MOBI-COLLAB: Collaborative Applications for Mobile Devices

João Sousa
IST
Technical University of Lisbon
joao.f.sousa@ist.utl.pt

ABSTRACT

Today, mobile devices are becoming more powerful, and the ever increasing connectivity of this type of devices motivates 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, capitalising 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.

Keywords

Collaboration; Groupware; Mobility; Middleware; Portability; Widgets

1. INTRODUCTION

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 technologies, motivate us to look into what can be achievable by mobile technologies.

Considering these factors, the concept of ubiquitous computing, described by Mark Weiser [16], 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. New collaborative groupware solutions must 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 PUC (Plataforma Unificada de Colaboração). It is a collaboration platform that aims to provide an unified service architecture which may be used by external client applications, who want to provide collaborative functionality to the end-user.

Mobi-Collab is also part of this initiative, but it distinguishes itself by focusing on the delivery of a common framework for the development of collaborative applications in mobile devices, using PUC as a supporting platform.

1.1 Objectives

Current mobile groupware solutions have been developed in specific programming languages which are tightly integrated with the native APIs of the targeted operating systems [2, 6, 14, 13, 8, 15]. This results in the development of solutions that offer low portability levels. This limitation is addressed in our solution by exploring the Mobile Web Development approach and a Web services based communication between the underlying system components.

In this paper, we propose the accomplishment of 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.

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.

2. RELATED WORK

Collaboration has its foundations in the concept of *Computer Supported Cooperative Work* (CSCW) [5], which introduced shared work stations; shared file systems and conferencing tools. *Groupware* [4] evolved this concept by formally introducing the notion of a *group* of people that engage in a common task (or goal) through an interface in a shared environment.

2.1 Groupware Support

Due to the sharing and dissemination of information in its applications, groupware research has been overlapped with many disciplines. Support for groupware applications must take into account many concepts from different perspectives, including: distributed systems; network communications; Human-computer interaction and social theory.

This has led to the development of groupware solutions that generally follow a *time space taxonomy*, that distinguishes groupware interactions from the locality of the interactions (local or distributed) and time (synchronous or asynchronous).

Concerning the architectural of groupware systems, two approaches are generally followed [10]:

- **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.
- **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.

The development of Groupware also comprises several challenging issues, that include access control; awareness; coordination; concurrency control; session management and tailorability.

2.2 Mobile Groupware Support

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.

The concept of micro-mobility [7] shows us that small artifacts, like a piece of paper, for example, 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.

This as led to the development of groupware solutions that explore the theoretical ideal of *access, anytime, anywhere*, by focusing in the delivery of these type of solutions to mobile devices.

Achieving a groupware experience in mobile devices comparable with the desktop counterpart is not easy. Mobile environments introduce new challenges, such as communication issues; data distribution and consistency and user interface issues [3, 11, 12].

In order to overcome these challenges and provide a **faster service development and deployment**, service and application frameworks and platforms have been developed adopting the definition of middleware [9, 1].

2.3 Middleware in Mobile Groupware

Among the requirements that middleware for mobile groupware application need to address we can find: the provision of an Execution support layer; persistence support; adaptability; provision of a distributed information base; extensibility and portability.

Several solutions have been developed in this area.

For example, YCab [2], is a decentralised groupware framework that provides a Java based API for developers to quickly create collaborative applications with a minimal learning curve. 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.

Su et al. [14] 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. 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.

MoCA [13], interestingly provides proxies that 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. In this way, it is possible to develop client and server applications while not concerning about portability and content adaptability issues, leaving that for the proxy to handle.

As one solution that closely resembles our own, we have Vimoware [15]. It relies on a SOAP based Web Services model in the deployment and use of its services. Thus, exploiting the interoperability, flexibility,and re-usability of Web services based software.

3. ARCHITECTURE

3.1 Mobi-Collab Overview

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

Five different entities are involved in the communication process:

- **PUC Platform (in-house developed):** a set of collaboration service enablers, exposed by a Public API that gives support for our groupware functionality.
- **Mobile Terminal:** the mobile device itself. All application code is locally deployed in the device, as are, our middleware components. 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. Allows 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 the *adaptability* requirement, by providing resources according the client device's capabilities.
- **Bayeux Event Broker:** 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:** these are the actual servers which host resources that can be accessed by our groupware applications.

3.2 Communication

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 (see Fig.1).

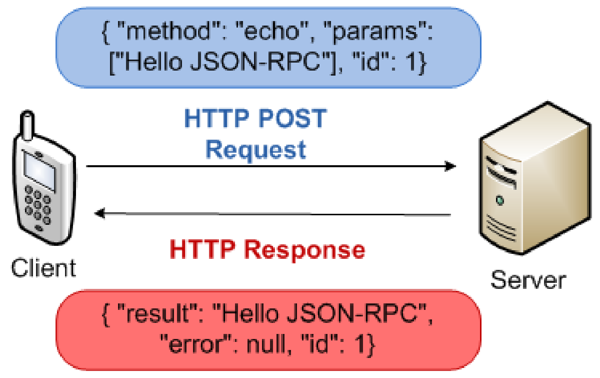

{ "method": "echo", "params": ["Hello JSON-RPC"], "id": 1}
HTTP POST Request
Client
HTTP Response
Server
{ "result": "Hello JSON-RPC", "error": null, "id": 1}

Figure 1: JSON-RPC Communication Example

HTTP POST requests and responses follow the JSON-RPC[1] over HTTP protocol. It is basically, a lightweight remote procedure call protocol that uses the JSON message format. Returned JSON strings in HTTP responses are then unmarshalled on the client-side JavaScript code to create data objects of the returned information.

The Communication with the Event Broker follows the Bayeux message transportation protocol.[2] In order to minimise latency in server-client message delivery, we opted to use Bayeux's *long-polling* transport variant (see Fig.2). 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.

[1]JSON-RPC Specification - http://json-rpc.org/wiki/specification
[2]Bayeux Protocol - http://svn.cometd.com/trunk/bayeux.html

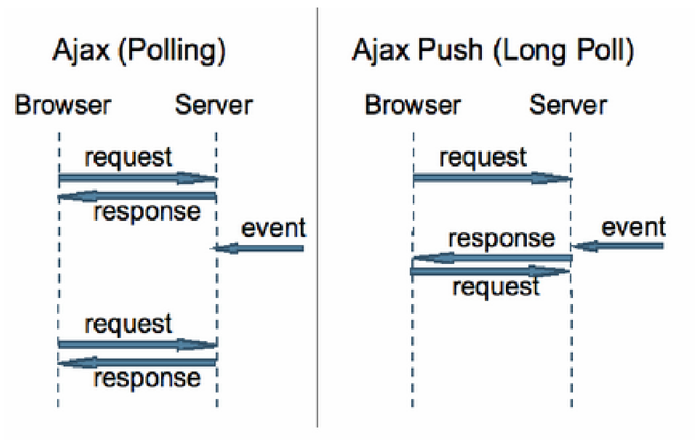


Figure 2: Standard Polling versus Long Polling Transportation

Communication between the PUC gateway and PUC's Public API is accomplished through Enterprise Java Beans[3] remote method invocations to exposed remote interfaces.

3.3 PUC Gateway

The PUC Gateway has a 3-tier architecture:

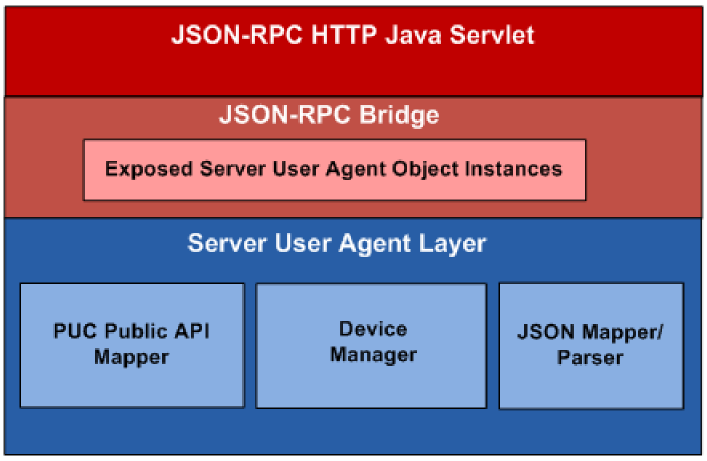


Figure 3: View of PUC Gateway's Architecture

- **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.

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

PUC Gateway also has its own data model, designed with the objective of providing a simpler data model for resource constrained devices.

3.4 Mobile Terminal

The architecture of the mobile terminal has also a layered design with the upper layers (Application + Core User Agent Layer) which are closer to the application side, being developed by us. The underlying layers correspond to third-party components that give support to our own core libraries.

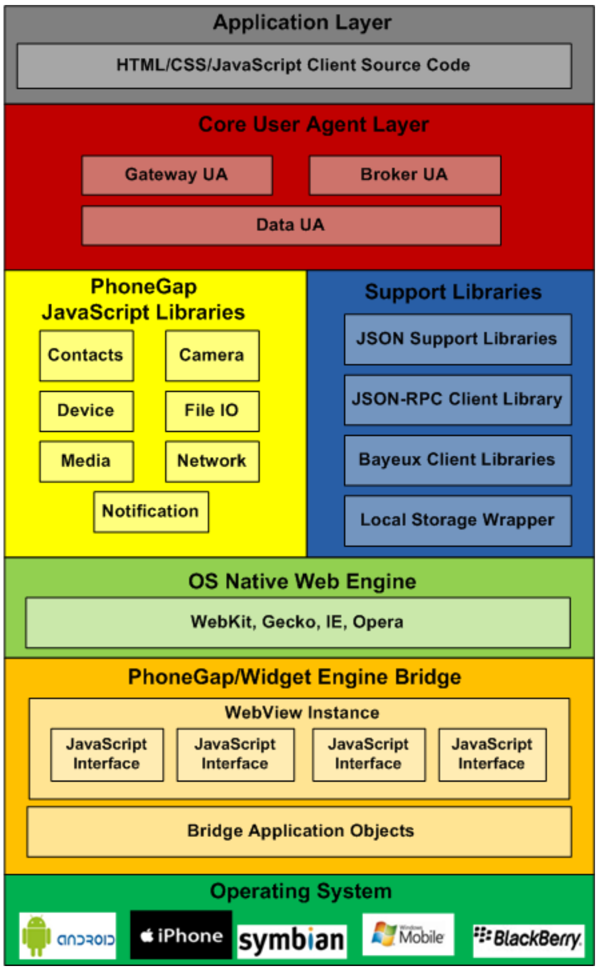


Figure 4: View of Mobile Terminal's Architecture

The above design comprises:

- **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 and abstracts the JSON-RPC based communication with PUC Gateway to the end-developer. A *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.

- **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. The *bayeux client libraries* abstract the Bayeux message transportation protocol and 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 underlying native functionality.

- **Operating System (Third party) :** provides the execution runtime and the device's native functionality.

4. IMPLEMENTATION

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 (see Table.1).

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

Table 1: Technology Usage Map

4.1 Implementing the PUC Gateway

PUC Gateway's main implementation challenges lied in how to leverage available technology in order to minimise the development time of the Web hosting and initialisation of its application components.

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.

Clients are then allowed to make JSON-RPC calls to the gateway, as long as they send their requests to the configured url.

In order for client requests to be routed for the corresponding Java objects, a JSON-RPC Bridge is needed. For that we used the JSON-RPC servlet that is provided by the Jabsorb Framework.

To implement object migration between PUC Gateway and the mobile terminal we used the provided conversion tools by the JSON Tools framework.

We have managed to design a Server User Agent component that accomplishes the delivery of resources according to the accessing device's capabilities.

The information regarding a device's capabilities and supported collaborative features is saved in the form of object instances of the Gateway's *User Agent Data Model* . This information is generated from two manually edited XML documents, in conformation with a specific XML schema and loaded on the Gateway's application start up.

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.

4.2 Prototype Application Widgets

In order to validate our terminal middleware layer we developed a groupware application prototype that makes use of both our middleware and PUC Gateway components.

As it was developed using Google Web Toolkit[4] our application is totally Java based. 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 (see Fig.5).

5. EVALUATION

All of our developed components were subject of assessment through pre-established **qualitative** criteria and **quantitative** metrics.

[4]GWT Overview - http://code.google.com/intl/pt-PT/webtoolkit/overview.html

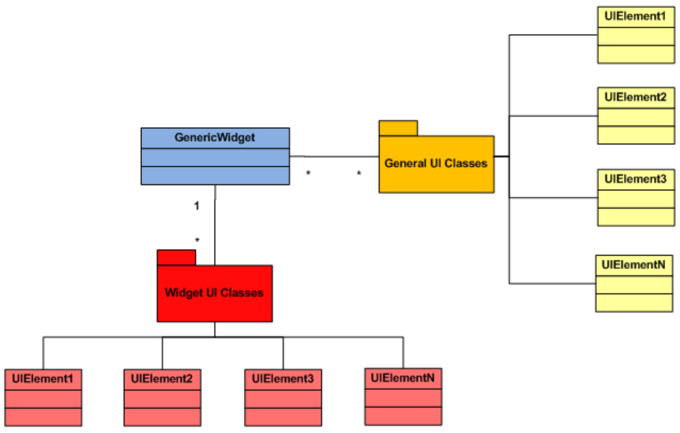


Figure 5: Generic Widget Java Object Structure

We have developed the Gateway component with intent of causing 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.

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.

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. Therefore we need to know much of an overhead the Gateway to PUC's sole execution times and if memory consumption scales adequately in order for the system not to crash due to lack of available memory in the Java Heap Space.

5.1.1 Testing Environment

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.

5.1.2 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**.

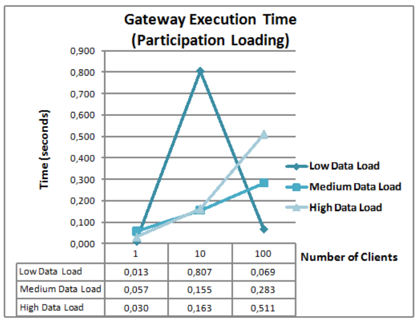
Concerning the number of clients, we have established boundary values that go from **1**, to **10** and **100** clients.

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 Table.5.1.2.

5.1.3 Execution Time Tests



	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

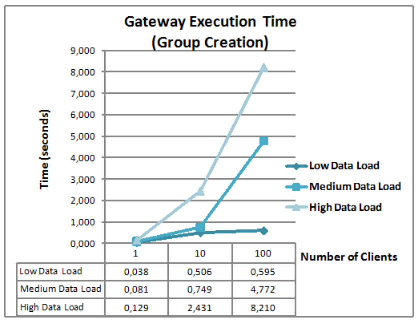
Figure 6: PUC Gateway-Only Execution Time Results (on Average) for Session Participation Loading Operations

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. That is because these operations have the least object conversion logic, being ultimately unaffected by the increase in object sizes. In contrast, Conversation Loading and Group Creation operations suffer more, because they are heavy in object conversion logic and as we can observe by the Group Creation, 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.

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 com-

	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

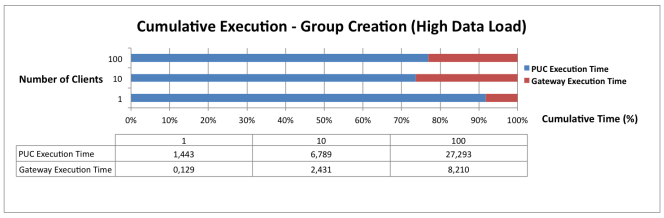
Table 2: Data Load Variations for Client Thread Executions



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

putation 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.



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

Figure 8: Cumulative Execution Time Results (on Average) for Group Creation Operations (individual execution times in seconds)

The introduced overhead by the Gateway in PUC's execution times, is for Conversation and Participation Loading, 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) (Fig.8). Group Creation 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 performance. Nevertheless, the Gateway execution times never surpass PUC's execution times (they do not even generally reach a 50% overhead).

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.

5.1.4 *Memory Consumption Tests*

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.

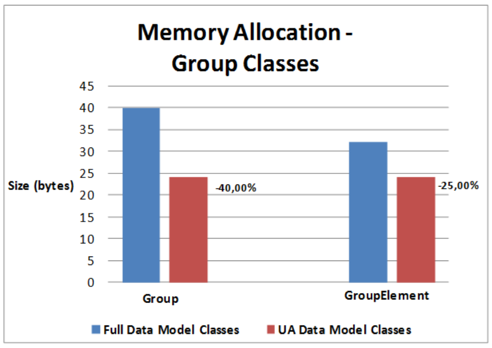


Figure 9: Allocated Size (in bytes) per object in the Java Heap Space for both Full and UA Data Models (Group classes)

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.9), other classes, such as User and Resource classes have only achieved reduction values around 11% (Fig.10).

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

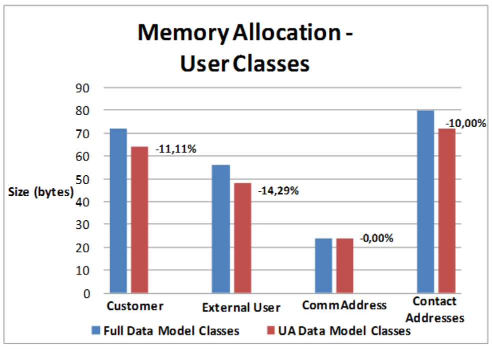


Figure 10: Allocated Size (in bytes) per object in the Java Heap Space for both Full and UA Data Models (User classes)

later spawning of multiple requests to obtain that same information.

The utilised tools permitted a detailed analysis over the Java Heap Space during test execution. This allowed us to observe how many Garbage Collection Cycles an object survives during test execution and the total allocated memory per object type.

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. 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.

5.2 Client-Side Implementation Assessment

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.

5.2.1 Testing Environment

The testing environment for the assessment of the client-side implementation follows the same network topology that was described in the previous section, 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 Application in the Android operating system over a Wi-Fi connection.

5.2.2 Portability Evaluation

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: Android; 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.

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.

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.

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.

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 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.

5.2.3 Quantitative Evaluation

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.

Fig.11 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

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 3: Overview of The Portability of Our Implementation Across The Targeted Operating Systems

that the function has been successfully executed in the application, in order to evaluate the introduced overhead by the application.

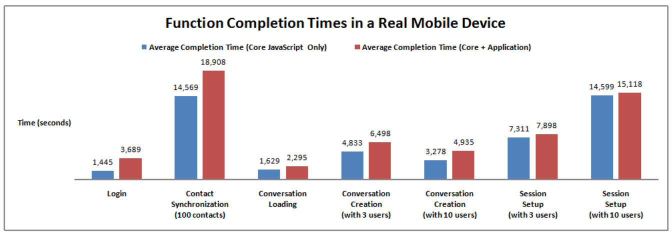


Figure 11: Measured Average Completion Times of Specific Functions in The Client-Side Implementation Executing in a Real Mobile Device

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.

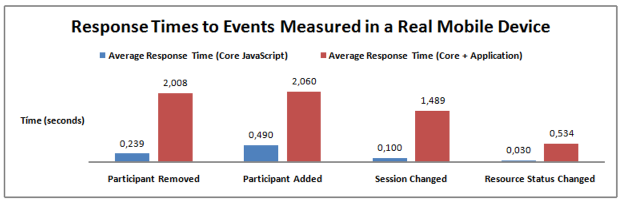


Figure 12: Measured Average Response Times To Specific Events in The Client-Side Implementation Executing in a Real Mobile Device

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 *Battery Consumption*

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**. Our tests comprised a total execution time of 1 hour and 25 minutes on all user profiles.

Evidently, user profiles with an higher operation frequency detained more CPU Time. 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%).

Although battery usage percentages are small, the battery consumption percentages after test execution are significant. 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).

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

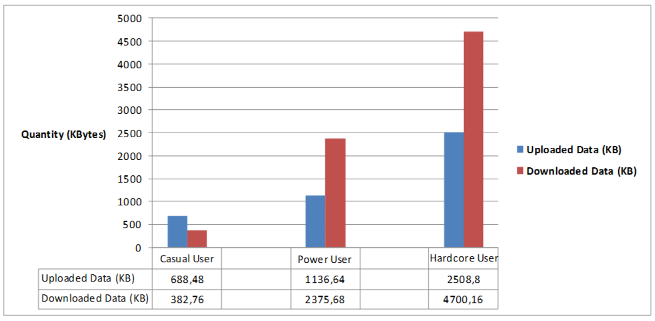
Table 4: Estimate of Application Usage Time in Each User Profile Until Battery Gets Drained

The results are encouraging. Even in the hardcore user profile, the battery can last a considerable amount of time, considering that 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.

5.2.5 *Data Usage over Mobile Networks*

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.

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.



	Casual User	Power User	Hardcore User
Uploaded Data (KB)	688,48	1136,64	2508,8
Downloaded Data (KB)	382,76	2375,68	4700,16

Figure 13: Amount of Transmitted Data By The Mobi-Collab Prototype App Over a Mobile Network on Each User Profile

6. CONCLUSIONS

All of the undertaken work definitely paid off, as the evaluation phase, 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.

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.

Future Work

- **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. We need to guarantee the confidentiality, integrity and authentication of exchanged messages in order to build collaborative applications that implement the minimal security features that are currently available.

- **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.

- **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.

7. REFERENCES

[1] P. Bellavista and A. Corradi. *The handbook of mobile middleware*. CRC Press, 2006.

[2] D. Buszko, W.-H. D. Lee, and A. S. Helal. 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*, pages 5–14, New York, NY, USA, 2001. ACM.

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

[4] C. A. Ellis, S. J. Gibbs, and G. Rein. Groupware: some issues and experiences. *Commun. ACM*, 34(1):39–58, 1991.

[5] D. C. Engelbart and W. K. English. 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*, pages 395–410, New York, NY, USA, 1968. ACM.

[6] E. Kirda, P. Fenkam, G. Reif, and H. Gall. A service architecture for mobile teamwork. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 513–518, New York, NY, USA, 2002. ACM.

[7] P. Luff and C. Heath. Mobility in collaboration. In *CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 305–314, New York, NY, USA, 1998. ACM.

[8] S. K. Prasad, V. Madisetti, S. B. Navathe, R. Sunderraman, E. Dogdu, A. Bourgeois, M. Weeks, B. Liu, J. Balasooriya, A. Hariharan, W. Xie, P. Madiraju, S. Malladi, R. Sivakumar, A. Zelikovsky, Y. Zhang, Y. Pan, and S. Belkasim. 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*, pages 352–371, New York, NY, USA, 2004. Springer-Verlag New York, Inc.

[9] K. Raatikainen, H. B. Christensen, and T. Nakajima. Application requirements for middleware for mobile and pervasive systems. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):16–24, 2002.

[10] W. Reinhard, J. Schweitzer, G. Völksen, and M. Weber. Cscw tools: Concepts and architectures. *Computer*, 27(5):28–36, 1994.

[11] J. Roth. Seven challenges for developers of mobile groupware. Technical report, University of Hagen, Department for Computer Science, Hagen, Germany, 2004.

[12] J. Roth. The resource framework for mobile applications. In *Enterprise Information Systems V*, pages 300–307. Springer Netherlands, 2005.

[13] V. Sacramento, M. Endler, H. Rubinsztejn, L. Lima, K. Goncalves, F. Nascimento, and G. Bueno. Moca: A middleware for developing collaborative applications for mobile users. *Distributed Systems Online, IEEE*, 5(10):2–2, Oct. 2004.

[14] X. Su, B. Prabhu, C.-C. Chu, and R. Gadh. Middleware for multimedia mobile collaborative system. In *Wireless Telecommunications Symposium, 2004*, pages 112–119, May 2004.

[15] H.-L. Truong, L. Juszczyk, S. Bashir, A. Manzoor, and S. Dustdar. Vimoware - a toolkit for mobile web services and collaborative computing. In *Software Engineering and Advanced Applications, 2008. SEAA '08. 34th Euromicro Conference*, pages 366–373, Sept. 2008.

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