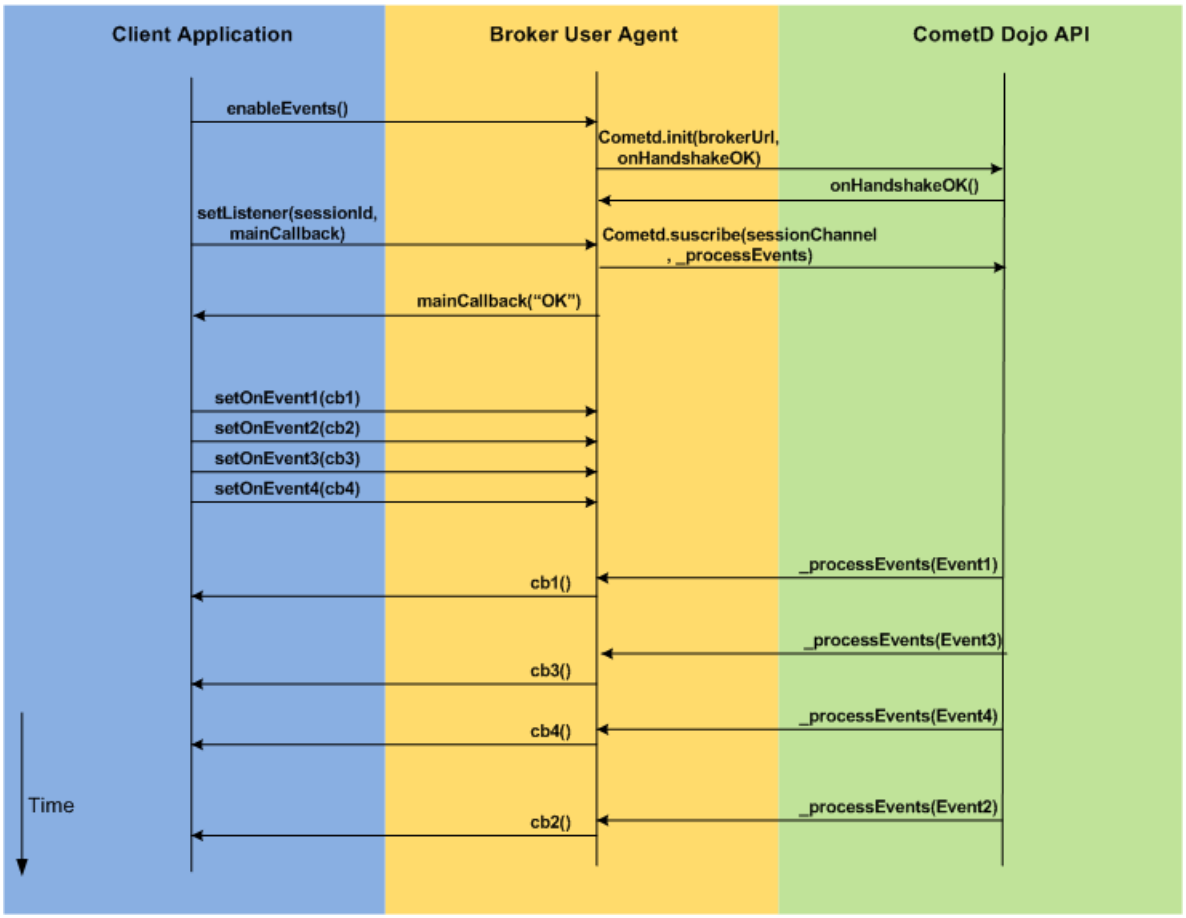


Figure 4.9: Example of Broker User Agent and CometD Interaction

A developer just needs to, first, enable our Broker User Agent, in order for it to initialise the Bayeux connection with the server side implementation through the CometD API. Then, the developer can set the User Agent to listen for events in a given collaborative session. As soon as our agent sees that it's listener has changed, it makes use of the CometD's *publish/subscribe* logic to subscribe to the Bayeux

message channel of the corresponding session which is hosted by the server-side Bayeux Event Broker. Our User Agent has a **processEvents()** private routine which is called by the CometD API everytime an event from the subscribed session gets launched. End developers have the possibility to set any function from his/her application to wait for execution as soon as the corresponding event gets launched, through the **setOnEnvent(callback)** routines. When the **processEvents()** routine gets called, it executes one of these routines, according to the events they were registered to.

With this implementation we managed to get a straightforward way for developers to define customised application behaviours according to different events launched in a given collaborative session. Developers can then use our event specification (see Table.4.3) in order to increase user awareness in their application.

Session Events	Participant Events	Resource Media Events	Terminal Events
SESSION_CHANGED	PARTICIPANT_CHANGED	RESOURCE_MEDIA_ADDED	TERMINAL _ADDED
SESSION_STATUS_CHANGED	PARTICIPANT_STATUS_CHANGED	RESOURCE_MEDIA_REMOVED	TERMINAL_REMOVED
	PARTICIPANT_ADDED		
	PARTICIPANT_REMOVED		

Table 4.3: Broker User Agent's Event Specification

4.4 Mobile Groupware Application Prototype

In order to validate our terminal middleware layer we developed a groupware application prototype that makes use of both the aforementioned layer and the server side implementation (Gateway and the PUC Platform).

Our application is all Java based, with the exception of the interface of the application components with the core JavaScript layer which needs to be raw JavaScript based and in conformity with the GWT JSNI specification.

Our application components are divided in single and independent widgets which are easily extendable with new in-widget components by using the Java inheritance mechanism or even by adding new widgets, which all are managed by our own developed *Widget Controller component*. In this way, we managed to always follow an object-oriented design in order to comply with our *extensibility* requirement.

The reader can get a glimpse of our general application design in section 4.4.1 in order to know how our widget components are integrated in a common framework that gets deployed to the end application user.

The reader can then further refer to section 4.4.2, which addresses in the implementation details of our *Core Application Widgets* and *Resource Widgets*.

4.4.1 General Application Design

Our application's general design is presented in Fig.4.10:

The application is composed by the following components:

- **Setup Constants:** a configuration file which holds setup constant values that can be loaded by any of our application's components.
- **Main Entrypoint Class:** this component holds the entrypoint methods which are executed as soon as the main application's html file gets open. It is responsible to instantiate all the below components (Widgets, the stack and the Widget Controller). The instantiation process depends on

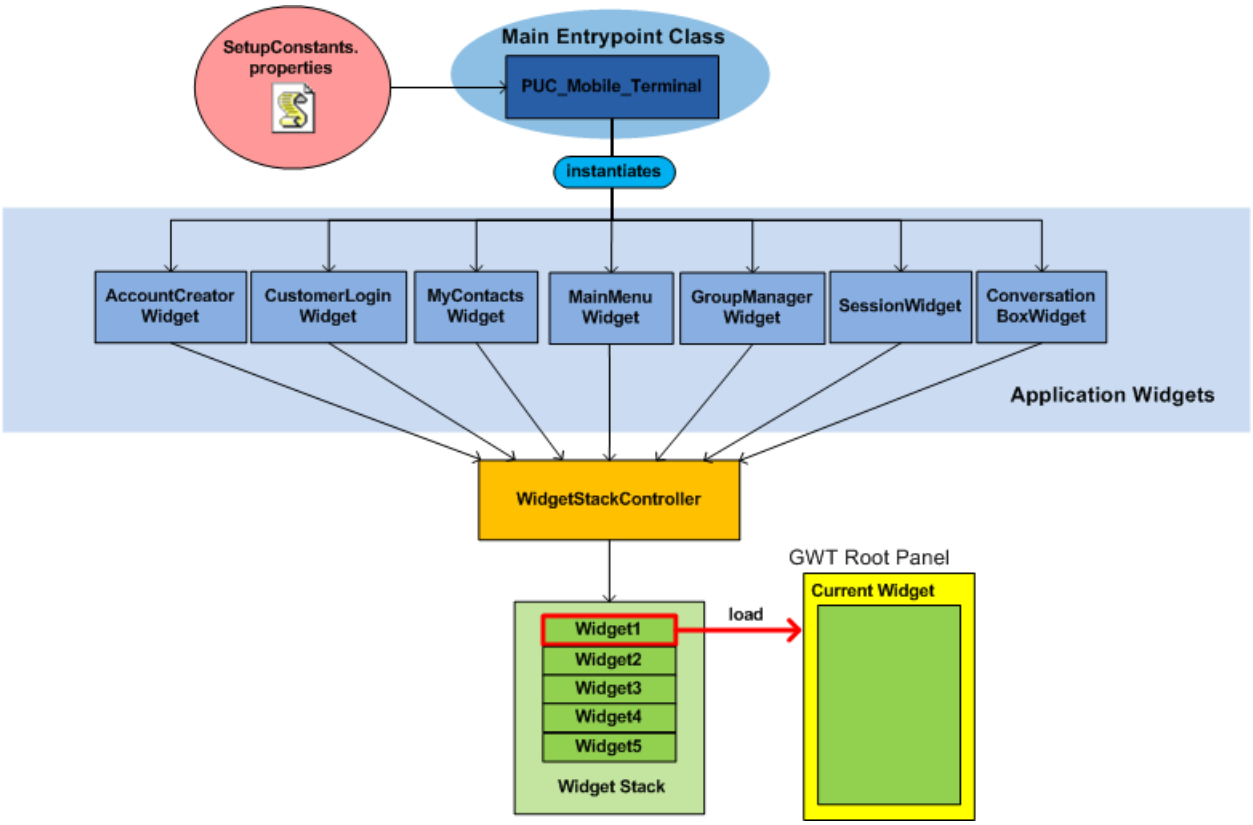


Figure 4.10: General Design of our Mobile Groupware App Prototype

the values stored in the Setup Constants, examples being, server addresses or specific widget sizes and behaviours.

- **Application Widgets:** form the basis of our collaborative application and each widget has a specific function. Specific functions from each Widget may use the *Widget Stack Controller* in order to control the application's execution flow. For example, the Conversation Box Widget uses the controller in order to switch the application view to the Session Widget when a given session gets open from the Conversation Box.
- **Widget Stack Controller:** this one is also instantiated by the Entrypoint Class and gives the developer the possibility to control the application's widget stack. Widgets are switched within the stack in a *push/pop* fashion with the top widget being the actual widget that gets executed in the GWT root panel that the end user gets to see in action.

All the application execution flow through different widgets is made through the Widget Stack. This provides better extensibility because application flow control code does not get dispersed in the widgets themselves. New widgets that are added later on to the application, get to be used just like any other that were previously available with little to no impact in already developed components.

All the application code which corresponds to all the aforementioned components, gets compiled by the GWT compiler, which in turn, transforms the developed Java based code into a package composed by generated JavaScript source files and a main html file which holds the frame that corresponds to the GWT main root panel where all the generated code gets loaded.

4.4.2 Groupware Prototype Widgets

Now that the reader has an insight of our application's general design, we can delve in a detailed overview of our *Widget* components. The reader may note that, here, the definition of *Widget* is not the same as the one defined by W3C (of which we refered to in section 2.4.3.1), but one, that takes only into account the lightweight and single purpose aspects of this concept.

The packaging and distribution aspects of this concept are left for the developer to comply with, which are only valid if the application gets to be executed in a specific W3C based widget engine, which is in fact, perfectly possible because GWT can compile one or more of our widgets into a single Web application that can be properly packed for a specific widget engine.

Although our widgets result in compiled and packaged single purposed JavaScript and HTML code, the real source of our Widgets is Java based source code with a properly defined structure that follows the lines of object-oriented design (see Fig.4.11).

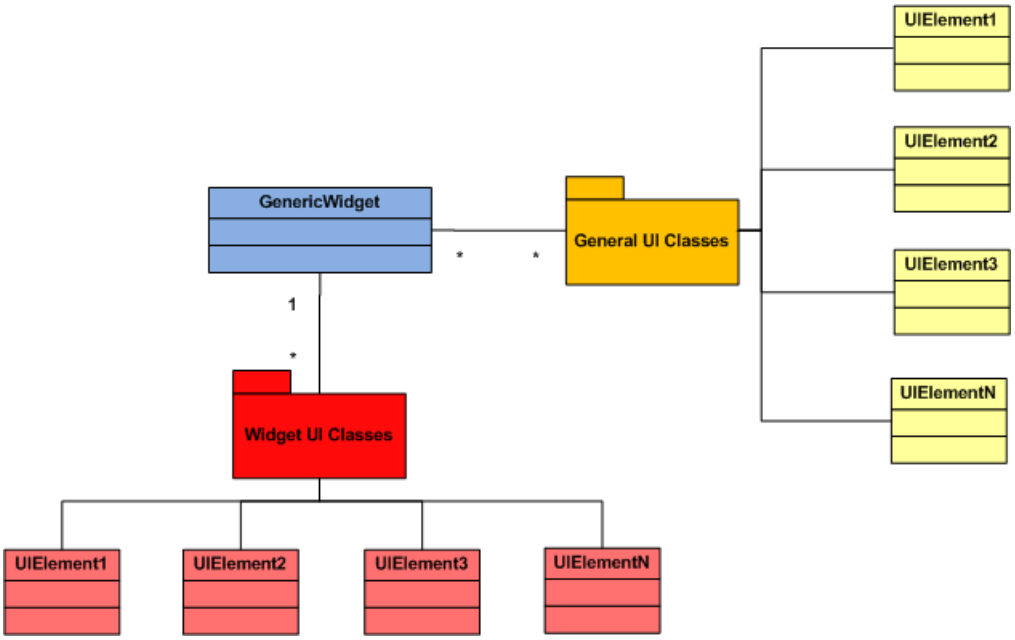


Figure 4.11: Generic Widget Java Object Structure

A widget of our application is then composed by a main *Generic Widget Class* which holds all single purposed application logic. This component also uses GWT JSNI in order to use the JavaScript core of our Middleware component. A main Widget class makes use of Generic User Interface objects, defined by *General UI Classes* which are shared by other widgets and need to be included in the compilation phase in order for the Widget to properly work. Additionally, a widget also has its own *Widget UI Classes* which define User Interface objects that are only instantiated in the widget itself and must not be shared by other widgets.

This object structure is used in all of our widgets, which are of two types: **Core Widgets** or **Resource Widgets**.

Core Widgets

Serve as our groupware application support components, that deliver the functionality the user needs in order to engage in collaborative sessions with other users. This includes functions like account, group and session management; contact synchronisation and application navigation.

Core Widgets essentially follow the generic implementation design depicted in Fig.4.11.

Resource Widgets

Are basically, application widgets that integrate with specific collaborative resources served by PUC's managed resource engine servers. Actually, these are the ones that actually provide real collaborative functionality like file sharing, chat or audio and video conferences.

Resource Widgets are instantiated and managed by the Session Widget itself. This instantiation process depends on the participation data of the device that the user currently uses to access the application. This data is aggregated in the session data that the device downloads from the PUC Gateway when a session is opened. Only Resource Widgets that are supported by the accessing device (according to the device's technical capabilities), get to be presented and usable for the user.

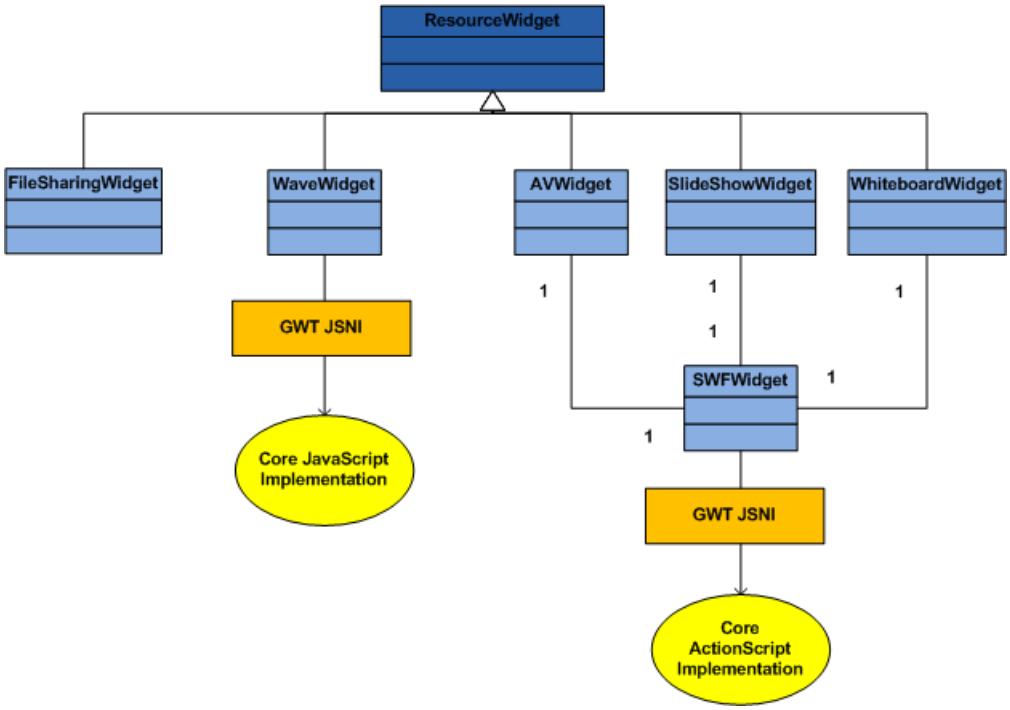


Figure 4.12: Resource Widgets Implementation Model

In our work's scope, we have fully developed a Wave Chat Widget, that provides chat based communication through the Google Wave Federation[46] Protocol and a JavaScript based File Sharing Widget that communicates with a remote file repository managed by PUC.

[46]Google Wave Federation Protocol - http://www.waveprotocol.org/

4.5 Chapter Summary

In this chapter, we addressed the challenges and relevant details of our implementation phase, in order to know how we turned our architectural designs into software implementations that fulfill our objectives.

We were able to leverage a significant set of supporting technologies (see section 4.1), in both server and client-side implementations.

The JBoss Application Server was used in order to deploy our PUC Gateway system component and integrate it with the PUC Platform itself (see section 4.2). The most relevant challenges that were faced during the development of this component, were the deployment of the Gateway itself according to the Java EE specification (section 4.2.1.2) and interface registrations in the Jabsorb's JSON-RPC bridge (section 4.2.1.3) in order to expose *Server User Agent Interfaces* to accessing mobile clients, and the implementation of a *Device Manager* component in PUC Gateway, that is responsible to deliver collaborative resources according to the accessing device's capabilities and XML based device profiles that get loaded at deploy time.

The implementation of the Mobile Terminal Middleware also had its share of relevant development challenges, which comprised the synchronisation of the mobile device's internal contact list with PUC, by exploiting the PhoneGap Contacts API (section 4.3.1), and a proper usage of the CometD implementation of the Bayeux specification, in order to build a Event Broker in our middleware layer, that totally abstracts the Bayeux protocol to the application developer who only needs to register any of his/her application's callback routines as listeners of a specific event that complies with our Event specification for collaborative applications (refer to section 4.3.2).

Finally, in section 4.4 we presented the design of our Mobi-Collab prototype application, which is totally developed using the Google Web Toolkit, which permits developers to create complex Web applications with a Java based source. We succeeded in leveraging this technology, by elaborating application designs for application widgets that exploit the advantages of object oriented design, concerning code reuse and extensibility.

5 Evaluation

"All truths are easy to understand once they are discovered; the point is to discover them."
– Galileo Galilei, Italian natural Philosopher, Astronomer and Mathematician, 1564-1642

We are now in the evaluation phase of our solution which has been built in hopes of ultimately being a usable and commercially viable solution that aims to be deployed in PT Inovação's context. Here, we will present and analyse our test results and try to make a realistic evaluation, considering this ultimate objective.

That being said, all of our developed components were subject of assessment through pre-established **qualitative** criteria and **quantitative** metrics.

Concerning our **server-side implementation** (PUC Gateway component and its integration with PUC), its conception and development came as a necessity, due to the communication and capability handling of restricted mobile devices not being possible just by using the PUC Platform directly. We have developed the Gateway component with the purpose of fulfilling this issue and cause the minimal overhead over PUC's performance, which is, considering that it is still on a development phase, lacking optimisation.

In our **client-side implementation** (terminal middleware and application prototype) our conception and development phase aimed ultimately for the construction of a common framework that would be mainly oriented for the development of collaborative applications in mobile devices. Here, performance evaluation is not our only concern because we need to satisfy both users, and application developers.

For the assessment of our server-side implementation, the reader can refer to section 5.1. As for the assessment of our the client-side implementation the reader should refer to section 5.2. Both assessments have been based on the above mentioned quantitative criteria and qualitative metrics.

Along the course of this chapter, we will be presenting results that were obtained during the validation phase of this project, and evaluate them according to our proposed metrics. Here, we will present the graphical representations that we find most relevant in order to support our analysis. As a reference we also present additional test results that the reader shall find in this document's apendix, under the chapter "Additional Test Results" (A), that strengthens the reached conclusions.

5.1 Server-Side Implementation Assessment

Our Server-Side implementation evaluation was mainly focused on our PUC Gateway and its integration with PUC Platform's API. In order to get a better understanding of the components here evaluated, we encourage the reader to review section 3.2 and look at Fig.3.2 which presents an overview of our system architecture. The Broker User Agent server that is represented in this figure was not a target of evaluation because its development and validation is not in this work's scope.

5.1.1 Objectives

The Gateway acts only as mediator component, it's execution is purely memory-driven with no data persistence at all. Tasks include object conversions, device profile loading and request forwarding.

We need to measure execution times of PUC with no Gateway at all. Additionally, we also need to know if memory consumption scales adequately in order for the system not to crash due to lack of available memory in the Java Heap Space.

In section 5.1.2 we present our elaborated testing environment for this particular assessment. Next, the reader shall refer to the different test execution setups that we previously defined in order to get proper results that meet our objectives, in section 5.1.3. As for the actual obtained results and respective analysis, the reader shall find them in section 5.1.4.

5.1.2 Testing Environment

In order to really evaluate the scalability of our server-side components we would need a testing environment composed by hundreds or even thousands of clients that would simultaneously make requests to our Gateway, or, Gateways! Unfortunately, that was not possible and our testing infrastructure was very limited.

Regardless, we tried to best accommodate with our limitations and focused on building a test infrastructure that could at least show us how our system would scale in face of multiple and simultaneous requests with variable data loads.

Our testing environment is depicted in Fig.5.1:

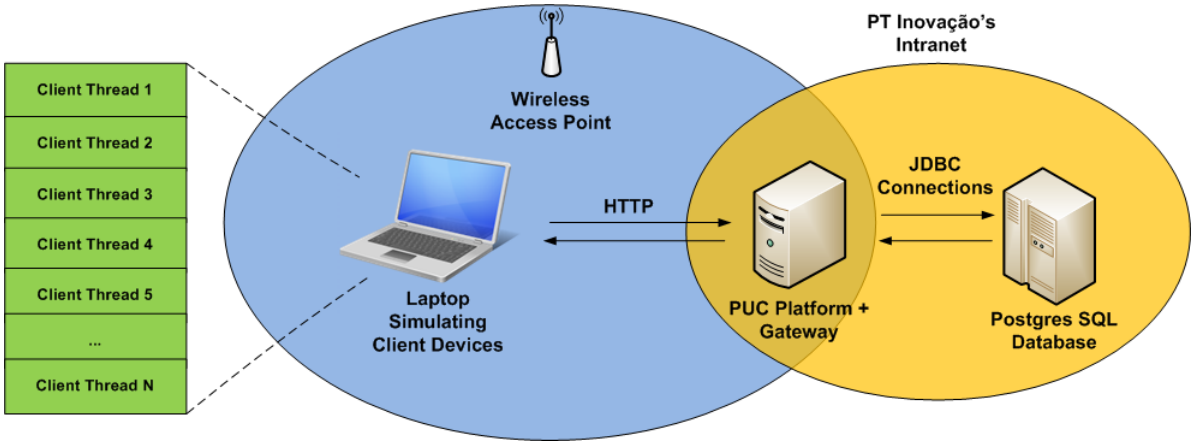


Figure 5.1: Testing Environment for The Server-Side Implementation

Both the PUC Platform and PUC Gateway were deployed on a single test machine. The PUC Gateway component is binded to an address which belongs to a publicly accessible wireless network in order to provide access to mobile clients over a wireless infrastructure. PUC itself, is then binded to a private address which belongs to PT Inovação's Intranet and directly communicates through JDBC, with a PostgreSQL server which resides on the same network.

In order to simulate multiple and concurrent mobile client accesses, we developed a Java based application that creates a set of client threads that concurrently execute specific requests using our Gateway's public JSON-RPC interfaces through HTTP. Each thread executes sets of functions which are grouped by specific functionality.

Test Machines

Considering the technical specifications of our test machines, we had the following:

Client Machine (Client Thread Host)

— 1.73 GHz Intel® Pentium® M Processor

— Level 2 cache with 2MB

— 1GB of main DDR memory @333Mhz

— Wi-Fi ready, equipped with a Intel® Pro/Wireless 2200 802.11b/g Wireless LAN card

Server Machine (PUC + PUC Gateway)

— Intel® Core 2 Duo CPU E4400 @2.00 GHz

— Level 2 cache with 2MB

— 2GB of main DDR memory @333Mhz

5.1.3 Test Setups

We prepared variable test setups that are defined by the **number of client threads in simultaneous execution** and the **quantity of aggregated data per request**.

Concerning the number of clients, we have established boundary values that go from **1**, to **10** and **100** clients. Our client machine's hardware supported the execution of up to 500 simultaneous client threads. However we failed to complete the execution of the test suite with this number of clients due to the lack of performance over concurrent database accesses of PUC's own database which would fail the execution of multiple queries when the number of simultaneous connections to the database exhausted the set upped limit on the database connection pool. Further optimisation of the database may resolve this issue, as such we decided to establish 100 as our maximum number of clients in our test setups.

Function execution order is different for each thread. Only the data load for each request by each thread gets maintained over an execution that is only over when all threads finish their work. This helps us to see how the variation in object size and number instantiations, impacts the overall performance of the requests.

Therefore, we have defined three variations in data load: **Low**, **Medium** and **High**.

All clients execute simultaneously, specific function groups with the corresponding data loads, as depicted in the following table:

	Low Data Load	**Medium Data Load**	**High Data Load**
Account Creation	* 1 account * 4 to 8 CommAddresses per user * strings with 8 chars	* 3 accounts * 6 to 10 CommAddresses per user * strings with 12 chars	* 5 accounts * 8 to 12 CommAddresses per user * strings with 15 chars
External User Creation	* 4 users * 2 to 4 CommAddresses per user * strings with 8 chars	* 8 users * 6 to 12 CommAddresses per user * strings with 12 chars	* 16 users * 12 to 24 CommAddresses per user * strings with 15 chars
Group Creation	* 2 groups with all users each * strings with 10 chars	* 3 groups with all users each * strings with 15 char	* 4 groups with all users each * strings with 20 char
Conversation Creation	* 2 conversations per group * strings with 10 chars	*3 conversations per group * strings with 15 char	*4 conversations per group * strings with 20 char
Session Creation & Setup	* 1 session per conversation group	* 1 session per conversation group	* 2 sessions per conversation group
Device Participation Loading	* 2 session loadings	* 3 session loadings	* 8 session loadings

Table 5.1: Data Load Variations for Client Thread Executions

Our criteria over the chosen data load values are based on utilisation patterns that we perceive as adequate, considering our application's capabilities. For low data load values we have object sizes that resemble the most foreseeable executions for all clients of our application. The medium data load variation imposes a respectable amount of data in each request, and corresponds to the average utilisation of our application for all accessing users. The high variation is our maximum defined limit and is one that goes way beyond normal utilisation. We even tried to stretch the variation even more, with an extreme level of utilisation, that we failed to accomplish due to the already mentioned lack of performance by PUC's database. However, we feel that the achieved test executions result in a proper analysis of our system's performance.

5.1.4 Results

All of the hereby presented results of our server-side implementation were obtained through a monitored execution of the PUC and PUC Gateway using the JBoss Tools[47] plugin with the Eclipse IDE, both in execution time and in memory consumption measurement modes.

5.1.4.1 Execution Time Tests

We have grouped our execution time result set, which corresponds to the execution of our test suite in accordance with our test setup (refer back to 5.1.3), in three functional groups:

— **Conversation Loading:** read-centric operations that ultimately result in the loading of all available conversations for a specific user. Here, object conversion is one-sided only, that is, objects are converted from Java to JSON only.

— **Group Creation:** necessary operations to create groups users to be included in conversations. Comprises contact synchronisation and group loading. This one is heavy on two-sided object conversion, both JAVA to JSON and JSON to JAVA. There is not much logic apart form object conversion here, so, this serves to acknowledge the impact on a two-sided object conversion centric execution.

— **Device Participation Loading:** includes the necessary operations in order for clients to properly load collaborative resources according to specifications of the accessing devices. This function set is the one with the most additional logic beyond object conversion.

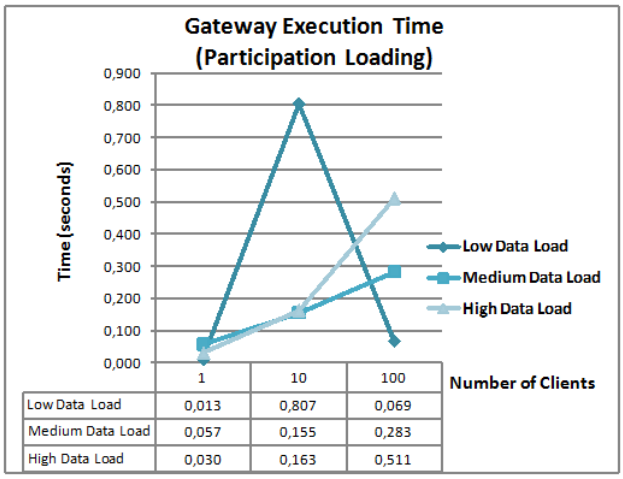
Result Analysis

First, we look how PUC Gateway performs solely in the execution of the respective functional groups.

All of the measured times are an average of all individual request times by each client. Immediately, we observe the higher impact in execution time of higher data loads versus the number of accessing clients. The only functional set that less suffers from the increase in data load is the Participation Loading Set (see Fig.5.2). That is because these operations have the least object conversion logic, being ultimately unaffected by the increase in object sizes (see Fig.A.6). In contrast, Conversation Loading and Group Creation operations suffer more, because they are heavy in object conversion logic (see Fig.A.5 and A.7) and as we can observe by the Group Creation (see Fig.5.3), object conversion tends to take too much time when the number of accessing clients increase, specially in High data loads, reaching an average of 8 seconds.

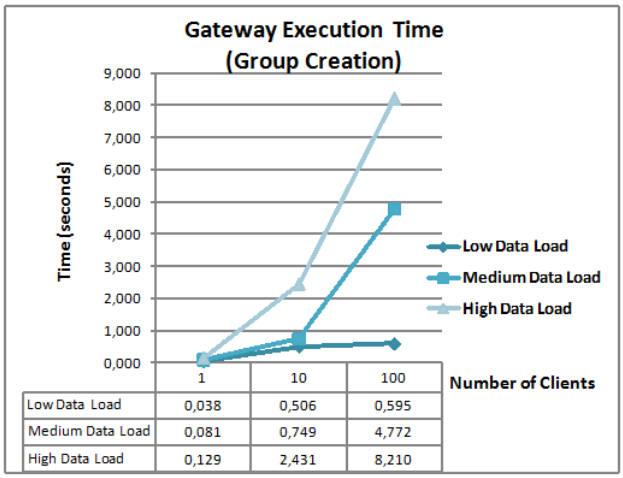
But these are only times taken in PUC Gateway's execution only. The real request times, as perceived by accessing clients are significantly higher.

[47] JBoss Tools Homepage - http://www.jboss.org/tools.html



	1	10	100
Low Data Load	0,013	0,807	0,069
Medium Data Load	0,057	0,155	0,283
High Data Load	0,030	0,163	0,511

Figure 5.2: PUC Gateway-Only Execution Time Results (on Average) for Session Participation Loading Operations

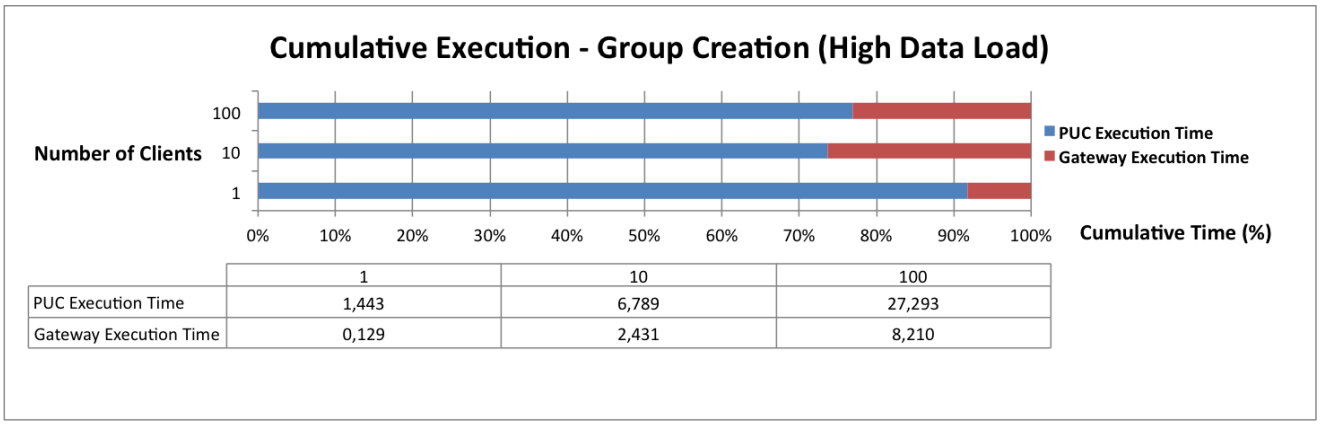


	1	10	100
Low Data Load	0,038	0,506	0,595
Medium Data Load	0,081	0,749	4,772
High Data Load	0,129	2,431	8,210

Figure 5.3: PUC Gateway-Only Execution Time Results (on Average) for Group Creation Operations

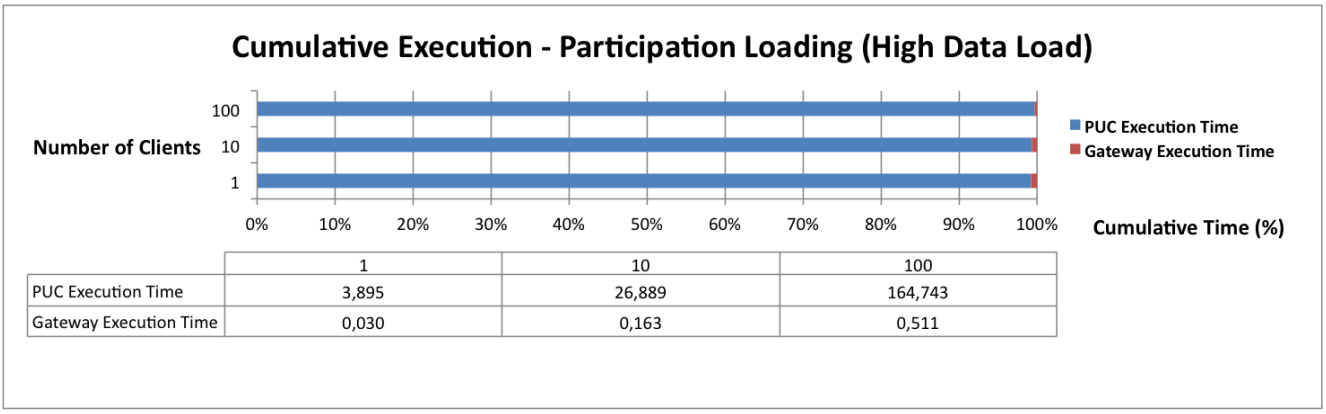
First thing to note is that, the execution times under PUC utterly surpass the Gateway times. PUC accesses a centralised database. Obviously, as the number of clients gets higher, the concurrent accesses to the database largely impact PUC's overall performance. With 100 clients and high Data Load, a single conversation load takes 140 seconds, which is not acceptable in order to provide a good user experience. We have also obtained an abnormal result in the Participation Loading Set for 10 clients in Low Data Load that may be justified by an unexpected and intensive computation on the machine that hosts the Gateway during the execution of the respective test, considering that, all the other results are in conformity to what was generally obtained.

As PUC's optimisation is out of this work's scope we need to assert if the Gateway itself impacts overall performance and needs to be optimised. By observing the cumulative execution graphs, the introduced overhead by the Gateway in PUC's execution times, is for Conversation and Participation Loading (see Fig.5.5) , almost insignificant, hardly reaching 5% of PUC's execution time even in High Data Load. However, in Group Creation it reached much higher values (arround 30% with 10 clients in all Data Loads). Group Creation (Fig.5.4) are also the operations that take most time and considering it's execution distribution purely based in two-sided object conversion, we can then conclude that JSON marshalling and unmarshalling of heavily sized objects, do make an impact in the system's overall



	1	10	100
PUC Execution Time	1,443	6,789	27,293
Gateway Execution Time	0,129	2,431	8,210

Figure 5.4: Cumulative Execution Time Results (on Average) for Group Creation Operations (individual execution times in seconds)



	1	10	100
PUC Execution Time	3,895	26,889	164,743
Gateway Execution Time	0,030	0,163	0,511

Figure 5.5: Cumulative Execution Time Results (on Average) for Participation Loading Operations (individual execution times in seconds)

performance. Nevertheless, the Gateway execution times never surpass PUC's execution times (they do not even generally reach a 50% overhead). If we assume an increase in PUC's performance that sets its execution times on par with the Gateway times, considering Group Creation, we would get a response time that would be around 16 seconds in the worst-case scenario. Considering the asynchronous nature of the operation, this could be acceptable.

Bottom line, the Gateway's execution times are excellent in operations which are light in object conversion logic. But, tend to increase significantly in operations heavily based in object conversion, which is basically the majority of PUC Gateway's operations. Although values get high (3 to 8 seconds) with multiple clients and higher data loads, values remain in an acceptable range for asynchronous operations. The introduced overhead is also acceptable and complies with our objective in not making the Gateway a significant overhead in the system's performance. Still, there is room for improvement. If in the future, we successfully optimise the JSON marshalling and unmarshalling tools we could reduce the overhead in object conversion even more, and provide a Gateway that has negligible impact in the platform. As of today, we have one that has small impact, that in a worst-case scenario translates to an introduced overhead around 30% of PUC's execution.

5.1.4.2 Memory Consumption Tests

For our memory consumption tests we used the same test setups as in the execution time tests. Here, the obtained results are basically object sizes grouped by object types or their respective class. In this regard, both PUC Full Data Model and PUC Gateway's UA Data Model, were subject of a comparative evaluation concerning their memory allocation sizes with the latter being also subject of a Garbage Collection analysis.

Result Analysis

Our main objective with the definition of the UA Data Model, was to create a simpler data model that would be bearable for mobile devices. Evidently, allocated UA Data objects must occupy lesser memory. We will first look into our obtained results and confirm if this is true. The Data Model Comparison results that are not presented here, can be found in the apendix section A.4.

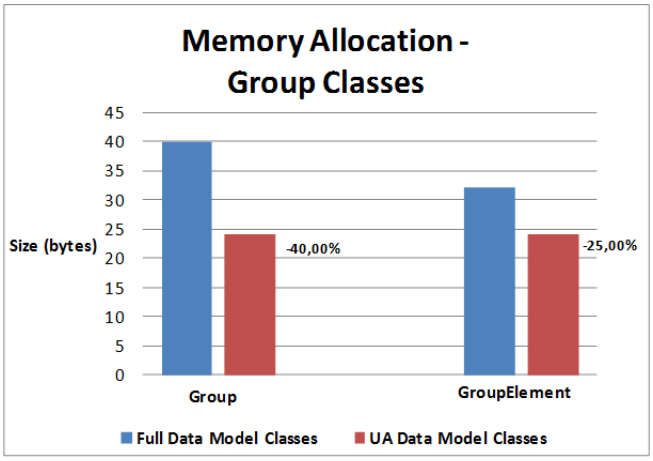


Figure 5.6: Allocated Size (in bytes) per object in the Java Heap Space for both Full and UA Data Models (Group classes)

As the results show, the objective of creating a smaller data model was successfully achieved. But, excluding the Conversations object classes, which achieved 25% reduction on average and Group classes which achieved 32,5% reduction (see Fig.5.6), other classes, such as User and Resource classes have only achieved reduction values around 11% (Fig.5.7). User objects are even the larger ones and occupy most of the available memory, considering the high number of contact synchronisations that may happen on the system when the number of accessing clients gets really high. One could say that this is far from acceptable. Truth is that, it is entirely possible to reduce object sizes of User and Resource object types. However, we would loose functionality, and to keep it, we would need make the client application request the respective information when needed, instead of downloading all the information when the object gets passed to the client in its first access. Object allocation would be more dynamic, however this would increase the creation of new user requests and the introduced overhead by new requests would not be compensatory for smaller objects. We opted to not reduce the object size that much and give the user all the information on its first access in order to avoid later spawning of multiple requests to obtain that same information.

The reader may note that the presented object sizes do not correspond to the real object size, but only to the allocated memory size on the Java Heap Space. Objects in both data model are heavily string based, which in Java is not a native type and it does not get allocated directly in the object. The JBoss Tools Profiler Monitor fails in giving us the real object sizes, but the strings are the same on a full object as in a UA data model, therefore, reduction percentages still apply if we consider a string value that is constant.

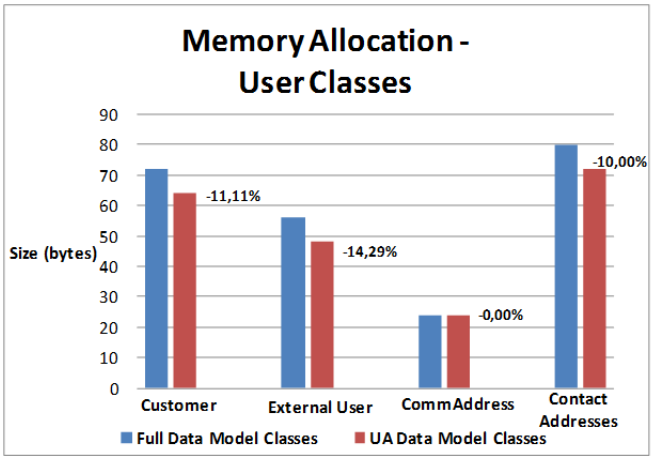


Figure 5.7: Allocated Size (in bytes) per object in the Java Heap Space for both Full and UA Data Models (User classes)

The utilised tools permitted a detailed analysis over the Java Heap Space during test execution, which we present in section A.5. This allowed us to observe how many Garbage Collection Cycles an object survives during test execution (Fig.A.10) and the total allocated memory per object type (Fig.A.11).

This serves to assert if there is the possibility of the Gateway exhausting the available Java Heap Space when object sizes and the number of accessing clients increases. By observing the obtained results, we conclude that the Java default Garbage Collection Mechanism does a good job of removing objects that get the most instantiations. We can see for example, that the CommAddress and ContactAddresses objects, which occupy more memory over the test course are objects that have little to no life during the execution. In contrast, objects less instantiated like DeviceResourceConnection or DeviceParticipation get more time to be eliminated.

During test execution, 523 Garbage Collection Cycles have passed with 10 clients, and 1156 with 100 clients. Every object age is well below these values. Customer objects get to be older ones, and in average, they lived 30 GCCs with 100 clients. As such, our test setup was insufficient in order to find the limit at which the Gateway would fail by lack of available Java Heap Space. The way how objects are rapidly removed shows that the Gateway execution is purely memory driven, where instantiations correspond in majority, to auxiliary variables that are used in object conversion logic. As a result the Gateway can easily hold the maximum load of our test suite. If we take the obtained results into account and consider a Java Heap with default size minus the total allocated memory during our test (128MB - 2,66MB) , it would need around 2 021 977 simultaneous session loadings in order to exhaust the Java Heap Space. Of course, this is just a rough estimate, but we can safely assume that, as it is right now, in regards to memory consumption the Gateway can surely hold up to 10 000 simultaneous client accesses without exhausting the Java Heap Space. This is a safe estimate because our previous calculation does not take into account the real size nor all of the instantiated objects. As we are unable to run our test suite with more than 100 clients, we cannot provide an exact estimate.

With the Gateway's memory being solely used in the instantiation of auxiliary variables during object conversion, the Java Garbage Collection mechanism does a great job, managing the memory consumption of the Gateway process according to our needs and we see no need for further optimisation in this department.

5.2 Client-Side Implementation Assessment

The assessment of the client-side implementation, targets both the core middleware layer and the developed prototype application that uses it.

5.2.1 Objectives

In contrast with PUC Gateway, which is a server-side component only that gets properly deployed just once and handles all requests that arrive automatically, the client-side components aim to be ultimately used by users and developers of collaborative applications. A quantitative based only approach in the evaluation is simply not enough. We need to also evaluate the components qualitatively.

In respect to **user satisfaction**, we need to evaluate:

— The application's general responsiveness and usability.

— If battery consumption remains at acceptable levels not turning the mobile device unusable in short time.

— If the quantity of uploaded and downloaded data by the application remains at acceptable levels considering limited data plans over 3G networks.

And In respect to **developer satisfaction**, we need to evaluate the portability of both the middleware layer and the application.

In the next sections we present the evaluation phases that our implementation was subject to in order to accomplish the aforementioned objectives. One based on **quantitative metrics** (section 5.2.4) and another based on **qualitative metrics** (section 5.2.3).

5.2.2 Testing Environment

The testing environment for the assessment of the client-side implementation follows the same network topology as depicted in Fig.5.1, except for the substitution of the laptop that runs Java client threads by a real mobile device running one instance of the Mobi-Collab Prototype App in the Android operating system over a Wi-Fi connection.

The utilised mobile device is the HTC Hero, depicted in Fig.C.1 and it has the following specifications:

— Qualcomm MSM7200A, 528 MHz Processor;

— Android Operating System;

— 288MB of RAM and a 512MB ROM;

— HSPA/WCDMA network capable with up to 2 Mbps up-link and 7.2 Mbps down-link speeds;

— Wi-Fi: IEEE 802.11 b/g capable.

5.2.3 Qualitative Evaluation

In our qualitative-metric based evaluation we will start by validating if the developed application succeeds in complying with earlier proposed functional requirements (section 5.2.3.1).

Both the application and the supporting middleware layer were subjected to an exhaustive analysis concerning the portability of its components across multiple mobile operating systems. This analysis was supported by individual execution of all developed functionality in each target operating system with a proper identification of encountered problems and needed coded changes in order to make the application work as in the original build. This will be presented in section 5.2.3.3.

As for an analysis of the developed application's final presentation and usability, the reader shall refer to the apendix chapter B. Additional photos of the testing devices that were used in this evaluation phase, can be found in the apendix chapter C.

5.2.3.1 Overall Functional Requirements Accomplishment

Soon in this work's conception phase, we identified a set of functional requirements for our Mobi-Collab Prototype Application that would comprise common groupware functionality (see section 1.4).

The objective to engage users in collaborative work through mobile devices has been successfully achieved. However, the accomplishment of the entire list was not possible.

In the group of features that **we could not implement** in the course of our work, we highlight the management of user roles and policies over session management (PUC does not support this, it only has the role of moderator and normal user); the enriched user profiles and presence information and contact synchronisation with external services.

Concerning the offline work support, the developed middleware layer supports data persistence, but, it does not support data consistency control mechanisms, nor does PUC itself as a platform that is still in development. As there is no support in the platform itself, this feature had to be left out.

With respect to the collaborative resources themselves, we have successfully integrated our application with Adobe Flash based widgets that have been developed in-house and implement the audio-video conference, slideshow and whiteboard features (see section 5.2.3.2).

5.2.3.2 Integration With Adobe Flash Based Resources

We have also developed an interface that integrates our GWT based Widgets with some in-house developed Adobe Flash based Widgets, namely, a Audio/Video Conference Widget; Slideshow Widget and a Whiteboard Widget (see Fig.C.2).

Some widgets, as is the case with the Wave Chat and Flash based widgets (see Fig.B.4), need to communicate with proprietary protocols which implementation resides in external libraries not based in GWT Java code.

In the case of the Wave Chat Widget, our GWT widget makes use of GWT's JSNI to communicate with a core JavaScript source that implements the communication with Google Wave Resource Server.

Our own developed interface with Adobe Flash based resources comes in three distinct GWT widgets that share in common a third party Widget: the gwt2swf[48] SWF Widget. This Widget makes the communication bridge of GWT code with ActionScript code possible and is what makes the instantiation and management of Flash based content inside our own Widgets possible.

A serious limitation with the Flash based Widgets is that they are developed in ActionScript 3.0 which requires Adobe Flash Lite 4 or Flash 10.1 compliant devices in order to get executed.

We were able to implement in-widget interaction with Flash based content, that is ActionScript code being executed inside our Session GWT widget in a Flash 10.1 based device, the Google Nexus One. But we have failed to implement it in a Flash Lite 4 compliant device, the HTC Hero (see Fig.C.3).

The reason is that it is technically impossible, to run the Flash Lite 4 plugin in a Android's WebView, which is the component where all our GWT Widgets and middleware layer gets executed. Nevertheless, we have developed an alternate interaction method which opens a browser window from the session widget itself when a user connects with flash based collaborative resources. The downside here is that the user looses the functionality provided by the Session Widget and is left only with the Flash based Widget and has to manually get back to the application which is not entirely acceptable in terms of usability.

As Flash compliant devices, we only had these two Android Smartphone models. Implementation of Flash content integration with our Widgets in other operating systems is still yet to accomplish.

[48] gwt2swf Flash/Flex Widget for GWT - http://code.google.com/p/gwt2swf/

5.2.3.3 Portability

Our Mobi-Collab prototype application was first built and tested in the Android Operating System. In order to assert our promise of delivering a framework with the maximum level of portability, we developed additional builds of our app. One for each target operating system: iPhoneOS, Symbian S60 5th Edition; Blackberry OS 5.0 and Windows Mobile 6.5.

On a first basis, we maintained the same Web code for each build and properly switched the native bridging code (Phonegap Libraries/Widget Engines) in order for the Web code to get properly compiled in the target operating system native languages and executed in the respective emulators of each SDK. Once incompatibilities were being found over the application's execution on each platform, we registered them, and if they could be solved, we would solve them and would register the respective changes in the Web code. For those issues that required drastic changes, we made an estimate of the necessary changes and their impact concerning the difficulty in developing new code.

We have achieved an excellent level of portability in operating systems in which the Web Rendering Engine is based on WebKit. This is due to their very similar implementations and excellent compatibility with the majority of currently available Web standards. In contrast, operating systems with other Web engines failed to execute our application without issues. Some of them quite severe.

The Flash based resource widgets remained untested in operating systems other than Android due to the inability of running Flash based content on all emulators.

Operating Systems With WebKit (Android, iPhone OS, Symbian S60 5th edition)

With WebKit based platforms we achieved a very high level of portability, which resulted in changes of 9 lines of code in the iPhone OS and 13 lines of code in Symbian S60 5th edition. Better yet, the changes targeted application components only, and with very low impact in development as they were originated by minor CSS issues and minor differences in the PhoneGap Contacts API. As a result, we managed to create the two ports for the iPhone and Symbian platforms, in a couple of hours.

Apart from the failed execution of Flash content due to incompatibility in both targeted systems, and lack of Local Storage support in Symbian S60, all the other functions that had been developed in the original Android version, were executed flawlessly with just minor differences in the user interface presentation (see Fig.5.8).

BlackBerry OS 5.0 and Windows Mobile 6.5

The experience with these systems (whose proprietary browser engines differ in a significant way concerning compatibility with most Web standards in comparison with WebKit), was considerably worse.

We managed to get a BlackBerry port up and running until the Session Widget where the real collaborative functionality takes place, but we failed in getting this latter widget to work properly in due time. This was due to a myriad of issues caused by the technical limitations of the platform's browser engine. The engine was incompatible with the CometD support library rendering the build incapable of supporting events, and even, the Wave Chat widget. Even the JSON-RPC third party library needed changes in order to work properly. We managed to make it work but it resulted in changes over 30 code lines, and as the code was not developed by us, it ultimately resulted in an entire day of test and debug work.

The application code also had to be changed in many user interface elements that revealed to be incompatible. Even with changes that comprised almost one hundred code lines, the end result still left much to be desired (see Fig.5.9).

In the Windows Mobile platform the experience was extremely worse. Here, the Mobi-Collab prototype app failed to even initialise. The IE6 Mobile engine revealed itself incompatible with GWT and

Figure 5.8: View of The Session Widget on WebKit based Operating Systems

as result none of our application's widgets even worked. Nevertheless, we successfully tested our Core User Agent library, and JSON-RPC calls to the Gateway services are still possible. But we could get any further, as developing an entirely new application, not GWT-based, would simply be not achievable in due time.

Portability Overview

In Table.5.2.3.3 we present an overview that describes our Portability evaluation.

Operating System	Portability Level	Code Line Changes	Problems Faced
Android 2.0, 2.1, 2.2	Full	none (original version)	none
Android 1.5, 1.6	Very High	none	Local Storage is technically not supported
iPhoneOs 3.2, 4.0	Very High	9 (application and CSS changes)	minor differences in Phonegap API; no Flash
Symbian S60 (5th Edition)	Very High	13 (application and CSS changes)	Local Storage is not supported
BlackBerry OS 5.0	Partial	84 (application); 8 (core); 30 (suport libraries);	Many UI Issues; incompatibility with CometD
Windows Mobile 6.5	Extremely Reduced	needs a totally redesigned application not GWT based	only JSON-RPC works

Table 5.2: Overview of The Portability of Our Implementation Across The Targeted Operating Systems

Considering the obtained results, we can assert that we are satisfied with the obtained results. The results are good, as we have successfully created fully functional builds for three out of five systems with just minor changes in the original build.

Our evaluation also shows that the portability levels of our implementation are directly influenced by the rendering and standard compatibility capacities of the hosting Web engines, and not by particular architectural differences of the different operating systems. This is natural, considering the Web based approach that we took in the conception and development of our components. Eventually, as Web engine implementations get better and get close to what is currently deployed in Desktop systems we may get fully functional builds in every operating system, ultimately resulting in the *write once, run everywhere* philosophy that we sought from the very beginning.

Figure 5.9: UI Issues and Incompatibilities of the BlackBerry Build

5.2.4 Quantitative Evaluation

In the quantitative-metric based evaluation, we start by analysing the performance of our client-side implementation (section 5.2.4.1) which comprises the responsiveness of the application and middleware layer, according to measurements of function completion times and response to certain events.

Next, in section 5.2.4.2, we also analyse the battery usage of the prototype app, through measurements made by the Android System Monitor during test executions according to different usage patterns.

All of the quantitative evaluation tests follow the testing environment described in section 5.2.2 with test setups that differ according to what is being analysed.

5.2.4.1 Overall Performance

The test setup for these performance analysis comprise a server-side database in a steady-state and the individual execution of each application's function in a real mobile device.

Time values were manually obtained after the execution of each function or after notification of the correspondent events, through the JavaScript Date Object value, before, and after function completion or event notification.

Result Analysis

We will first analyse how much time each function takes to be completed in mobile. We will also analyse the responsiveness of the application to events launched by the server-side platform.

Function Completion Times

Fig.5.10 shows the completion times for each application function. The graph distinguishes the time taken in the Core JavaScript libraries and the time taken until the user sees that the function has been

successfully executed in the application, in order to evaluate the introduced overhead by the application.

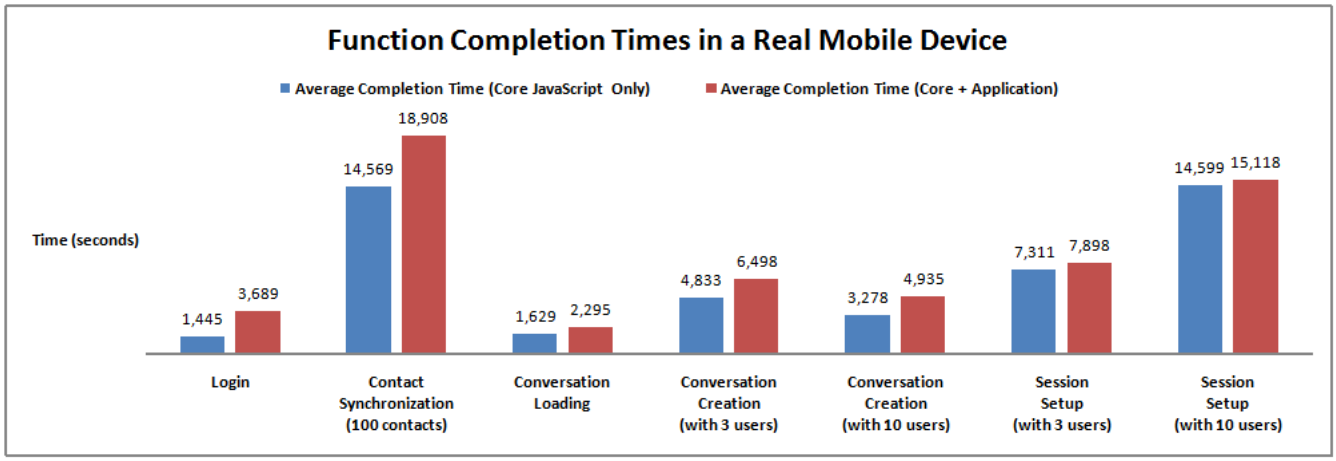


Figure 5.10: Measured Average Completion Times of Specific Functions in The Client-Side Implementation Executing in a Real Mobile Device

Functions that take considerable time are contact synchronisations and session setups. Fact that is understandable considering the good amount of contacts to be synchronised in our test device (100 contacts) and that session setups take considerable time in the platform itself, as resources need to be properly allocated and configure for all session users.

Although the obtained times in the core JavaScript are bearable for the asynchronous operations that they correspond to, they need some optimisation. If we want to engage in a conversation with another user urgently, we will have to wait around 30 seconds for that to happen, if we sum up times taken in contact synchronisation, conversation creation and session setup.

Considering the application's overhead, the function that really needs optimisation is the contact synchronisation. The lower performance is due to the great amount of user interface elements that get rendered on the screen.

Response to Events

Results for the response to particular application events are demonstrated in Fig.5.11.

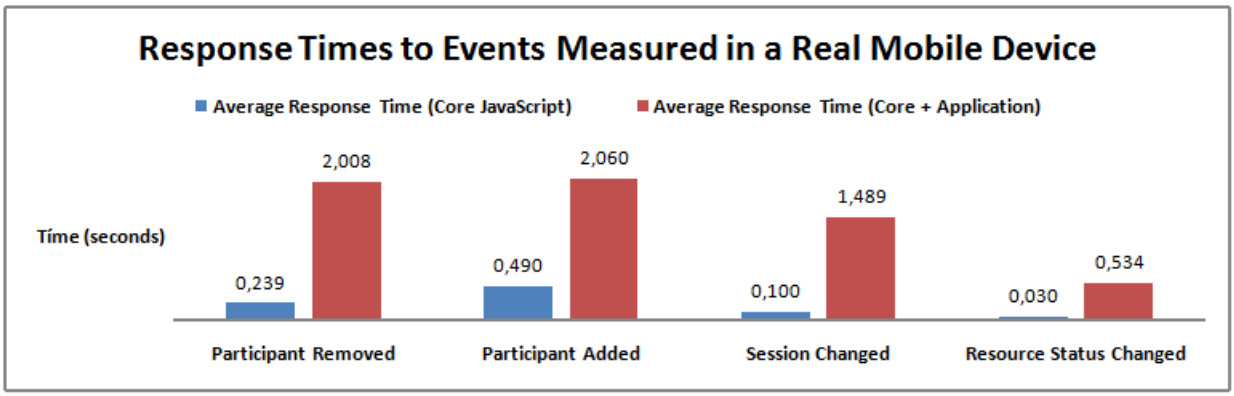


Figure 5.11: Measured Average Response Times To Specific Events in The Client-Side Implementation Executing in a Real Mobile Device

The CometD Broker User Agent does a remarkable job in delivering events with little to no delay, almost in real-time, as we can assert by observing the obtained response times in the Core JavaScript. However, the user is only informed of the event when the application notifies it and the downside is that, in comparison, the application introduces a large overhead. Events in the application take 1 to 2 seconds in being notified, which are still good values, considering that most Web applications that base event notification in standard poling techniques, set 5 seconds as the polling interval.

Such a large overhead that is introduced by the application is mostly due to the multiple software layers present even in the application layer itself, which is not a native application nor a Web application in raw JavaScript/HTML code, but, a GWT Web application. As GWT is not yet optimised for being executed in constrained mobile devices, applications run much slower, when compared with their desktop counterparts, even in mobile devices with respectable CPUs.

5.2.4.2 Battery Consumption

Offering collaborative functionality in mobile devices would not serve for nothing if the battery would be wasted in a short time. Battery consumption depends in many factors that go from the hardware, clever programming and different usage patterns by users, which can be more intense or not according to their intentions and user experience.

We took our implementation to the test in order to observe how it's battery usage would be according to three different usage profiles. A profile for a **Casual User**, a **Power User** and finally a **Hardcore User**.

For that we created test scripts, one for each profile, that automatically would execute a set functions according to the frequencies of the corresponding user profile and we set the total execution time to 1 hour and 25 minutes in all profiles.

The **Casual User** profile includes the following operations:

— 1 account every 30 minutes. Total: 2 accounts.

— 1 contact synchronisation every 12 minutes. Total: 7 synchronisations

— 1 new conversation every 8 minutes. Total: 10 conversations

— conversation loading every 4 minutes and 30 seconds. Total: 18 loads.

— 1 new message in a chat session every 1 minute and 25 seconds. Total: 59 messages

As for the **Power User**:

— 1 account every 10 minutes. Total: 8 accounts.

— 1 contact synchronisation every 6 minutes. Total: 14 synchronisations.

— 1 new conversation every 3 minutes and 20 seconds. Total: 25 conversations

— conversation loading every 1 minutes and 30 seconds. Total: 56 loads.

— 1 new message in a chat session every 32 seconds. Total: 156 messages.

Finally for the **Hardcore User**:

— 1 account every 5 minutes. Total: 16 accounts.

— 1 contact synchronisation every 2 minutes. Total: 42 synchronisations.

— 1 new conversation every 1 minutes and 10 seconds. Total: 72 conversations.

— conversation loading every 45 seconds. Total: 113 fetches.

— 1 new message in a chat session every 12 seconds. Total: 425 messages.

Result Analysis

We used the Android System Monitor in order to observe the percentage of battery that was being used by our application, in comparison with other applications running at the same time and the hardware components of the utilised mobile device.

Evidently, user profiles with an higher operation frequency detained more CPU Time, as we can assert by observing Table 5.2.4.2. This is naturally reflected in the battery usage percentages, albeit, the percentage being relatively small even in most intense usage pattern (19% in the Hardcore Profile in comparison with the battery usage of the device's display at 75%).

User Profile	Application's CPU Time
Casual User	**8m 10s**
Power User	**18m 28s**
Hardcore User	**31m 19s**

Table 5.3: Detained CPU-Time by The Prototype Application During Test Execution on Each User Profile

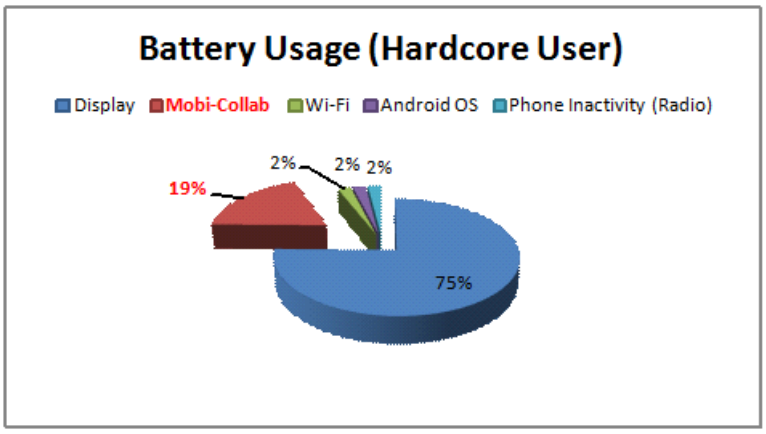


Figure 5.12: Percentage of Available Battery Energy That Was Used by The End-User Application During Test Execution (Hardcore User Profile)

Although battery usage percentages are small, the battery consumption percentages after test execution are significant. Fig.5.13 demonstrates battery consumption levels of 20% for the casual user, 25% for the power user and 40% for the hardcore user. Taking into account, the execution time of 1 hour and 25 minutes, we can estimate the approximate usage time of our application for each user profile with a fully charged battery (see Table.5.2.4.2).

User Profile	Approximate App Usage Time
Casual	7 hours
Power	5 hours and a half
Hardcore	3 hours and a half

Table 5.4: Estimate of Application Usage Time in Each User Profile Until Battery Gets Drained

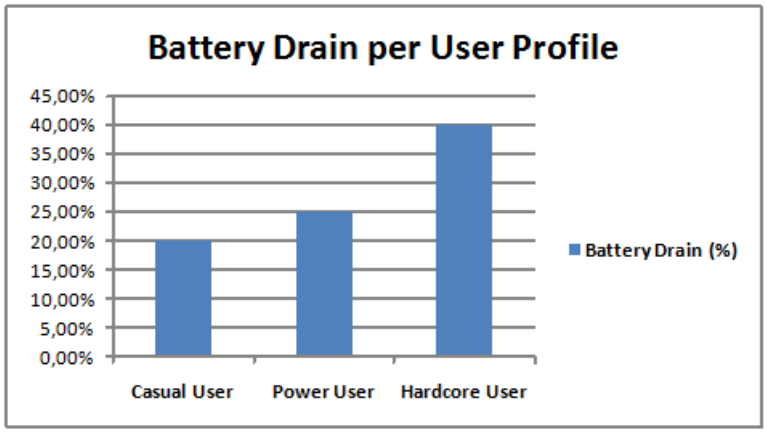


Figure 5.13: Percentage of Battery Energy drained by The Prototype Application During Test Execution on Each User Profile

The results are encouraging. Even in the hardcore user profile, the battery can last a considerable amount of time, considering that we are not expecting users to use our application in a mobile device not as long as they would use it in a desktop machine. The usage of a mobile device to collaborate is only expected on-the move, mainly in restrict environments or emergency situations when a given user will unlikely use it for more than a hour. Though, we must note that we did not test the application with a video-conference resource, as we were unable to test Flash based resources in emulators. In this situation, we should expect the battery consumption to rise significantly and it should be tested in the future.

5.2.4.3 Data Usage over Mobile Networks

We also made use of the Android System Monitor is order to obtain the amount of data transmitted over the wireless network. The obtained results are presented in Fig.5.14.

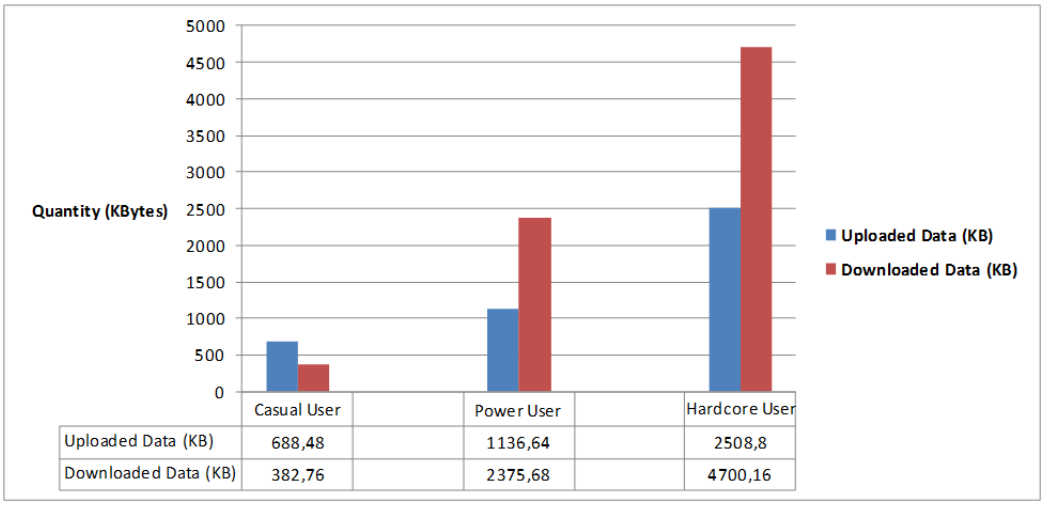


Figure 5.14: Amount of Transmitted Data By The Mobi-Collab Prototype App Over a Mobile Network on Each User Profile

The mobile network usage results are also encouraging. Assuming a very limited 3G dataplan that gives an hardcore user 15MB per day, he could still use the application for at least 2 hours which is more than enough for any collaborative context in a mobile device.

Fig.5.15 shows estimates of the JavaScript objects sizes in the data model of our application in a high data load situation. Observing the results, we can assert that we could still optimise even more the mobile network data usage, by reducing the object sizes of Group objects.

The obtained results show that our *User Agent Data Model* and the JSON message format really succeed in delivering lightweight objects for constrained devices that communicate through limited mobile networks. With even more optimisation in reducing the object sizes while maintaining similar frequency in calls to the Gateway we could certainly obtain even better results in the future.

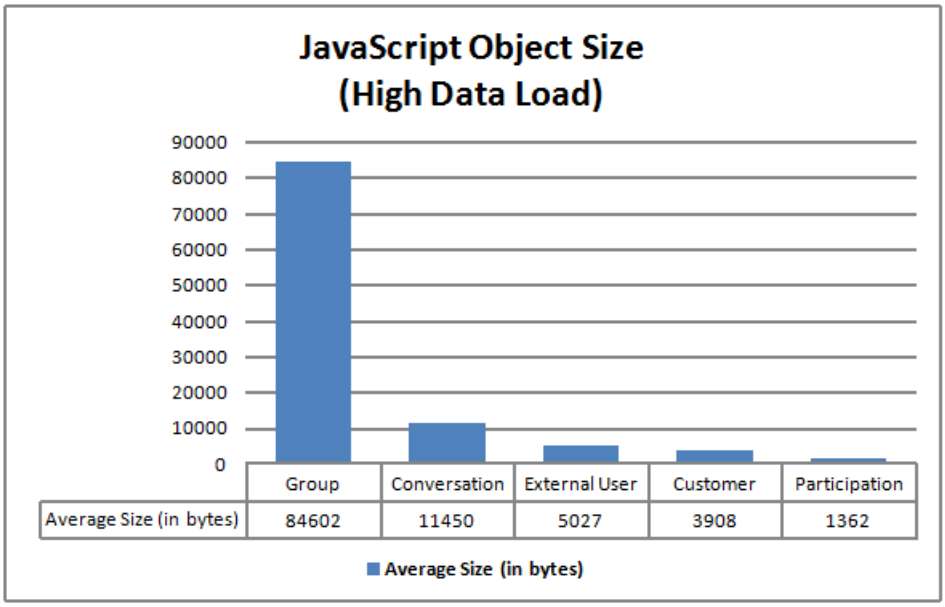


Figure 5.15: Average JavaScript Size in High Data Load

5.3 Chapter Summary

In this chapter, we performed an evaluation of our solution based in quantitative metrics for both the server-side (section 5.1) and client-side (section 5.2) implementations. We also evaluated our client-side implementation qualitatively in order to produce a solution that properly satisfies both users and application developers.

The evaluation of our server-side implementation focused in the PUC Gateway component and its integration with the PUC Platform. As the Gateway is basically a mediator component in the communication of the mobile device with the platform, we needed to measure the introduced overhead by the Gateway process and its effect in the cumulative execution of specific client requests according to different test setups with variable data load (section 5.1.4.1). The obtained results show that the Gateway has a small impact in the cumulative execution of the requests, introducing an average overhead of 30% of PUC's execution time, in the worst-case scenario. We also measured the memory consumption in the Gateway process (section 5.1.4.2), and we concluded that the Gateway's memory is being solely used in the instantiation of auxiliary variables during object conversion, therefore, the Java Garbage Collection mechanism with the default configuration does a great job of managing the memory consumption according to our needs, not needing any sort of optimisation.

Qualitatively, our client-side implementation implements most of our initially proposed functional requirements (section 5.2.3.1) excluding the management of user roles and policies; presence information; contact synchronisation with external services and data consistency control, due these requirements not being supported by the PUC Platform (which is still on development) in due time.

Concerning the portability evaluation (section 5.2.3.3), the results are very encouraging. We have successfully created fully functional builds for three out of five systems with just minor changes in the original build, and concluded that the portability levels of our implementation are directly influenced by the rendering and standard compatibility capacities of the hosting Web engines, and not by particular architectural differences of the different operating systems. We hope to eventually fulfil a *write once, run everywhere* philosophy when the mobile operating system Web engine implementations get better and get close to what is currently deployed in Desktop systems.

As for the quantitative evaluation of our client-side implementation one of our objectives was to measure the general responsiveness of our GWT application prototype concerning both request times (section 5.2.4.1) and response to events (section 5.2.4.1) and how times measured in-application compare to times taken in the middleware layer itself. Here, we concluded that the application generally provides a good user experience, but that it introduces a significant overhead (compared with the time taken in the middleware) in some cases. This is due to the application being developed with GWT, a tool that is not yet optimised for Web applications running in mobile devices resulting in multiple software layers that perform slower than a raw JavaScript implementation. We have also measured the battery consumption levels of our prototype application (section 5.2.4.2) and the transmitted data over a mobile network (section 5.2.4.3), and concluded that, even in an intense usage pattern, the application performs very well with battery consumption levels that permit the user to collaborate in a significant amount of time, and a quantity of transmitted data that copes well with restrict network traffic limits.

In the final chapter, we will look into our evaluation results and present our final conclusions, that will validate the accomplishment of our objectives, and will propose possible future work.

6 Conclusions

"Success is not final, failure is not fatal: it is the courage to continue that counts."
Winston Churchill – British Orator, Author and Prime Minister during World War II. 1874-1965

We have now reached the end of this document, and will present our final conclusions, by first, presenting a review on all the phases of this work and this document's chapters (in section 6.1).

Then, we will look back to what we done, and revise the objectives that were initially proposed in this document's introduction (refer to chapter 1), in section 6.2.

We end our conclusion by presenting possible future work in our solution that we have identified during the course of our work, in section 6.3.

6.1 Review

Our objectives led us in an elaboration of an extensive survey in topics that comprised the foundations of CSCW and groupware; mobile environments and mobile groupware, that we presented in chapter 2.

With this survey, we soon identified the different taxonomies in groupware time-space interactions; the different groupware system architectures and common challenges that we would face in developing a groupware solution. This was still not enough, as we proposed to deliver a mobile groupware solution. Therefore, we extended our research to the common issues in mobile environments; the intricacies of mobile groupware development and different middleware solutions that have been applied in this area.

This survey resulted in a requirement list that our middleware solution would need to comply with, such as: provision of a execution support layer; persistence support; adaptability; a distributed information base; extensibility and portability.

The requirement of portability was identified as a limitation in current mobile groupware solutions. As this is one limitation that we proposed to address, we also studied the features of currently available and most used mobile operating systems and the development approaches based in Mobile Web Development frameworks and Widgets.

This latter study had revealed quintessential for the conception of a system architecture (refer to chapter 3) that capitalised in the usage of supporting technologies and transport mechanisms widely used on the Web; and a implementation (refer to chapter 4), that aimed to maximise portability; code re-usability and extensibility in all of the designed internal components and integrations with external components, such as the PUC platform itself or the Bayeux Event Broker. With the latter one, not being initially foreseen.

All of the undertaken work definitely paid off, as the evaluation phase, which we presented in chapter 5, revealed very promising results, both performance-wise and qualitatively.

In the end, we feel that the proposed limitations, which comprise the portability, re-usability and extensibility of mobile groupware applications; and the contribution of delivering a cross-platform solution that satisfies both users and developers of mobile groupware applications, have been properly addressed.

6.2 Revised Objectives

We can assert that the development of a server-side component that integrates with the PUC Platform and mediates the communication of accessing mobile terminals to the platform's services (PUC Gateway) has been fully addressed and with excellent performance results, resulting in a mediator component that introduces small overhead over the platform which it integrates with.

Concerning the implementation of the mobile terminal middleware layer, we feel that it has been properly addressed. It resulted in a implementation that is extremely easy to port, in both middleware and application layers, in three out of five target operating systems. The development of our application prototype validated that our solution is a easy to use, re-usable and extensible solution for developers of mobile collaborative applications.

For the operating systems where the porting process has revealed more difficult, we hope that future Web browser engine implementations of the respective systems continue to improve, because, our portability evaluation shows that the portability of our solution is directly influenced by the rendering capabilities and Web standards compatibility of the underlying operating systems. This shows that, although our solution provides excellent portability levels compared to natively developed solutions, it depends more in third-party implementations, having the risk of not working properly if the underlying third-party implementations change overtime.

6.3 Future Work

During the course of this work, we have identified a set of functional requirements that we were not able to implement in due time, and desired properties that we would like to see implemented in future, such as:

— **Network Security Mechanisms in The PUC Gateway:** the current JSON-RPC implementation of the Jabsorb framework, does not yet support JSON message cyphering and user authentication mechanisms. Guaranteeing the confidentiality, integrity and authentication of the exchanged messages is a essential requirement for collaborative applications that require minimal security measures that are commonly found in today's applications.

— **Data Consistency During Offline Work:** Although our solution supports cross-platform data persistence, it does not offer any sort of data consistency mechanisms. It would be interesting to implement this feature in both the middleware layer and the PUC platform. Guaranteeing data consistency in disconnected operations is important to relieve users from troublesome experiences resulting from information in inconsistent states.

— **Additional Features for The Application Prototype:** some of the functional requirements that we proposed in the beginning were not possible to implement in due time. Therefore we would like to add additional features to our mobile groupware application prototype, such as: management of user roles and policies over session management; enriched user profiles and presence information and contact synchronisation with external Web services.

We hope that the gained knowledge base, during the course of this work, is sufficient for the future development of the above mentioned requirements and properties, and that our solution as it is today, becomes useful for future commercial developments in PT Inovação's context.

Bibliography

[1] APPELT, W. What groupware functionality do users really use? analysis of the usage of the bscw system. *Parallel, Distributed, and Network-Based Processing, Euromicro Conference on* (2001), 337.

[2] BELLAVISTA, P., AND CORRADI, A. *The handbook of mobile middleware*. CRC Press, 2006.

[3] BERKENBROCK, C., DA SILVA, A., AND HIRATA, C. Designing and evaluating interfaces for mobile groupware systems. In *Computer Supported Cooperative Work in Design, 2009. CSCWD 2009. 13th International Conference on* (22-24 2009), pp. 368 –373.

[4] BOSTRÖM, F., NURMI, P., FLORÉEN, P., LIU, T., OIKARINEN, T.-K., VETEK, A., AND BODA, P. Capricorn - an intelligent user interface for mobile widgets. In *MobileHCI '08: Proceedings of the 10th international conference on Human computer interaction with mobile devices and services* (New York, NY, USA, 2008), ACM, pp. 327–330.

[5] BUSZKO, D., LEE, W.-H. D., AND HELAL, A. S. Decentralized ad-hoc groupware api and framework for mobile collaboration. In *GROUP '01: Proceedings of the 2001 International ACM SIGGROUP Conference on Supporting Group Work* (New York, NY, USA, 2001), ACM, pp. 5–14.

[6] CROWLEY, T., MILAZZO, P., BAKER, E., FORSDICK, H., AND TOMLINSON, R. Mmconf: an infrastructure for building shared multimedia applications. In *CSCW '90: Proceedings of the 1990 ACM conference on Computer-supported cooperative work* (New York, NY, USA, 1990), ACM, pp. 329–342.

[7] DEINERT, F., AND MAGEDANZ, T. Introducing widget-based ims client applications. *Mobile Networks and Applications* (2010), 1–8.

[8] DEMERS, A., PETERSEN, K., SPREITZER, M., TERRY, D., THEIMER, M., AND WELCH, B. The bayou architecture: Support for data sharing among mobile users. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on* (Dec. 1994), pp. 2–7.

[9] DOMMEL, H.-P., AND GARCIA-LUNA-ACEVES, J. Floor control for multimedia conferencing and collaboration. *Multimedia Systems 5* (January 1997), 23–38.

[10] DOMMEL, H.-P., AND GARCIA-LUNA-ACEVES, J. Efficacy of floor control protocols in distributed multimedia collaboration. *Cluster Computing 2*, 1 (July 1999), 17–33.

[11] DOURISH, P., AND BELLOTTI, V. Awareness and coordination in shared workspaces. In *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work* (New York, NY, USA, 1992), ACM, pp. 107–114.

[12] DUSTDAR, S., AND GALL, H. Architectural concerns in distributed and mobile collaborative systems. *Journal of Systems Architecture 49*, 10-11 (2003), 457 – 473. Evolutions in parallel distributed and network-based processing.

[13] EDWARDS, W. K. A framework for information sharing in collaborative applications. In *CHI '94: Conference companion on Human factors in computing systems* (New York, NY, USA, 1994), ACM, pp. 89–90.

[14] EDWARDS, W. K. Session management for collaborative applications. In *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work* (New York, NY, USA, 1994), ACM, pp. 323–330.

[15] EDWARDS, W. K. Policies and roles in collaborative applications. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work* (New York, NY, USA, 1996), ACM, pp. 11–20.

[16] ELLIS, C. A., GIBBS, S. J., AND REIN, G. Groupware: some issues and experiences. *Commun. ACM 34*, 1 (1991), 39–58.

[17] ENGELBART, D. C., AND ENGLISH, W. K. A research center for augmenting human intellect. In *AFIPS '68 (Fall, part I): Proceedings of the December 9-11, 1968, fall joint computer conference, part I* (New York, NY, USA, 1968), ACM, pp. 395–410.

[18] ISHII, H. Teamworkstation: towards a seamless shared workspace. In *CSCW '90: Proceedings of the 1990 ACM conference on Computer-supported cooperative work* (New York, NY, USA, 1990), ACM, pp. 13–26.

[19] KAAR, C. An introduction to widgets with particular emphasis on mobile widgets. Tech. rep., Mobile Computing University of Applied Sciences, Hagenberg, October 2007.

[20] KIRDA, E., FENKAM, P., REIF, G., AND GALL, H. A service architecture for mobile teamwork. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering* (New York, NY, USA, 2002), ACM, pp. 513–518.

[21] KREBS, A. M., AND MARSIC, I. Adaptive applications for ubiquitous collaboration in mobile environments. *Hawaii International Conference on System Sciences 1* (2004).

[22] LAMPSON, B. W. Protection. *SIGOPS Oper. Syst. Rev. 8*, 1 (1974), 18–24.

[23] LI, D., AND MUNTZ, R. Coca: collaborative objects coordination architecture. In *CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work* (New York, NY, USA, 1998), ACM, pp. 179–188.

[24] LUFF, P., AND HEATH, C. Mobility in collaboration. In *CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work* (New York, NY, USA, 1998), ACM, pp. 305–314.

[25] MORAN, T. P., AND ANDERSON, R. J. The workaday world as a paradigm for cscw design. In *CSCW '90: Proceedings of the 1990 ACM conference on Computer-supported cooperative work* (New York, NY, USA, 1990), ACM, pp. 381–393.

[26] MØRCH, A. Three levels of end-user tailoring: customization, integration, and extension. *Computers and design in context* (1997), 51–76.

[27] MUNSON, J., AND DEWAN, P. A concurrency control framework for collaborative systems. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work* (New York, NY, USA, 1996), ACM, pp. 278–287.

[28] PERRY, M., O'HARA, K., SELLEN, A., BROWN, B., AND HARPER, R. Dealing with mobility: understanding access anytime, anywhere. *ACM Trans. Comput.-Hum. Interact. 8*, 4 (2001), 323–347.

[29] PRASAD, S. K., MADISETTI, V., NAVATHE, S. B., SUNDERRAMAN, R., DOGDU, E., BOURGEOIS, A., WEEKS, M., LIU, B., BALASOORIYA, J., HARIHARAN, A., XIE, W., MADIRAJU, P., MALLADI, S., SIVAKUMAR, R., ZELIKOVSKY, A., ZHANG, Y., PAN, Y., AND BELKASIM, S. Syd: a middleware testbed for collaborative applications over small heterogeneous devices and data stores. In *Middleware '04: Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware* (New York, NY, USA, 2004), Springer-Verlag New York, Inc., pp. 352–371.

[30] R, D. A., H.S., D. S., BERNHARD, J., AND MAHARDITO, A. Study case: Distributed groupware and collaborative web agents. In *Software Engineering, 2009. WCSE '09. WRI World Congress on* (May 2009), vol. 1, pp. 195–198.

[31] RAATIKAINEN, K., CHRISTENSEN, H. B., AND NAKAJIMA, T. Application requirements for middleware for mobile and pervasive systems. *SIGMOBILE Mob. Comput. Commun. Rev. 6*, 4 (2002), 16–24.

[32] REINHARD, W., SCHWEITZER, J., VÖLKSEN, G., AND WEBER, M. Cscw tools: Concepts and architectures. *Computer 27*, 5 (1994), 28–36.

[33] REKIMOTO, J. Tilting operations for small screen interfaces. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology* (New York, NY, USA, 1996), ACM, pp. 167–168.

[34] ROSEMAN, M., AND GREENBERG, S. Groupkit: a groupware toolkit for building real-time conferencing applications. In *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work* (New York, NY, USA, 1992), ACM, pp. 43–50.

[35] ROTH, J. Seven challenges for developers of mobile groupware. Tech. rep., University of Hagen, Department for Computer Science, Hagen, Germany, 2004.

[36] ROTH, J. The resource framework for mobile applications. In *Enterprise Information Systems V*. Springer Netherlands, 2005, pp. 300–307.

[37] SACRAMENTO, V., ENDLER, M., RUBINSZTEJN, H., LIMA, L., GONCALVES, K., NASCIMENTO, F., AND BUENO, G. Moca: A middleware for developing collaborative applications for mobile users. *Distributed Systems Online, IEEE 5*, 10 (Oct. 2004), 2–2.

[38] SHEN, H., AND DEWAN, P. Access control for collaborative environments. In *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work* (New York, NY, USA, 1992), ACM, pp. 51–58.

[39] SLAGTER, R., BIEMANS, M., AND TER HOFTE, H. Evolution in use of groupware: facilitating tailoring to the extreme. In *Groupware, 2001. Proceedings. Seventh International Workshop on* (2001), pp. 68–73.

[40] SU, X., PRABHU, B., CHU, C.-C., AND GADH, R. Middleware for multimedia mobile collaborative system. In *Wireless Telecommunications Symposium, 2004* (May 2004), pp. 112–119.

[41] SULEIMAN, M., CART, M., AND FERRIE, J. Concurrent operations in a distributed and mobile collaborative environment. In *Data Engineering, 1998. Proceedings., 14th International Conference on* (Feb 1998), pp. 36–45.

[42] TERRY, D. B., DEMERS, A. J., PETERSEN, K., SPREITZER, M., THEIMER, M., AND WELCH, B. W. Session guarantees for weakly consistent replicated data. In *PDIS '94: Proceedings of the Third International Conference on Parallel and Distributed Information Systems* (Washington, DC, USA, 1994), IEEE Computer Society, pp. 140–149.

[43] TRUONG, H.-L., JUSZCZYK, L., BASHIR, S., MANZOOR, A., AND DUSTDAR, S. Vimoware - a toolkit for mobile web services and collaborative computing. In *Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference* (Sept. 2008), pp. 366–373.

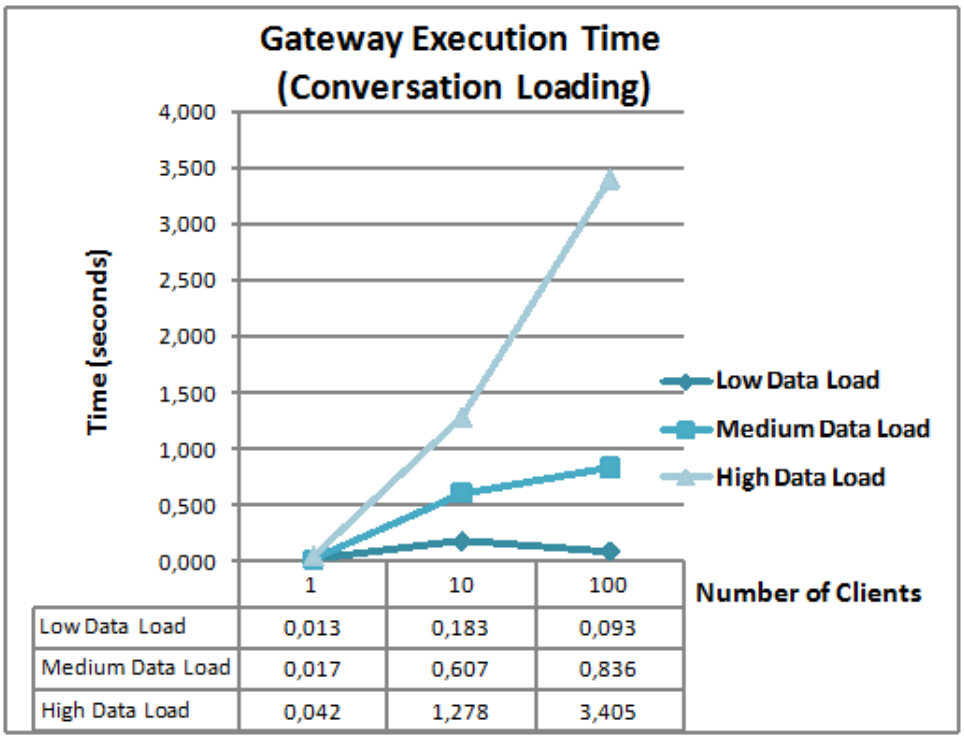
[44] WEISER, M. Some computer science issues in ubiquitous computing. *Commun. ACM 36*, 7 (1993), 75–84.

[45] YLIANTTILA, M., HARJULA, E., KOSKELA, T., KASSINEN, O., AND RIEKKI, J. Mobile plug-and-play architecture for collaborative hybrid peer-to-peer applications. In *Congress on Services Part II, 2008. SERVICES-2. IEEE* (Sept. 2008), pp. 81–87.

Appendices

A Additional Test Results

A.1 PUC Gateway-Only Execution Time Measurements



	1	10	100
Low Data Load	0,013	0,183	0,093
Medium Data Load	0,017	0,607	0,836
High Data Load	0,042	1,278	3,405

Figure A.1: PUC Gateway-Only Execution Time Results (on Average) for Conversation Loading Operations

A.2 Cumulative Execution Time Measurements Conversation Loading

	1	10	100
PUC Execution Time	0,322	1,009	7,350
Gateway Execution Time	0,013	0,183	0,093

	1	10	100
PUC Execution Time	3,053	20,486	96,239
Gateway Execution Time	0,017	0,607	0,836

	1	10	100
PUC Execution Time	5,623	45,738	139,643
Gateway Execution Time	0,042	1,278	3,405

Figure A.2: Cumulative Execution Time Results (on Average) for Conversation Loading Operations (individual execution times in seconds)

Group Creation

	1	10	100
PUC Execution Time	0,269	1,229	5,637
Gateway Execution Time	0,038	0,506	0,595

	1	10	100
PUC Execution Time	0,633	2,009	11,322
Gateway Execution Time	0,081	0,749	4,772

	1	10	100
PUC Execution Time	1,443	6,789	27,293
Gateway Execution Time	0,129	2,431	8,210

Figure A.3: Cumulative Execution Time Results (on Average) for Group Creation Operations (individual execution times in seconds)

Participation Loading

	1	10	100
PUC Execution Time	0,195	0,628	4,743
Gateway Execution Time	0,013	0,807	0,069

	1	10	100
PUC Execution Time	2,530	20,817	83,757
Gateway Execution Time	0,057	0,155	0,283

	1	10	100
PUC Execution Time	3,895	26,889	164,743
Gateway Execution Time	0,030	0,163	0,511

Figure A.4: Cumulative Execution Time Results (on Average) for Session Participation Loading Operations (individual execution times in seconds)

A.3 PUC Gateway Execution Time Distribution Conversation Loading

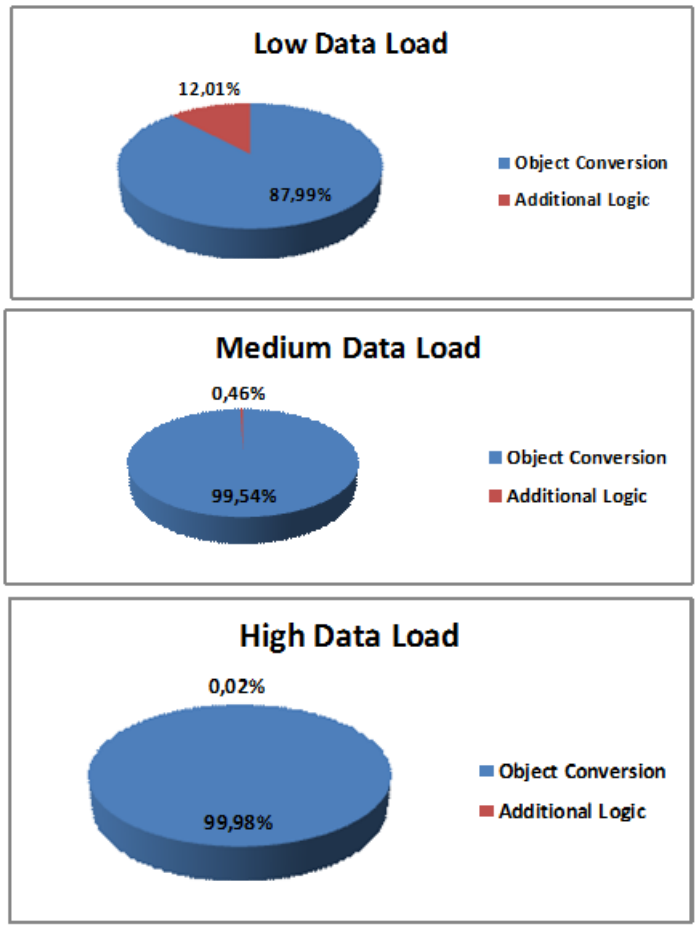


Figure A.5: PUC Gateway Execution Time Distribution (in percentage) over Conversation Loading Operations

Participation Loading

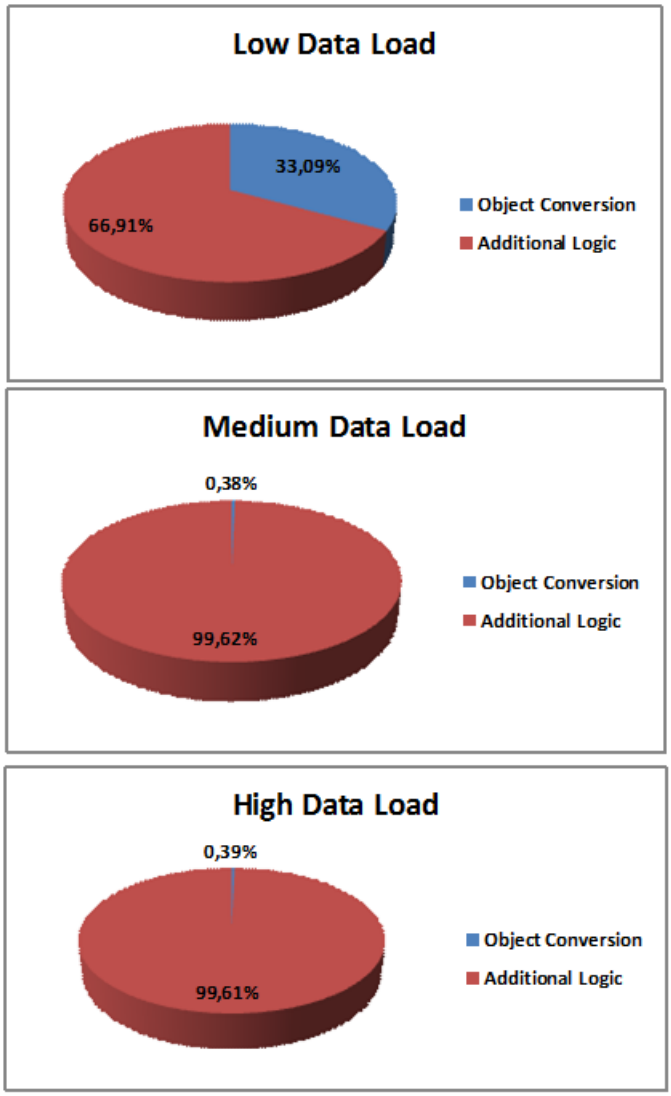


Figure A.6: PUC Gateway Execution Time Distribution (in percentage) over Session Participation Loading Operations

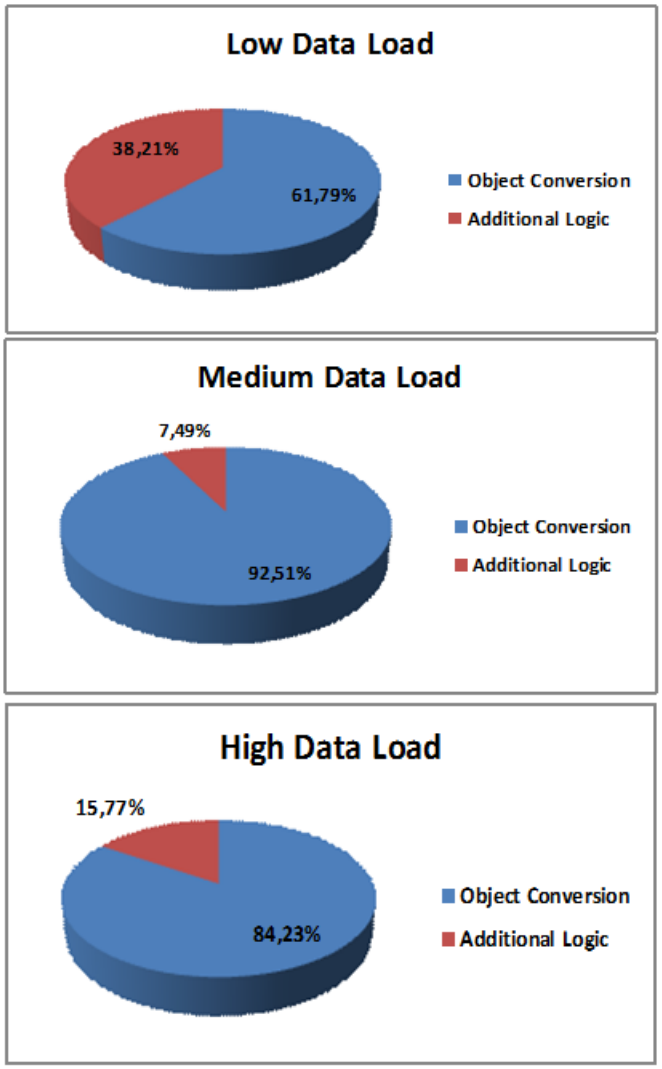


Figure A.7: PUC Gateway Execution Time Distribution (in percentage) over Group Creation Operations

A.4 Memory Consumption - Full Data Model and UA Data Model Comparison

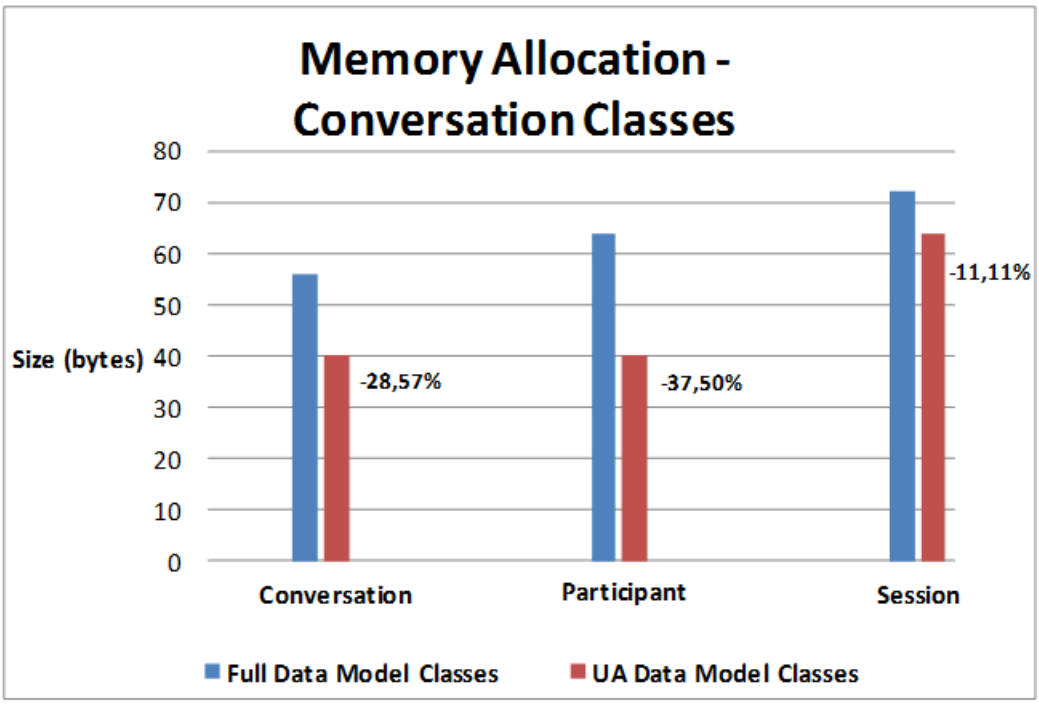


Figure A.8: Allocated Size (in bytes) per object in the Java Heap Space for both Full and UA Data Models (Conversation classes)

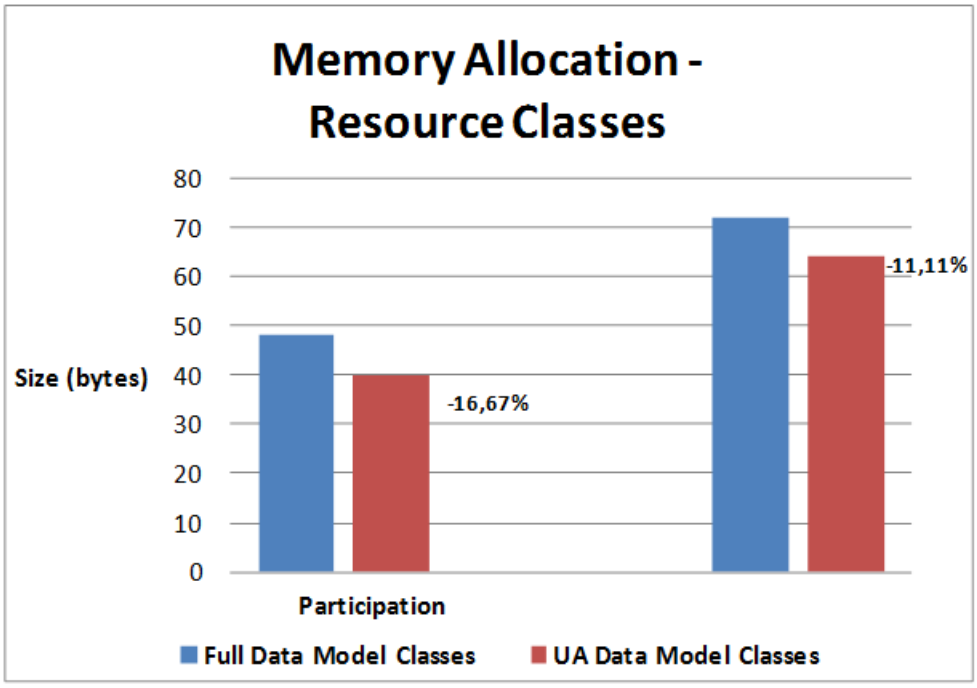


Figure A.9: Allocated Size (in bytes) per object in the Java Heap Space for both Full and UA Data Models (Resource classes)

A.5 PUC Gateway Memory Consumption - Allocated Memory and Object Age Comparison

Average Object Age During Test Execution

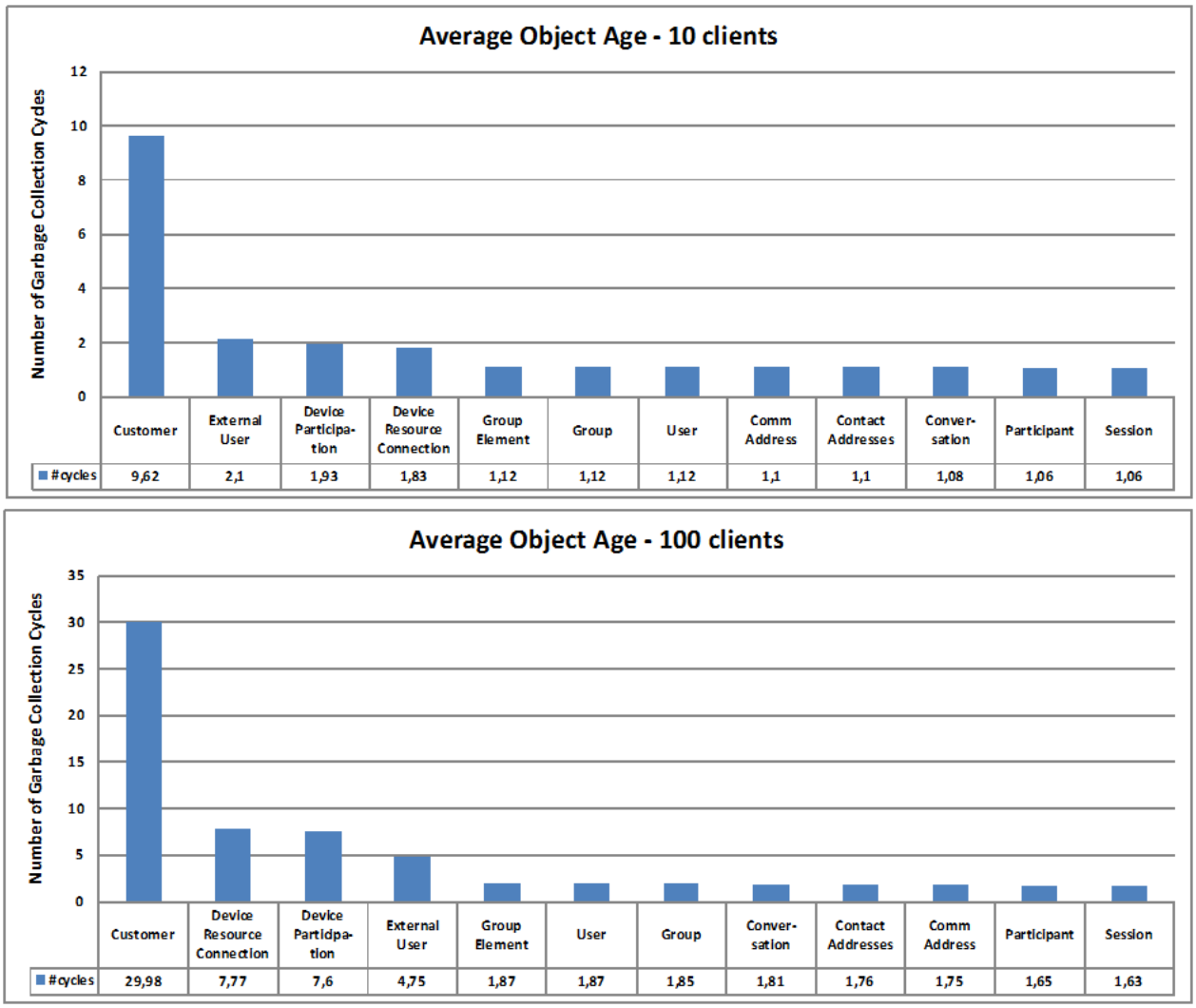


Figure A.10: Average Object Age Measured in The Number of Surviving Garbage Collection Cycles by each UA Data Model Object

Amount of Allocated Memory During Test Execution

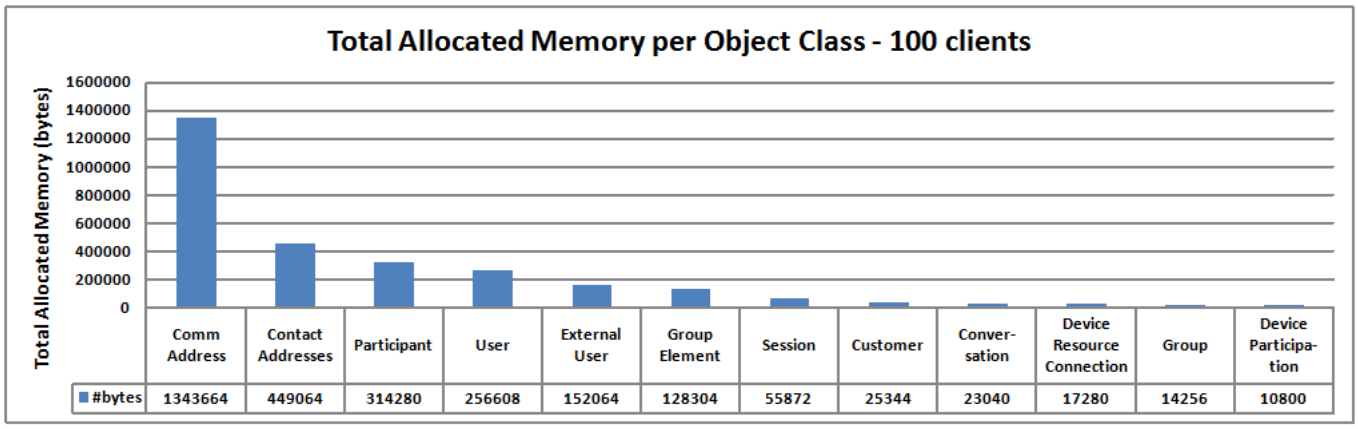


Figure A.11: Total Allocated Memory by each UA Data Model Over a Test Execution With 100 Clients and Medium Data Load

A.6 Prototype Application - Battery Consumption

Battery Usage

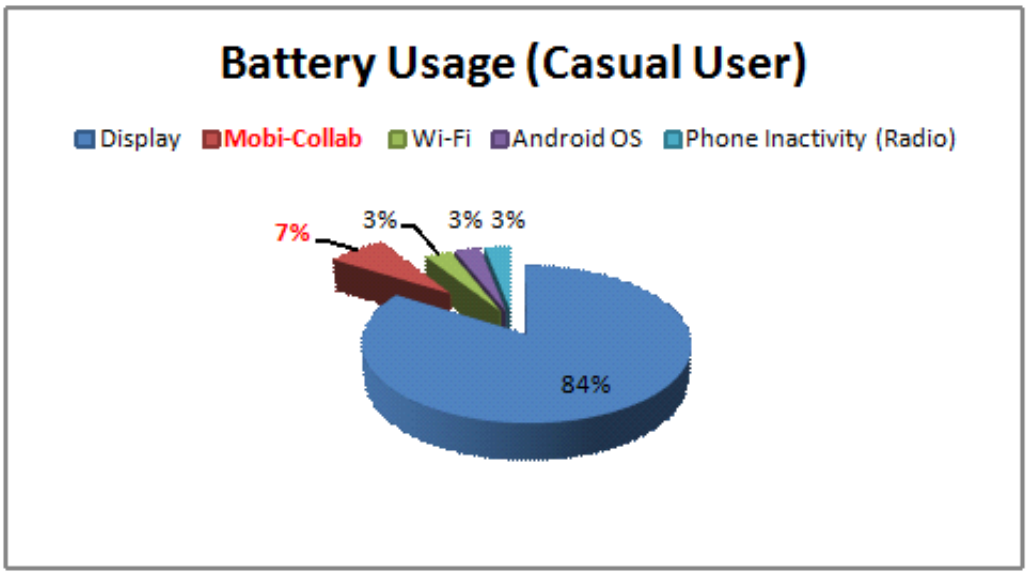


Figure A.12: Percentage of Available Battery Energy Used by The End-User Application During Test Execution (Casual User Profile)

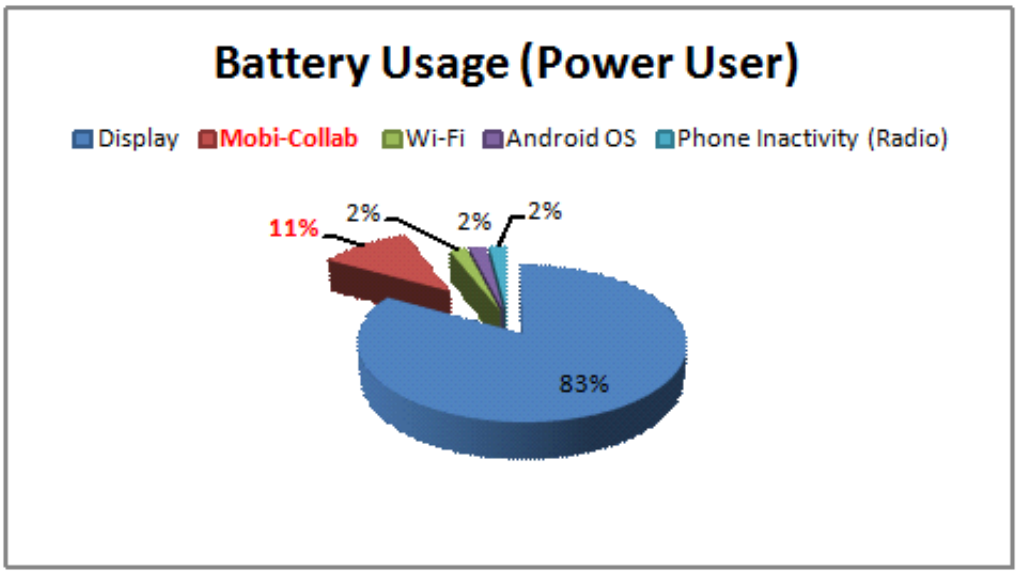


Figure A.13: Percentage of Available Battery Energy That Was Used by The End-User Application During Test Execution (Power User Profile)

B Developed Application Widgets

In this apendix chapter, we present the developed application widgets, as they are presented and usable by the end-user.

Login and Account Widgets

The Login Widget (presented in Fig.B.1) is the first widget that gets loaded in our application. As the name implies, it makes the login of provisioned end-users that are registered in our server side database.

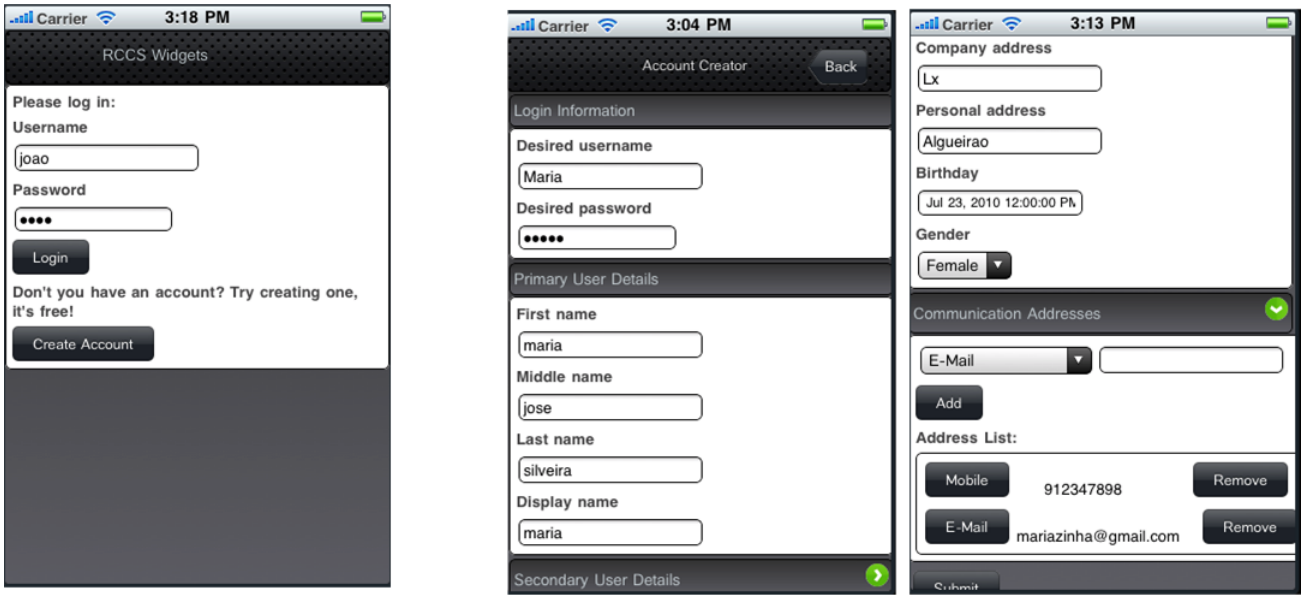


Figure B.1: Login and Account Widgets Screenshots

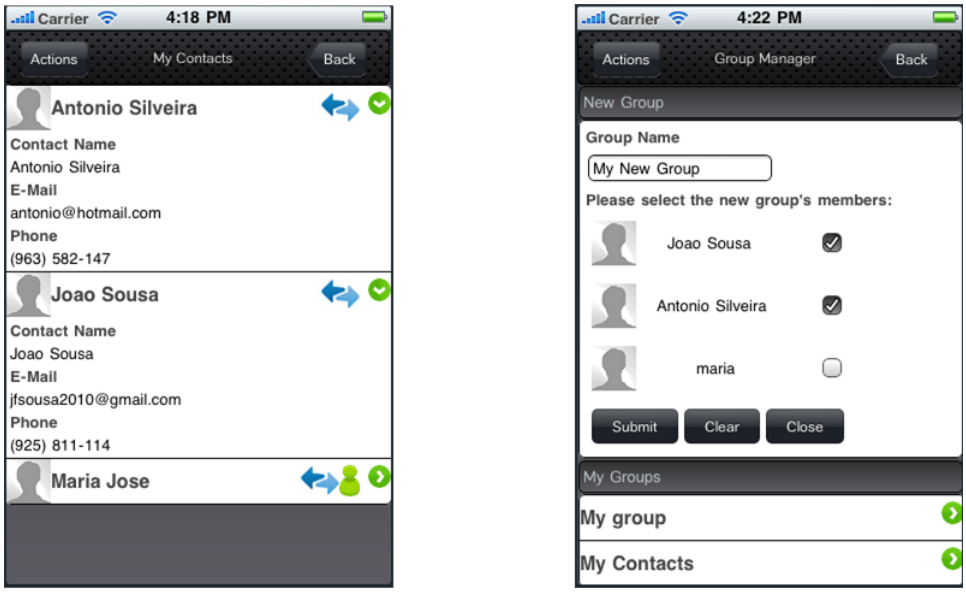
There two users types: *Customers* or *External Users*. Customers are essentially users that create an account in our system through the Account Widget. Customers have access to all of our Widgets and features; can store relevant personal information and a customised list of *Communication Addresses* (fixed phone or mobile numbers, E-Mails, etc.) that the user wishes to expose to other users. External Users act as guests that can be invited by Customer Users into given collaborative sessions.

A given user can create an account in two easy steps: by filling his/her login details a specific user information and register set of contact addresses (phone numbers or E-Mails) that he/her wants to expose to other users (see the Account Widget's second screenshot in Fig.B.1).

My Contacts and Group Widget

These widgets are represented in Fig.B.2. A given user can deliberately synchronise all contacts in his/her mobile device's contact list. For this, the user just needs to open the My Contacts Widget and

select "Synchronise Contacts" in the Actions Menu with just two clicks.



My Contacts Widget **Group Widget**

Figure B.2: My Contacts and Group Widgets Screenshots

Additionally, it is also possible to create specific user Groups that will be later used in the created collaborative sessions, by opening the Group Manager through the "Actions" in the My Contacts list.

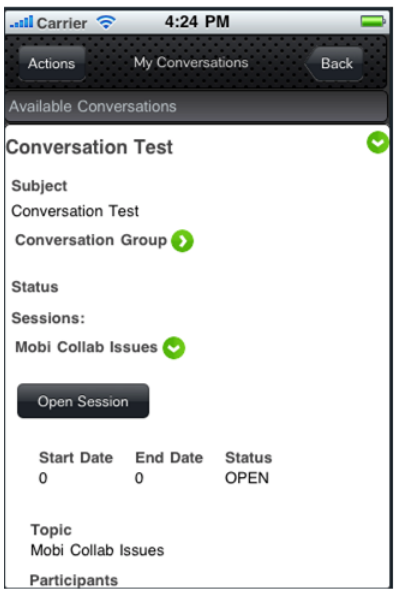
Successfully synchronised contacts get properly assigned as External Users or as Customers if they have a Customer Account in PUC. A given user immediately knows what are the contacts already synchronised and which of them have a Customer account. Contact identification is not made with the local names stored in-device, but, with the phone numbers or E-Mails themselves. If a synchronised communication address belongs in a given Customer Account, the respective contact gets assigned as a Customer in the user's My Contacts view, and all the information that the user has locally in his/her device about that contact, gets overwritten with the information residing in the respective contact's customer account.

Conversation Box and Session Widget

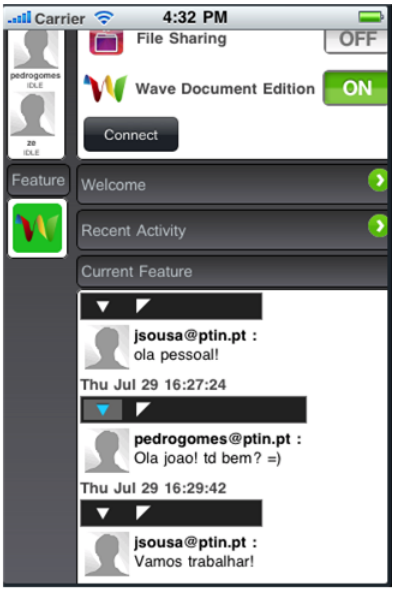
Collaborative sessions are grouped in Conversations with each belonging in a given Conversation Group. These groups of users are constructed from a given user's contact list and therefore the user owns this group and has administrative rights over it and over sessions that are created subsequently on the corresponding conversation.

Conversation/Session creation, removal and loading are functions that reside in the Conversation Box Widget (see Fig.B.3). Upon completion of a given session's loading, the Session Widget gets presented to the user, and then functionality like participant management, resource management and status management gets available.

The developed Resource Widgets that provide file sharing and chat functionalities are presented in Fig.B.4.

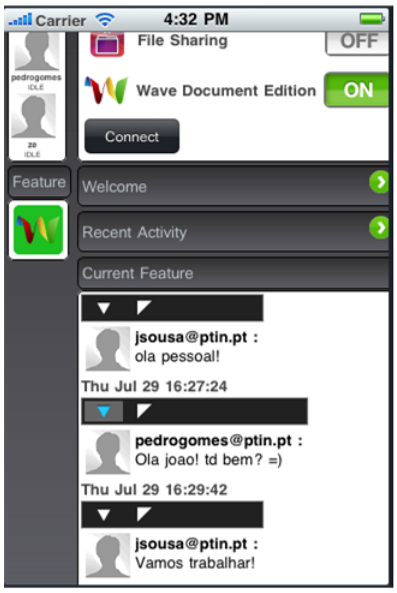


Conversation Box Widget



Session Widget

Figure B.3: Conversation Box and Session Widgets Screenshots



Wave Chat Widget

File Sharing Widget

Figure B.4: File Sharing and Wave Chat Widgets Screenshots

C Photos of Testing Devices

Figure C.1: Mobile Device Utilised in The Client-Side Implementation Assessment

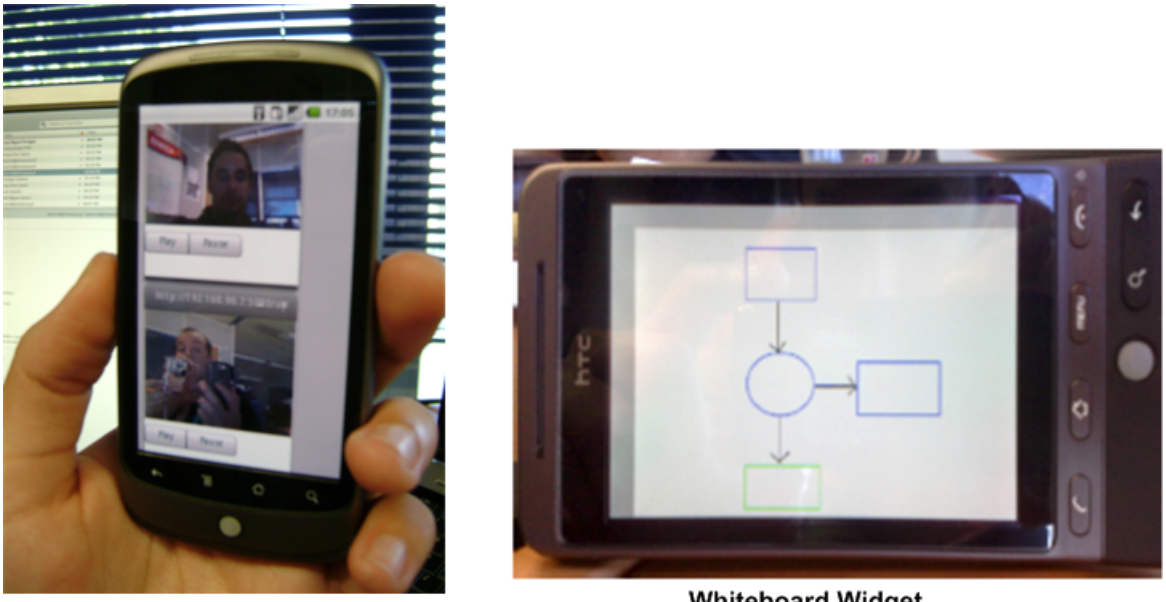


Figure C.2: Audio/Video Conference and Whiteboard Flash Widgets Running in Android Devices

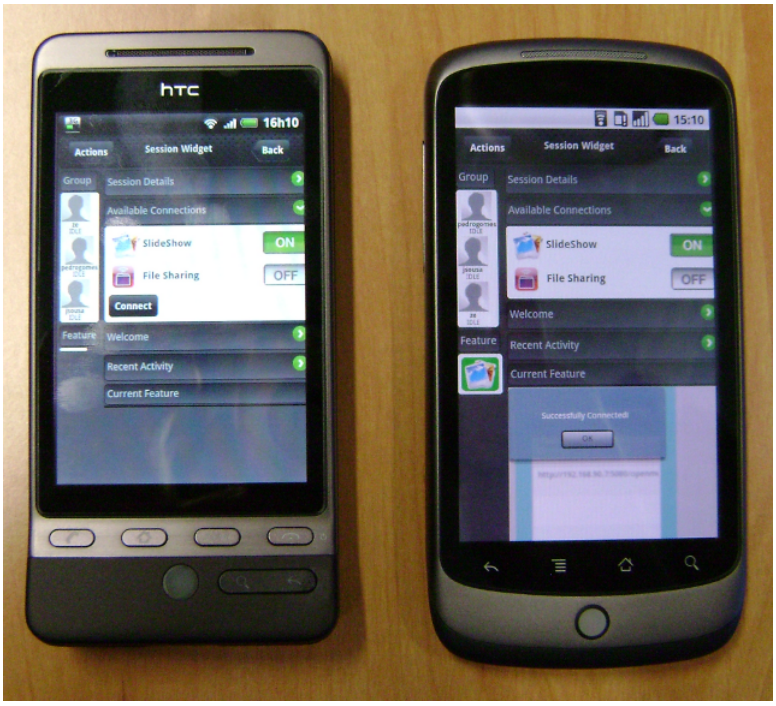


Figure C.3: HTC Hero's failed In-Widget Flash Content Execution Vs Google Nexus One