Reference: Fabio Bella, Filippo Forchino, Jarmo Kalaoja, Jürgen Münch, Alexis Ocampo, Mario Negro Ponzi, Marco Torchiano. Developing Services for the Wireless Internet, chapter Pilot Projects, pages 131-156. Springer, 2006.
DOI: 10.1007/978-1-84628-589-9_6; URL: http://link.springer.com/chapter/10.1007/978-1-84628-589-9_6

Developing Services for the Wireless Internet:

Pilot Projects

Fabio Bella[1], Filippo Forchino[2], Jarmo Kalaoja[3], Jürgen Münch[1], Alexis Ocampo[1], Mario Negro Ponzi[2], Marco Torchiano[4]
[1]Fraunhofer IESE
{Fabio.Bella, Alexis.Ocampo, Juergen.Muench }@fraunhofer.iese.de
[2]Motorola GSG Italy
{Filippo.Forchino, Mario.Negroponzi }@motorola.com
[3]VTT Electronics
Jarmo.Kalaoja@vtt.fi
[4]Politecnico di Torino
Marco.Torchiano @polito.it

Due to its recentness, the engineering of wireless Internet services lacks explicit experience based on quantitative data [1]. No historical data exist on related technologies, techniques, and suitable software development process models. Some of the most critical consequences to be expected are unreliable project planning, incorrect effort estimates, and high risk with respect to process, resource, and technology planning. Under these circumstances, the quality of the target application turns out to be very hard to predict.

This chapter presents examples of wireless solutions engineered following the approach discussed in the book. Two pilot projects are highlighted from different perspectives, such as engineering lifecycle, technology, and architecture. Special attention is given to the analysis of historical data gathered from the pilot projects [2], [3]. The chapter aims at giving managers and developers a sense of the behavior of projects in the wireless Internet domain by presenting the experience gathered through the development of pilot services within the context of the WISE project.

The pilot services were developed following the iterative and incremental process described in chapter 2. During each of the three iterations, the development focus was placed on a different set of requirements [8]. The first iteration took place during the period December 2001 – December 2002 (13 months), the second iteration during the period February 2003 – October 2003 (9 months), the third iteration during the period November 2003 – September 2004 (11 months). Figure 1 shows the time distribution of the three iterations.

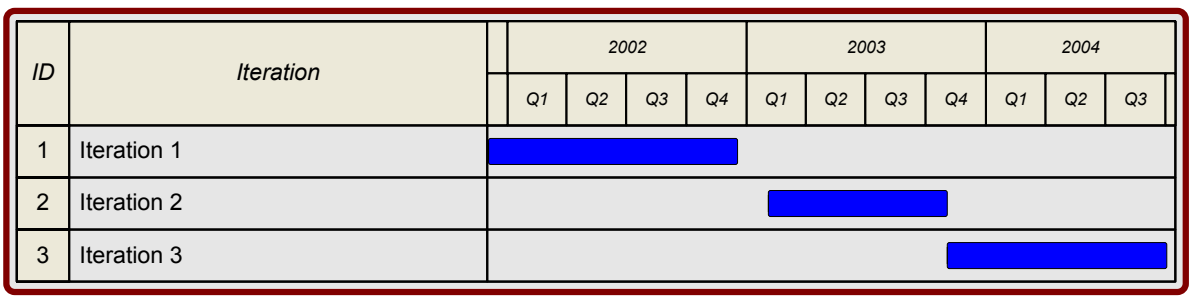

Figure 1: Iteration Overview

The pilot projects represent significant examples of the application of the concepts described in other parts of the book. The approach followed to collect and analyze the data is based on the process modeling methodology introduced in chapter 2 [4] and on goal-oriented measurement methods as discussed in [14]. Also, mutual relationships exist between this chapter and chapter 3 “Technology”, since technology issues represent one of the most important keys for understanding the behavior of the projects under observation [11]. The terminology introduced in chapter 4 “Architecture” is consequently adopted to show the high-level architecture of the pilot services.

Many of the results presented in this chapter are based on previous work published by the authors [5], [6], [7], [13].

This chapter is organized around two main sections, one for each pilot project. Each section contains a brief description of the pilot service, an overview of the process that has been applied for its development, a discussion of the historical data gathered during the development, a discussion of the main technologies applied, and a description of the high-level architecture. At the end of the chapter, a final section subsumes major domain-specific issues observed during the development activities.

6.1 Pilot 1

Pilot service one provides a solution for real time stock tracking on mobile devices: the user can view real time quotes concerning a whole market or define his/her own watch lists. The company responsible for this development is a provider of high-end trading services on the Internet, aimed at banks and brokers. The pilot was the adaptation of an existing Web-based information service. After developing the pilot, critical usability issues arose due to the huge amount of data needed by a financial operator to perform an analysis and the small size of the display of mobile devices. Furthermore, since the Internet traffic on mobile devices was paid for by the end user, based on data volume and not on connection time, and since frequent refresh of a large amount of financial data was required, the adoption of the push technology instead of the pull technology was an important issue, since it avoids unnecessary data refreshes for the user.

6.1.1 Applied process

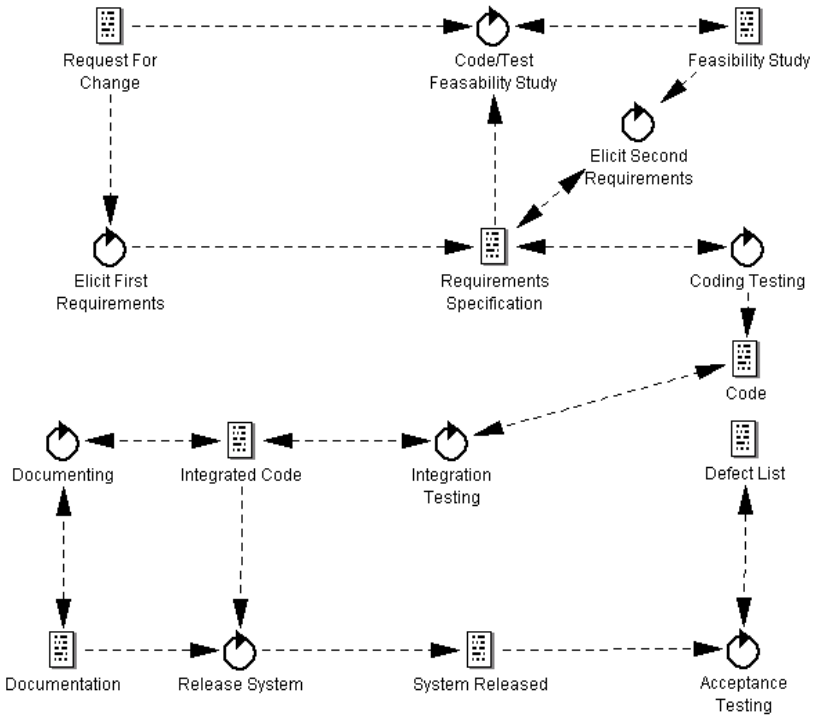

Figure 2: Overview process Pilot 1

Figure 2 shows the process used during each of the three iterations by the organization in charge of Pilot 1's development. The process was the product of tailoring the Reference Process Model presented in chapter 2 of this book. The Reference Process Model phases can be mapped to Pilot 1's process as follows: The *Requirements Phase* includes the activities *Elicit First Requirements*, *Code/Test Feasibility Study*, and *Elicit Second Requirements*; the *Coding Phase* only consists of the activity *Coding Testing*; and the *Testing Phase* includes the activities *Integration Testing*, *Documenting*, *Release System*, and *Acceptance Testing*.
Due to the fact that this organization develops its code on top of an existing application, the name of the customer requirements document is *Request for Change.* This document was analyzed in one meeting (Elicit first requirements). This meeting had the purpose of establishing what could be implemented and what should be carried out as a feasibility study (*Code/Test Feasibility Study*). The feasibility study was carried out to look for an answer to the following issues:
1. Select a set of mobile devices with the right display capabilities for fundamental data of stock information. In this application it is important to avoid that the user interacts by scrolling the data. Push technology was found suitable for this kind of applications.
2. Look at network issues (e.g., network type, bandwidth, coverage, etc.) and select a set of suitable devices.
With this information, sets of small prototypes were coded in order to test for the best suitable option.

With the results from the *Feasibility Study* and the original *Request for Change,* a new meeting was held with the objective of deciding which functionality could be implemented. After this, coding and unit testing was performed using as a basis the *Requirements Specification* and the prototypes coded during the *Feasibility Study*. An explicit design activity was not needed, since most of the design related issues were constrained by the already existing server infrastructure. Developers did unit testing in an informal way. There

was no test report document or standard where test cases, inputs, or outputs were described. Each developer was responsible for delivering the desired functionality on time. If there was a problem that the developer could not handle, then this was taken to an internal meeting where the results from the tests were discussed.

Once all of the code was ready, it was integrated and tested by developers and market experts (*Integration Testing*). The customer role was emulated with developers from other projects within the organization or market experts who were not involved during the application's development. These emulated customers performed informal test cases that were not documented. The results of these informal test cases led to finding defects or proposed ideas for improving the application. This report was also informal; it did not have a structure and was presented in an internal meeting, where decisions were made regarding the test results. Defects found as well as suggestions were reported informally. After this, a demo was packaged into the device and submitted to the market division for further tests (e.g., user interface interaction). At the same time, developers were writing the technical and user manuals (*Documentation*). There were different types of clients, and for each of them, the product had to be customized and packaged. A review by the market division approved releasing the system (*Release System*), which was finally tested by friendly end users for some time (*Acceptance Testing*), before its official final release.

6.1.2 Effort baselines

For the analysis, the effort data related to the single activities were consolidated according to the three main phases: *Requirements, Development,* and *Test* phase.

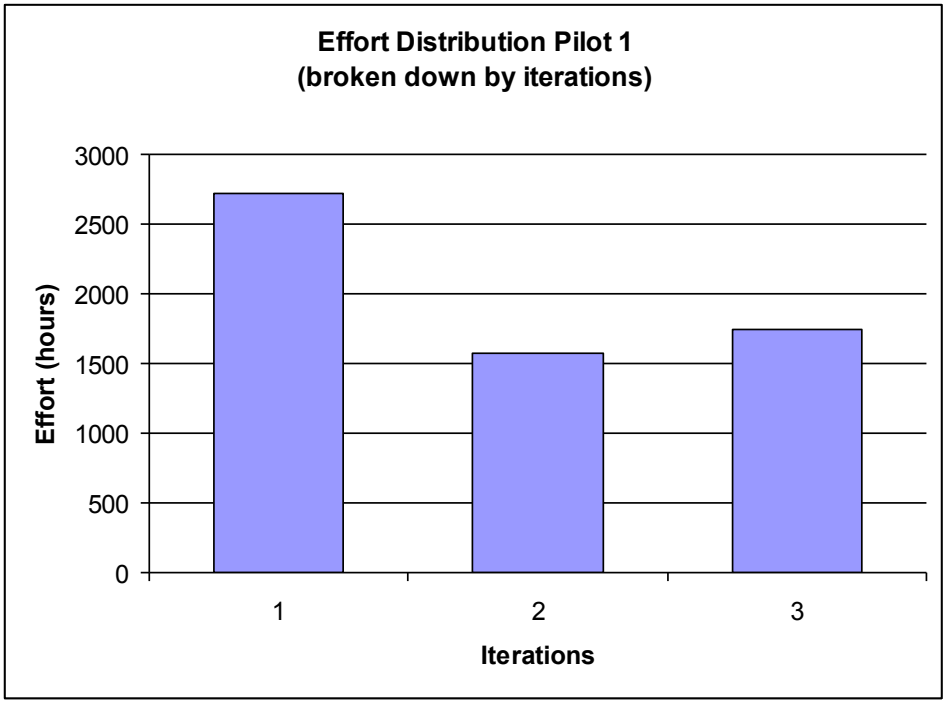

Figure 3: Effort Distribution (broken down by iterations)

Over 750 man-days of effort were spent on the overall development of the service. Figure 3 shows the distribution among the three iterations: the greatest effort was spent during the first iteration (about 340 man-days in 13 months). During this iteration, the existing server infrastructure had to be adapted to support the wireless version of the information service. The

first version of the wireless service was implemented in WML and derived from an existing HTML version of the service provided over the traditional Internet. During the second iteration, a Java version of the service was created for Blackberry devices. The choice of a full programming language like Java was motivated by the need for push functionality not supported by WML and by the idea to explore Java's graphical facilities. During the third iteration, the service developed in the second iteration applying Blackberry-specific libraries was modified to be able run on devices with a color display without the help of any proprietary library. First attempts to also run the service on UMTS devices in the context of the Italian market had to be given up due to the difficulties encountered when trying to deploy the client application on the devices: in the first half of 2004, only one network provider was distributing UMTS devices in Italy, and its policy was to protect them against the deployment of non-trusted applications. Regarding the effort spent on the last two iterations, the amount was comparable if we take into account that the second iteration was shorter than the third iteration (about 200 man-days in 9 months against about 220 man-days in 11 months).

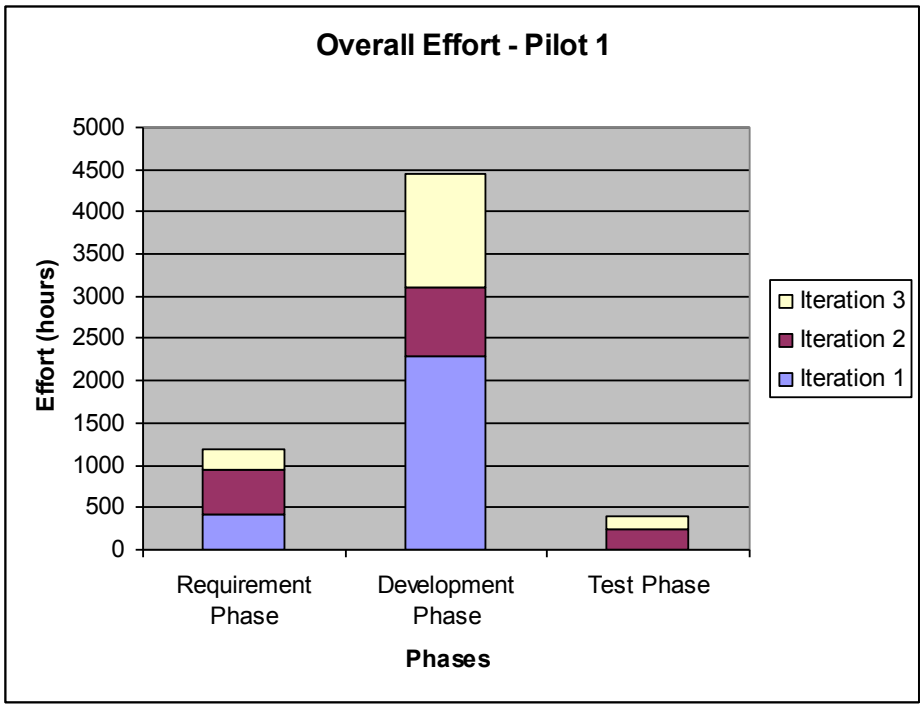

Figure 4: Effort Overview

Figure 4 shows the effort spent on the different phases during the whole project. During each iteration, most of the effort was spent on implementing the service (73% of the overall effort for the development phase). Relatively little effort was spent on the requirements phase (20%).This happened mainly due to two reasons: on the one hand, the functional requirements of the service were already known at the beginning of the project, since the same functionality provided by the existing Web-based version of the service had to be provided on the wireless Internet; on the other hand, due to the novelty of the technologies applied, the many non-functional requirements, which were mainly concerned with usability and performance-related issues, could often not be clarified before coding. During the second iteration, greater effort was spent on the requirements phase than in the first iteration (about 65 man-days). One of the main reasons was the introduction of time-consuming feasibility studies performed to explore the capabilities of Java-related technologies. This was necessary, since during the first iteration it was noticed that a great amount of the effort spent on coding did not produce any

tangible result but was motivated by the need to explore technology-related issues that could not be understood without coding. Therefore, feasibility studies were introduced to explicitly gather this kind of experience and to better understand the decision-making process concerned with the selection of requirements to be fulfilled. A total of about 65 man-days (i.e., over 8% of the overall development effort) was needed to perform the feasibility studies.

Concerning the test phase, the little effort spent on it (a total amount of about 50 man-days, approximately 7% of the overall effort) can be explained by the low complexity of the functionality required and the greater priority placed on exploring new technologies than on achieving a version of the service to be considered mature for the market. The last point led, for example, to the decision to even skip the test phase at the end of the first iteration due to deadline pressure. On the other hand, it should also be considered that some effort was spent by third organizations on evaluating the service, but this effort was not documented, since the evaluation did not happen within the scope of the WISE project: the organizations were interested in buying the client application and evaluated the business model underlying the service as well as the overall usability of the service.

6.1.3 Technology

The development of the pilot service began in the first iteration with a thin-client approach. Since the reference desktop client application is Web-based, it was natural to try to adopt the same design choices on wireless clients. It was considered the best approach because of the limited service interaction and because of the much larger base of WML-enabled devices (see also chapter 3). However, after the first iteration, differences and incompatibilities among wireless clients and the technological limitation of the thin-client approach suggested switching to a fat-client solution. In order to keep the installed client base large, the adopted technology was Java2 Micro Edition (J2ME™), as in Pilot 2. It was then necessary to build up a server-side adaptation layer (in order to push and pull data to and from the same server that works with desktop clients) to communicate with the new client. In the third iteration it was then possible to add full interaction, and the developed pilot is now a commercial product.

Technological constraints concerned with User Interface (UI) classes[1] and a market analysis suggested targeting specific devices that apply proprietary J2ME extensions[2]. While this approach enforces using only a specific family of devices, it is easy to port the pilot service on different devices provided the non-standard libraries are ported, too. This was not seen as an overwhelming task.

For Pilot 1, network bandwidth and latency did not represent a critical issue. This was particularly true during the second and third iteration, when wireless data networks were very reliable and stable. Wired Internet http connections are, in fact, comparably slow when used with home connections.

This pilot application demonstrated that the thin-client approach often turns out to be not applicable: In practice, its use should be limited to providing data lists with a very limited interaction. As soon as graphical feedback and bidirectional interaction become more important, problems with different screen size, the impossibility to precisely set the position and size of pictures, and the limited interaction force designers to switch to fat-clients.

[1] The J2ME standard does not provide any class for building complex tables. Because of the business-oriented origin of the Blackberry devices, the company added some proprietary extension to their J2ME implementation, which is targeted at handling data.

[2] Compare this approach with the one adopted by Pilot 2, which has developed in-house UI libraries (see later on in this chapter).

6.1.4 Architecture

In the following, the high-level architecture applied in the third iteration is described. A slightly different architecture was used during the first iteration due to the thin-client approach initially adopted. Compared with the architecture applied in the second iteration, the architecture presented should be considered an enhanced version fully supporting bidirectional interaction between clients and server.

Figure 5 depicts the system architecture (**structural viewpoint**). The *Blackberry MDS Proxy Server* is the server responsible for enabling access to the service by the wireless clients. It is also responsible for formatting the data used by the wireless clients. It supports bidirectional interaction between subscribers and the server and interacts with the same subscription server accessed by the wired clients. Client software is deployed on Blackberry devices and is an application coded in J2ME. The *WEB Download Server* allows downloading the application directly over the air.

The *User Web DB* is the same data base accessed by the thin client developed during the first iteration. The data base acts as a user management module that authenticates clients during the login phase.

The whole system is integrated with the pre-existing server environment called *Data Feed System*. The Data Feed System provides financial market variations (called *ticks* or *quotes*). Access to stock markets, orders, trades, and updates of user accounts/positions are handled by the *Trading System* (not shown in the figure).

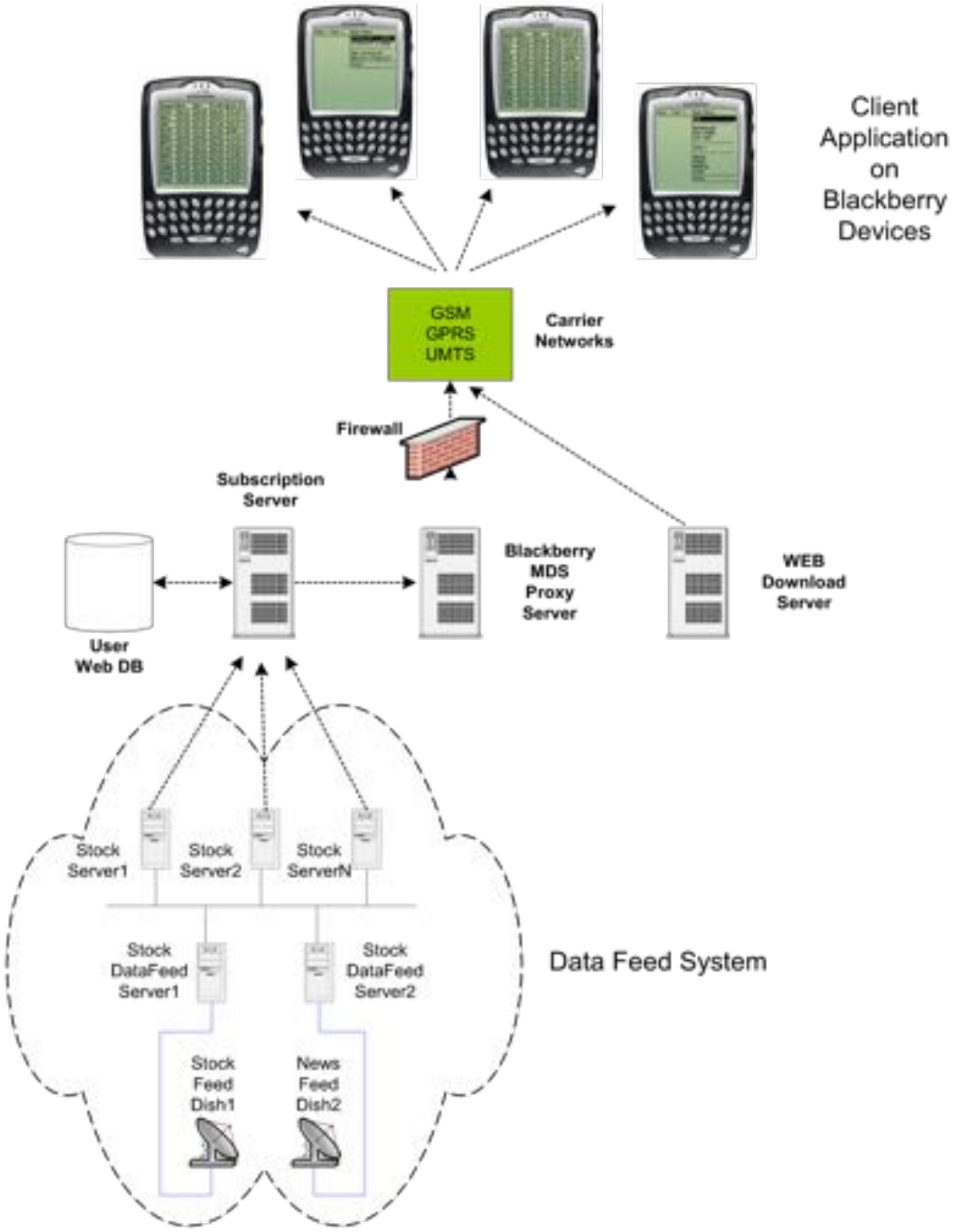

Figure 5: System architecture (structural viewpoint)

The **Domain Information Model** (depicted in Figure 6) shows the basic concepts used to model the application; these are common terms in the context of trading.

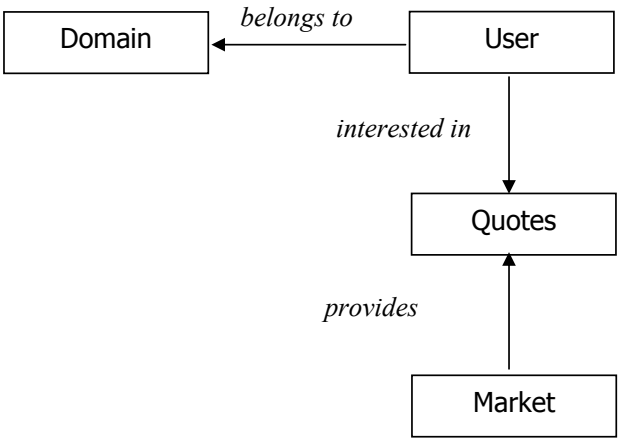

Figure 6: Domain Information Model

The information elements presented in the above figure are describe in Table 1.

Table 1: Trading Information Model Description.

Conceptual Element	Description
User	Uses stock exchange information from mobile device
Domain	The domain defines the context for user identification, authentication, authorization, and accounting.
Quotes	The quotes represent the values of stocks that come from the markets
Market	A market represents a real-world stock-exchange market, it provides updates of quotes

The main driver in the design of the architecture for this pilot was reusing as much as possible the pre-existing infrastructure, which supported a web-based service.
The functional structure of the pilot is presented in Figure 7. We can observe the typical subdivision into high-level domains as prescribed by the reference architecture. In addition we can see the services Quotes and News that belong to the Investnet domain, which is the back-end domain for the existing web-based application.
The responsibilities of the functional elements are presented in Table 2.

Table 2: Responsibilities of functional elements.

Conceptual Element	Responsibility
User Service	Provides mobile stock exchange service for mobile users on a Blackberry device.
Subscription Service	Manages the business logic of Pilot 1 service, i.e. enable client access and handles quotes information.
User Authentication	Takes care of authentication, security and user classes.
Quotes Service	Provides Stock Quote information

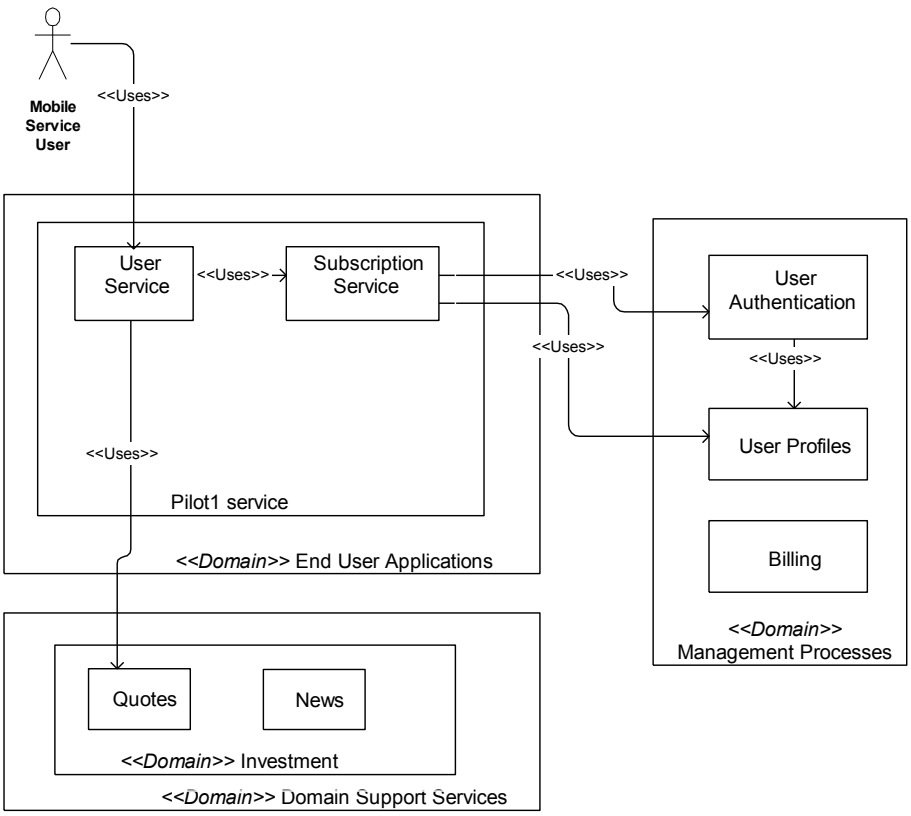

Figure 7 : Pilot 1 functional structure.

The deployment of the functional elements onto the network nodes is presented in Figure 8. The services Quotes and News are located on a back-end node that provides the information to the whole system. The management services and the Subscription service are located on a server node that is shared with the web-based service.
The wireless specific part of the service is located on a dedicated access node that communicates with the mobile device (where the end-user service is located) through a data service. The data service is specific for the device chose for the implementation.

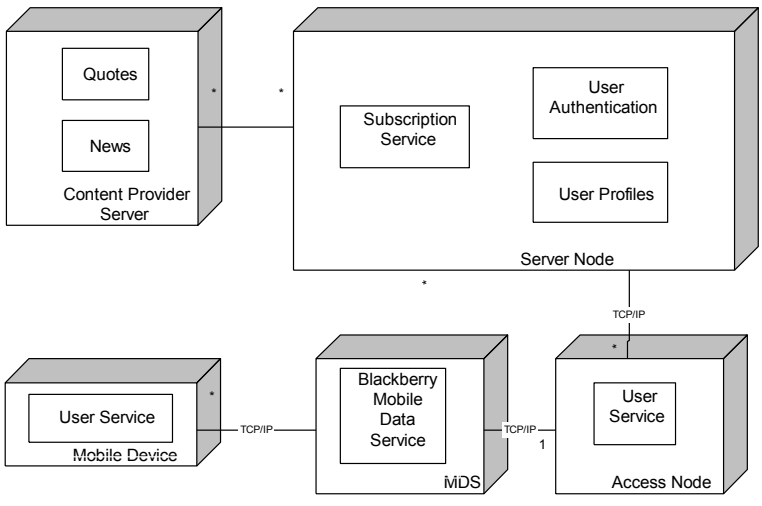

Figure 8: Deployment of Pilot 1.

6.2 Pilot 2

The second pilot service was concerned with the development of a multi-player online game for mobile devices. It features real time interaction among many players, who share a virtual environment, i.e., a fantasy world map. Players can collect items and transport them to other locations, chat, and fight against enemies and against each other. A player's character can also develop during the game: victories in battle or the completion of special missions award them experience points that can be used to upgrade their attributes (i.e., speed, strength, dexterity, etc.).

Games of this kind are popular in the wired Internet world where they are commonly called Massively Multiplayer Online Role Playing Games (MMORPG). Their success lies in the detailed implementation of a huge virtual world shared by many players (in the order of hundreds) who are given just adventure clues and not a predefined path to follow. Being a Role Playing Game, most of the fun and addictiveness lies in the development of a player's character (search/trade of rare and powerful items for facing mighty opponents or just for collector's fever).

In the mobile world, this kind of game has not yet boomed, though an undeniable interest of operators and gaming companies exists. Even if multiplayer games are considered by many the next step in wireless gaming, several reasons have been postponing the date of a massive release on the market:
- Technical limitations of the mobile terminals and the data network radio access
- Huge amount of data exchange (due to the real time interaction) combined with the relatively high cost of data traffic fees for the end users
- High cost of game maintenance, since such games require powerful servers able to guarantee a good quality of service 24/7 to many users.

The development of the pilot service was distributed between two different teams/organizations: one organization was responsible for the development of the client on the mobile device (focus on usability for the end-user); the other organization took care of the server side (focus on easy customizability of the service). As in every case of development involving two partners, considerable effort has been spent on the definition of the interfaces between the two components, in this case the client-server game communication protocol.

6.2.1 Applied process

Two different processes were followed for engineering the client and the server side of the pilot service. The underlying process models were tailored from the reference process model presented in chapter two.

The Reference Process Model phases can be mapped to Pilot 2's client development process as follows: The *Requirements Phase* includes the activities *Gather Requirements*, *Feasibility Study*, and *Analyze Requirements*; the *Design Phase* consists of the activities *System Design*, *Design*, and *Design Review*; the *Coding Phase* consists of the activities *Code, Unit Test, Integrate Code, and Release Code*; and the *Testing Phase* includes the activities *Plan Tests*, *Build Test Framework*, *Test System*, *Test Usability*, *Acceptance Test, and Analyze Defect*.

The organization in charge of the development of the client followed the traditional software development life cycle phases (see Figure 9). However, there were activities that were performed specifically for the development of wireless Internet services.

The requirements were gathered as two types: Request for change of existing services, or development from scratch of new services. This information was written in the *Requests From Customer*, and usually demanded the use of innovative user interface interaction. This is a challenge for any organization, because there is little information on what type of libraries, or COTS products, or open source solutions is the most appropriate one in order to develop a user-friendly wireless Internet service. Therefore, research for the best possibility was done through an activity called *Analyze User Interface Feasibility Requirements*. The idea was to validate the user interface interaction requirements with the capabilities of the existing libraries developed by the organization or external resources. Once this analysis was done, a decision was made on accepting the user interface requirements or rejecting them. Sometimes, the feasibility of these requirements was not clear. Therefore they were further analyzed in another activity called *Feasibility Study*. This study was different from the previous one, because it covered other important aspects of wireless Internet services such as wireless networks, bandwidth, quality of service, and security, among others (see chapter 1). Once all the technical issues were resolved, the requirements were specified in the *Software Requirements Specification,* which was used for the *Design Phase*. The design of the service on the client side was also affected by the characteristics of wireless Internet services. An optimization of this was needed in order to improve the performance of the application (e.g., game server).

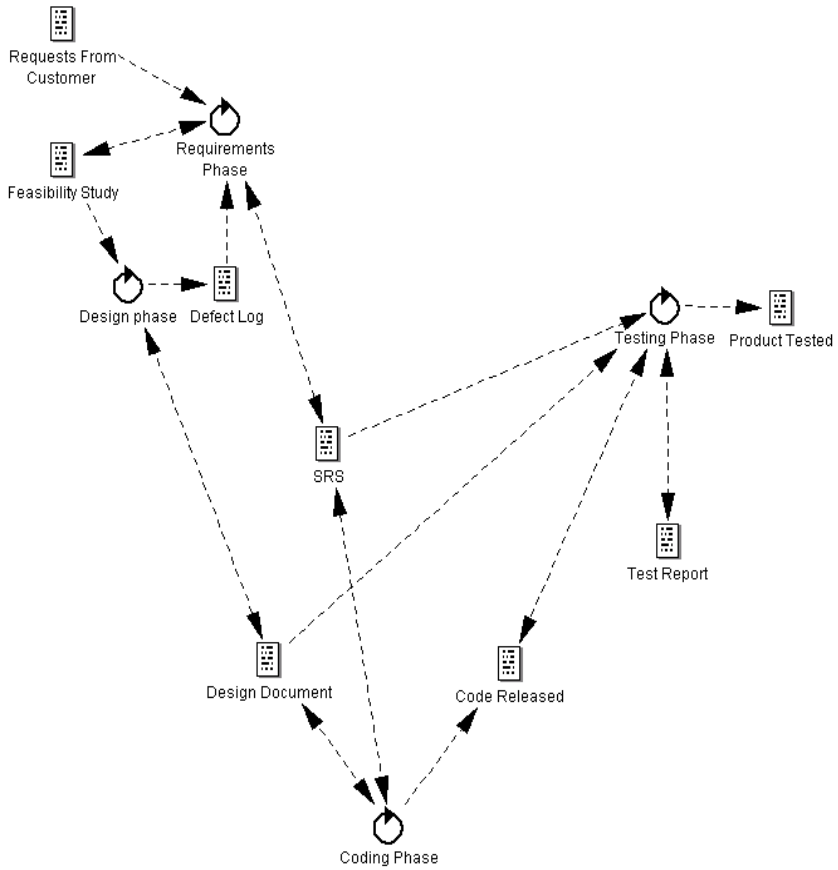

Figure 9: Overview process Pilot 2 client

After this, the code was produced, unit tested, integrated, and released (*Coding Phase*). The client developers performed their unit tests using the "J2ME Wireless Toolkit 2.1". This is a client emulator that allows running midlets on top of it, and emulates the functionality of a

mobile device. Specific components such as network modules were tested on cell phones, since they were critical for the rest of the application. The graphic library components were tested separately before including them in the application. The client part contained four modules: The communication module, the game engine module, the interface module, and the data storage module. They were tested separately and then integrated. The set of communication components were called the "transport layer". In order to test the transport layer, a server emulator ("server stub") had to be developed. The interface module component integration tests were performed using the "J2ME Wireless Toolkit 2.1".

The complete functionality of the system (client and server) was tested using the real environment (*Testing Phase*). No emulators were used during these tests. If a defect was found, the parties discussed it and assigned a person responsible for fixing it. Once the defect was fixed, it was declared a closed defect. A big concern of pilot developers was the use of guidelines or rules to test the usability of the user interface. The activity *test usability* was introduced as a result of this concern. This activity is based on Nielsen's approach [12], in which experts guided by a set of usability principles known as heuristics evaluate whether user-interface elements such as dialog boxes, menus, navigation structure, online help, and others conform to user needs.

The Reference Process Model phases can be mapped to Pilot 2's server development process as follows: The *Requirements Phase* is the *Exploration Phase*; the *Design Phase* consists of the activities *Analysis and Design*; the *Coding Phase* consists of the activities *Programming and Continuous Integration*; and the *Testing Phase* consists of the activity *Test System*.

An initial set of requirements was gathered from the organization developing the client side. The Pilot 2 server development team familiarized itself with the tools, technology, and practices they were to use during the project. The requirements were prioritized and an overall schedule was written (*Exploration Phase*).

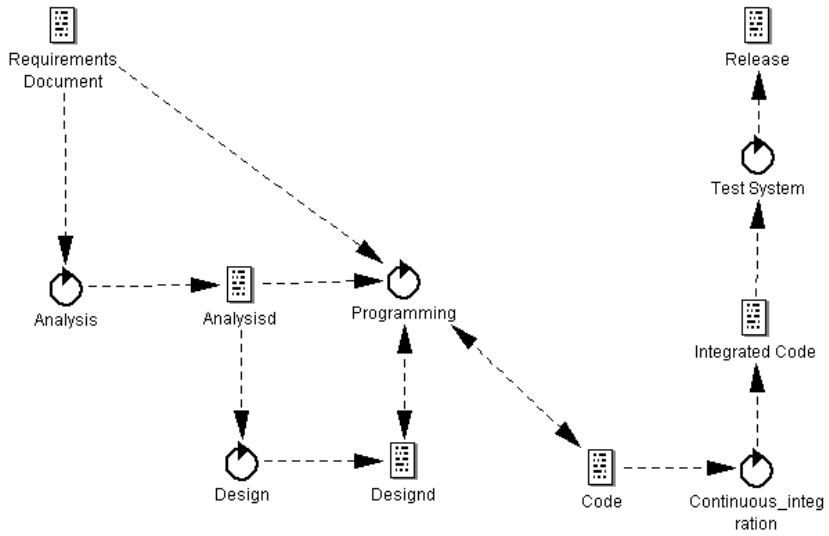

Figure 10: Overview process Pilot 2 server release phase

Figure 10 shows the activities followed for releasing the functionality. A coarse sketch of the release was made during *Design*. Development was done in such a way that any implemented new functionality was integrated into the code base as soon as it was ready. Immediately after the new functionality was integrated, the complete server part was tested.

6.2.2 Effort baselines

In the following, effort data gathered from the development of the client and the server parts are first presented separately; then, aggregated data are discussed in order to analyze the overall effort spent on Pilot Service 2.

About 420 man-days were needed to develop the client side of the pilot service. Figure 11 shows that a comparable amount of effort was spent during the three iterations: during the first iteration, about 140 man-days were spent in 13 months; during the second iteration, about 125 man-days in 9 moths; and during the third iteration, about 150 man-days in 11 months.

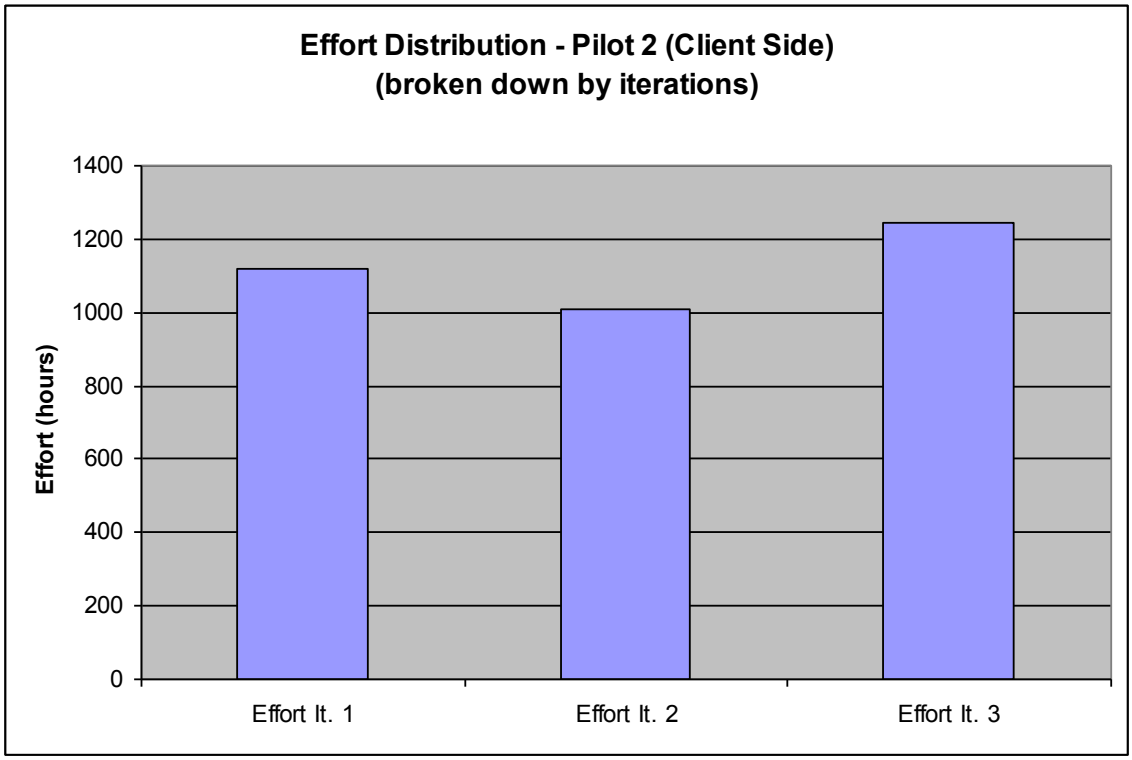

Figure 11: Effort Distribution - Client Side (broken down by iterations)

Figure 12 shows the same effort data broken down by phases. During each iteration and therefore during the whole project, most of the effort was spent on coding (about 50% of the overall effort), whereas the design phase took the least effort (about 14%). The requirements and the testing phase required a comparable amount of effort (the requirements phase took 80 man-days, 19% of effort; test phase 70 man-days, 17%).

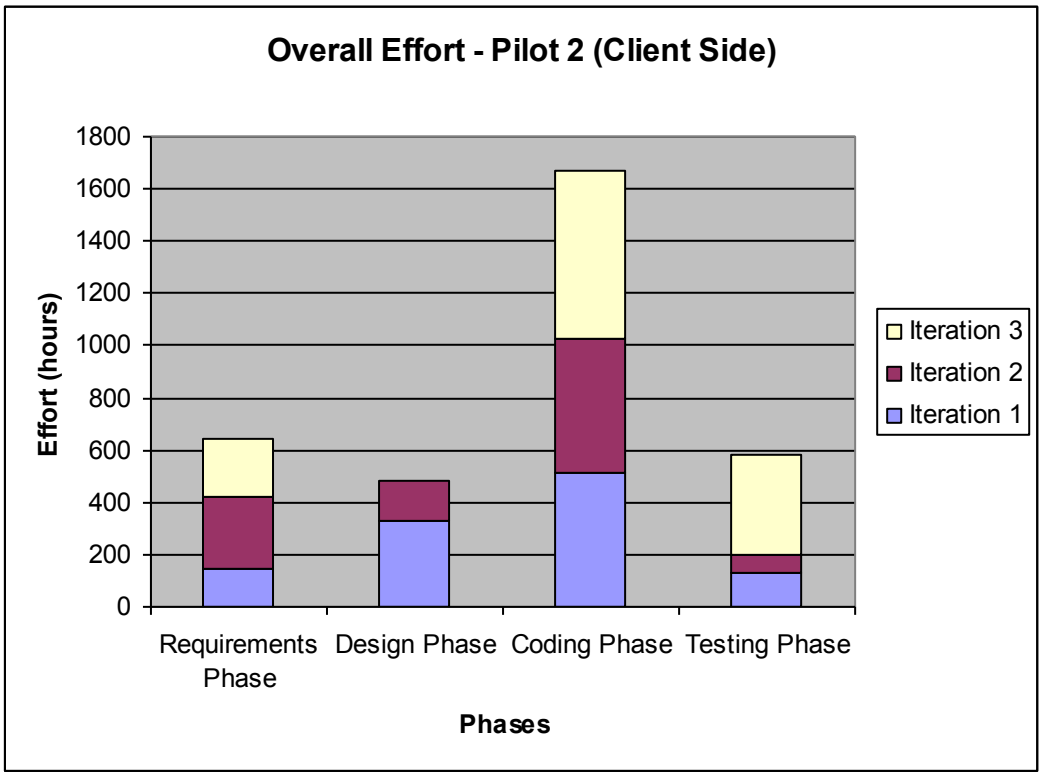

Figure 12: Effort Distribution - Client Side (broken down by phases)

For a more detailed analysis of the requirements phases, it should be mentioned that most of the requirements were specified and prioritized during the first iteration. After that, the involved parties agreed at the beginning of each iteration on a new set of requirements to be addressed, and further specification was done verbally and by email. The greater effort spent on the requirements phase during the second and the third iterations can be explained with the introduction of feasibility studies starting from the second iteration. This was needed, since, similar to the case of Pilot 1, it became clear after the first iteration that many technology-related issues could only be clarified through coding. A total of about 30 man-days (i.e., over 7% of the effort spent on the development of the client side) was needed for feasibility studies.

Concerning the design phase, it can be noted that less effort was spent on design in every new iteration, which can be explained by the fact that many of the most relevant design-related decisions were already made after the first iteration. Although design-related activities were performed during the last iteration in order to address design issues concerned with the communication between client and server side, this did not lead to any change in the main design document and the negligible resulting effort was not collected as design-related but as coding-related, since the problems were informally addressed through verbal communication and immediately solved through changes in code.

The coding phase was carried out in a systematic way according to the prioritized requirements and included automated unit testing. A comparable amount of effort was spent on this phase during each iteration (between 60 and 80 man-days per iteration).

Regarding system test, about 17% of the effort was spent on this phase. Most of testing was performed at the end of the third iteration, when the whole system was ready. Deadline pressure and the high priority placed on the exploration of available technologies were

additional reasons for shifting the main test activities to the end of the project. Another important reason was the intrinsic difficulties of testing applications on mobile devices: on the one hand, testing on real devices is extremely time-consuming due to the lack of automation; on the other hand, emulators are not reliable substitutes of real devices. As an inevitable consequence, the testing process turns out to be difficult to estimate and to plan. The situation is even worse if the application should run, and therefore should be tested, on different devices.

Concerning the development of the server side, this undertaking took about 260 man-days. Two different organizations were involved in the development: the organization first responsible for the server left the project after the end of the first iteration, therefore, another partner assumed the role for the remaining two iterations. Figure 13 shows that most of the effort related to the server side was spent during the first iteration (135 man-days, 52% of effort). At the end of the first iteration, the basic infrastructure, an Enterprise Java Beans server, had been set. The same infrastructure was deployed during the second iteration on servers hosted by the other partner and enhanced during the two iterations to support all requirements formalized at the beginning of the first iteration.

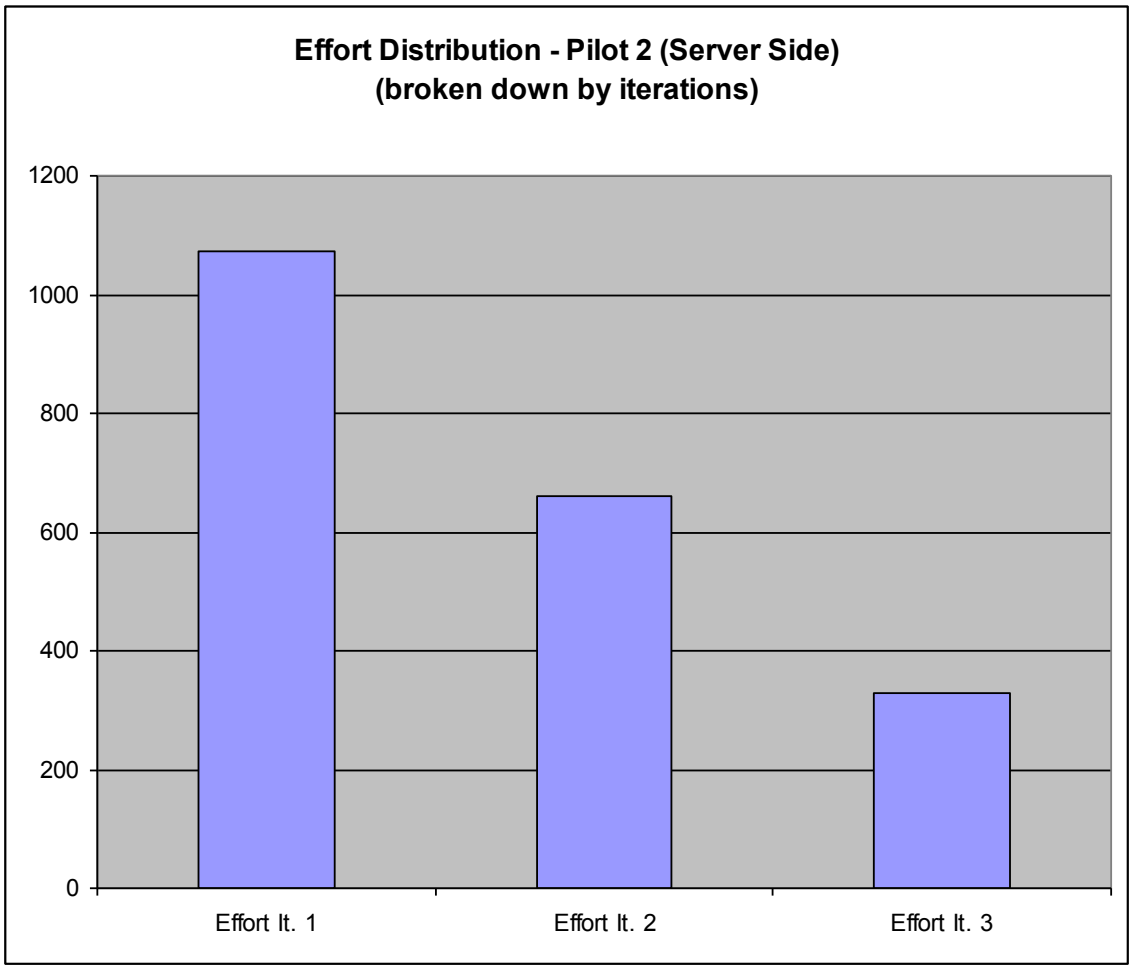

Figure 13: Effort Distribution - Server Side (broken down by iterations)

Figure 14 shows the distribution of the effort spent on the server side among the development phases. Since the whole development was driven by the organization responsible for the client side, very little effort was spent on the requirements and on the testing phase (about 10 man-days, 4% of effort, per each phase). According to the data related to the development of the

client side, in this case there was again less effort spent on design in the last two iterations. The main effort was spent on coding-related activities (160 man-days, 62% of effort).

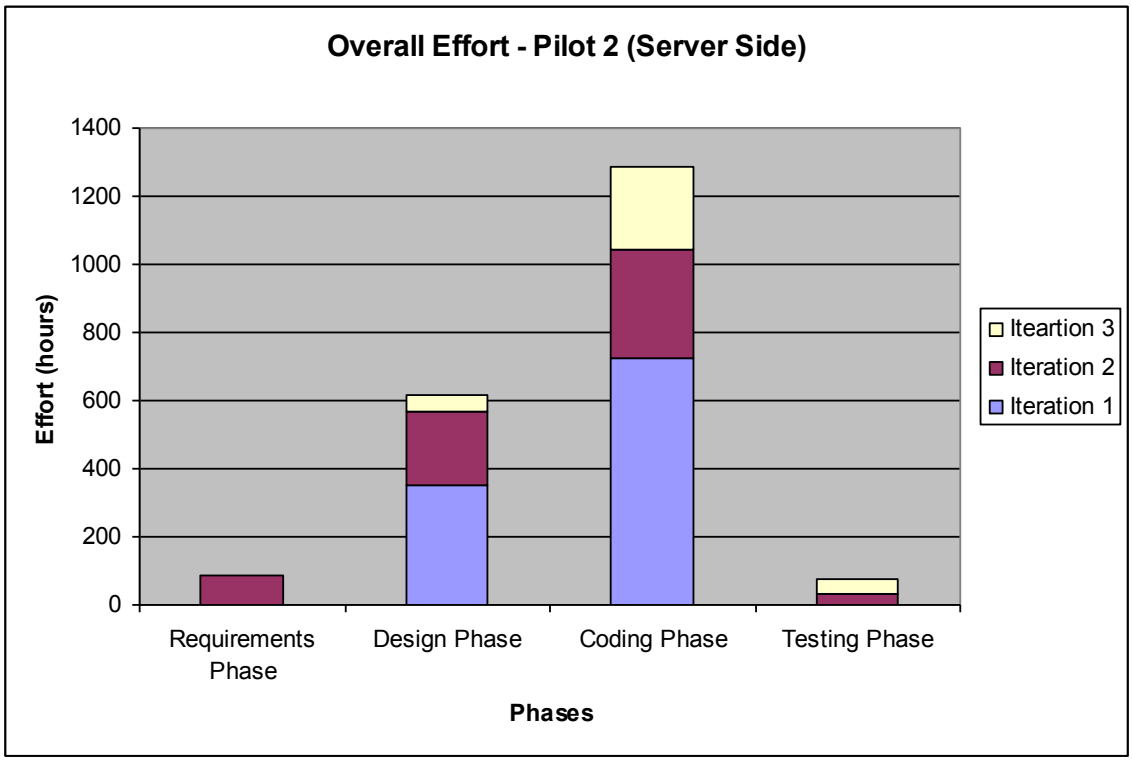

Figure 14: Effort Distribution - Server Side (broken down by phases)

A similar picture is given in Figure 15 where the involvement of the organizations becomes clearer: although the whole development was driven by the organization responsible for the client side, comparable efforts were needed to address design-related issues and to implement the service.

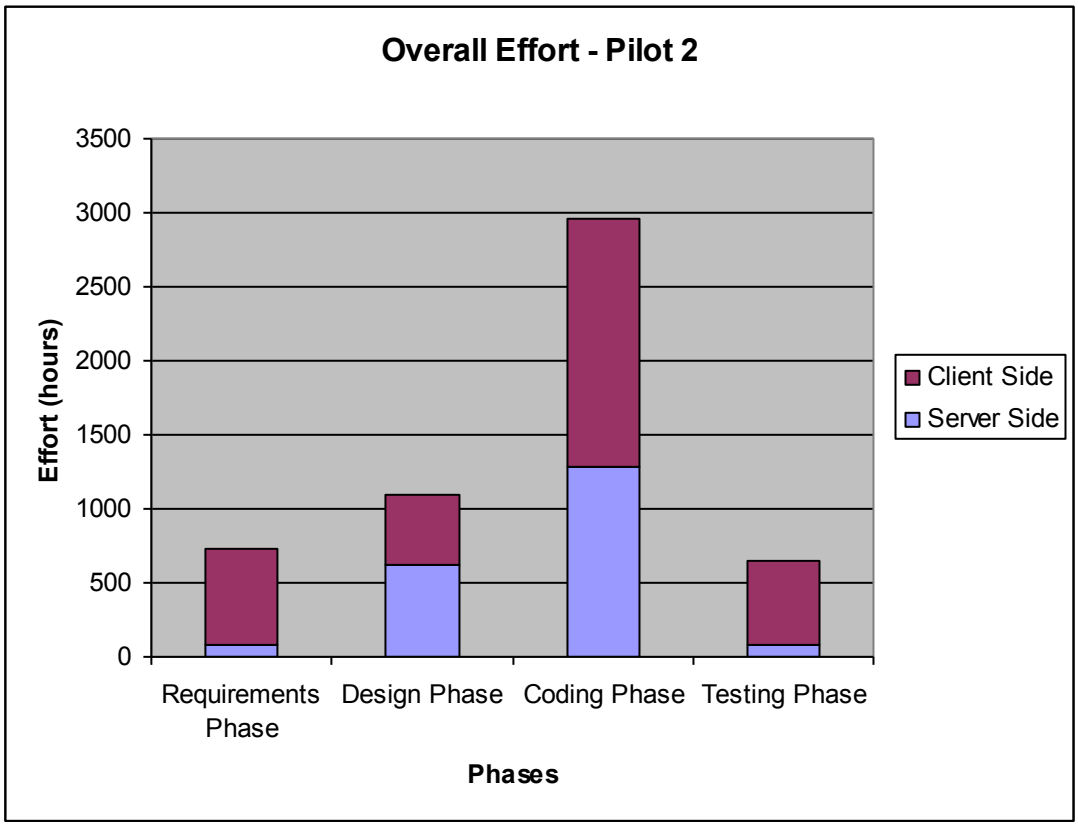

Figure 15: Overview Organizations' Involvement

To achieve Pilot Service 2, a total of about 680 man-days were required. Figure 16 shows the distribution of the overall effort among the four main phases: Most of the effort was spent on coding (55%); design-related activities required about 20% of the work; whereas the requirements and the test phases together took up the remaining 25%.

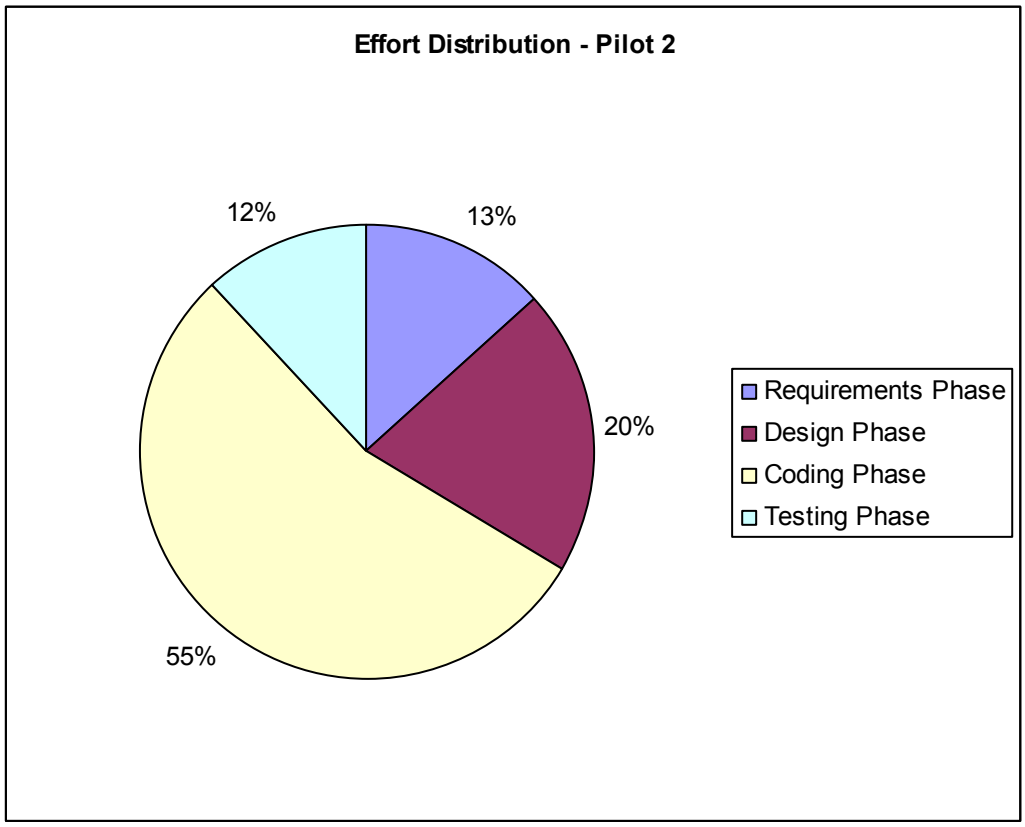

Figure 16: Effort Distribution - Pilot 2 (broken down by phases)

6.2.3 Technology

The features of the service are such that the choice of a fat-client approach is practically mandatory. The game needs a complex user interface, with many different screens that must dynamically react to user actions. The main game screen, for example, shows a player-controlled character moving on a map, together with monsters or other players' characters; players should be easily allowed to do a multitude of actions ranging from chatting to fighting, from browsing their inventory to checking out the game hall of fame.

Figure 17: Sample screenshot of Pilot Service 2

The usage of a fat client also allows the usage of multimedia effects like sound effects, music, animated and full screen graphics, since the application has full control of the device display.

The service is organized as a traditional client/server application where multiple clients running on mobile devices connect to a multi-threaded server listening for connections on the Internet. Mobile devices may connect to the Internet in several ways (i.e., GPRS radio access, UMTS, Wi-Fi, etc.) but this is absolutely transparent to the two parties, since they both simply rely on a TCP/IP connection. They communicate through a custom binary communication protocol that has been designed to be simple and minimal.

On the client side, J2ME has been selected as the development language. The choice was driven mainly by the fact that it is a widely supported standard. Though manufacturers distribute their own proprietary extensions to J2ME, only the standard APIs were used in this project, in order to ensure maximum portability of the code. The lack of "nice", game-oriented User Interface libraries made it necessary to develop a lightweight, extendable GUI library. This library was then used to create the dialogs of the client application.

The target devices are mainly mobile phones, characterized by a relatively small display and limited processing and multimedia capabilities. The user interface treats screen width and height as variables, rearranging the display of text and graphics dynamically.

Multimedia effects (sound, music and animation) are also extensively used. The target devices' multimedia capabilities usually include:
- MP3/Wave sound support (for sound effects or sampled music),
- MIDI music support (for FM music).
- PNG is the only supported graphic format (J2ME standard, on MIDP 2 JPEG can also be supported)
- Vibration and potentially also embedded camera

On the server side, the natural choice was to use a similar Java technology: Java 2 Enterprise Edition (J2EE). This allows some common code to be shared between client and server or to be reused (i.e., the communication protocol library). The server is integrated with an external Service Management Component, which is used to perform common tasks like authentication or service subscription.

6.2.4 Architecture

The networked environment (structural viewpoint) for the game (shown in Figure 18) is presented next. Clients had access to a GPRS (or UMTS) network connected to the Internet by means of a GGS Node. Having this access to the Internet, clients were able to connect to the Game Server. The Game Server made use of a series of Management Services (such as authentication and authorization), which were provided by other servers. The system context affects the software architecture mainly as a rationale for the structuring of architecture and defines the nodes used in deployment views.

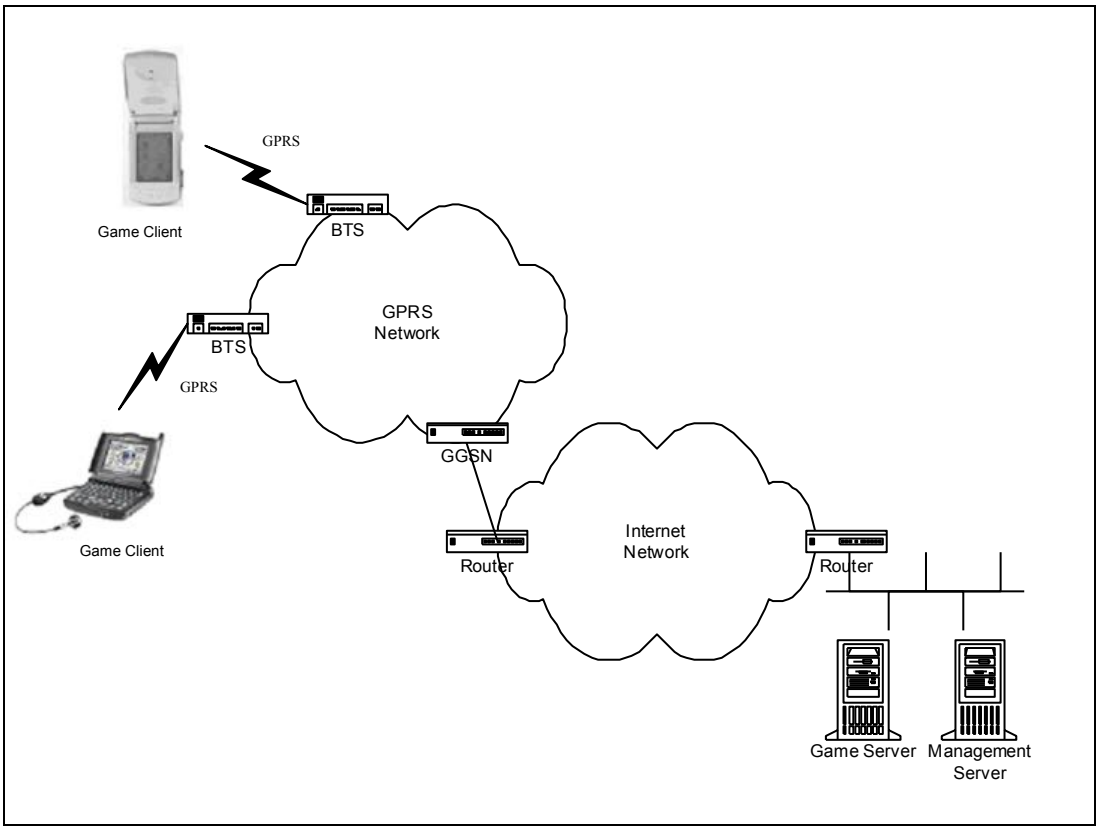

Figure 18: System architecture (structural viewpoint)

Figure 19 shows the Domain Information Model concerning the game. The model defines the terms of the domain and is important for the design of data shared in architectural interfaces.

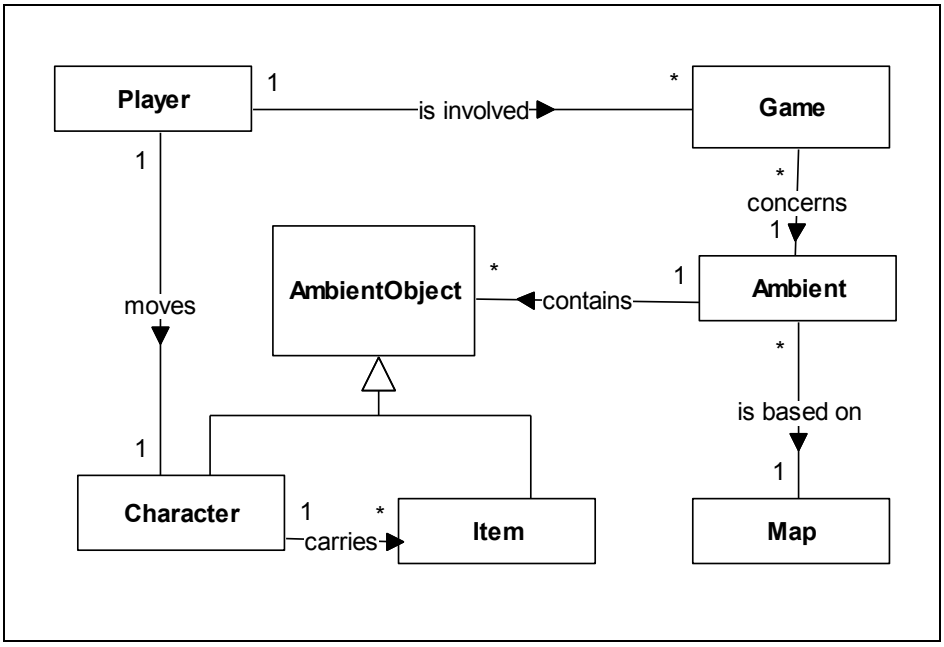

Figure 19: Game Domain Information Model

The objects presented in the figure are described in more detail in Table 3:

Table 3: Game Information Model Description

Object	Description
Player	A Player represents a user of the game service. An instance of this class is created when the user subscribes to the service, and it is removed when the user unsubscribes from the service.
Game	A Game represents the fact that a player is playing the game in a specific Ambient. For each Ambient only one game could exist for a player. An instance of this class is created when a user requests a new game, and it is destroyed when the game ends. A game ends when the Character the player uses in the Game is dead. A game can be suspended to be resumed later.
Ambient	An Ambient is an environment where a set of users plays their own game moving their own character. An Ambient consists of: - a map - a set of items placed in the environment described by the map, and - a set of characters acting in the environment described by the map
Map	A map is the static description of an Ambient. It is the landscape where the game takes place, e.g. a castle, a forest, a dungeon, etc.
AmbientObject	An AmbientObject is an object that is contained in an Ambient. Both Character and Item are sub-classes of AmbientObject.
Character	A Character is an entity controlled by a player, which moves and acts in an Ambient. An instance of this class is created when the player requests a new character, and it is destroyed when the character dies.
Item	An Item is an object (e.g., a treasure, a weapon, a spell book, etc.) that is present in the Ambient. An object could be carried by a Character.

One of the main goals achieved was to identify the generic application domain services that are common for different kinds of computer games (and possibly other entertainment services). These generic services can be reused as a platform for different game applications. The game conceptual model from the first iteration is shown in Figure 20. The architectural model also shows the main technology choices on both the client and the server sides. An opportunity for using a previously developed generic transport service was also identified. In architecture modeling at the conceptual level, the emphasis was on understanding the relation between the game client and the server and the services available from the service management domain of the reference architecture.

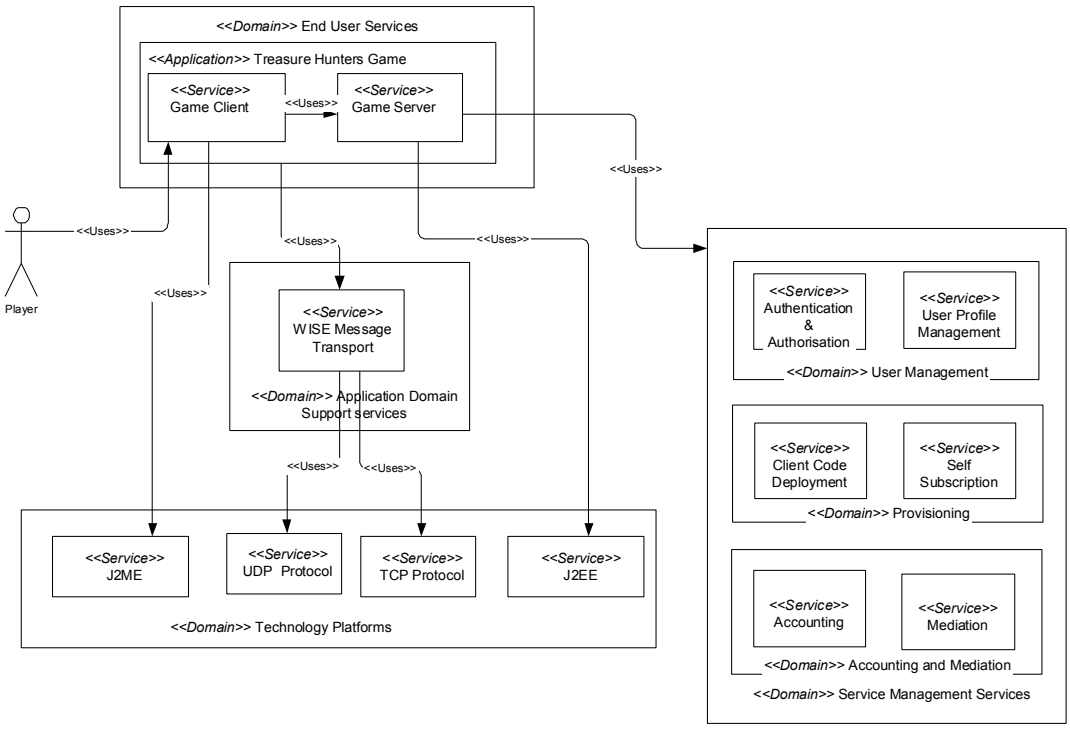

Figure 20: Game Conceptual Structural Model

The corresponding conceptual deployment model of these services and its behavioral view are presented in the notation examples of the Architecture chapter. Later iterations of the game architecture added services that are needed by two new stakeholders, i.e., a game designer and a game manager, and identified potential generic application support services for the mobile gaming domain of the reference architecture (Figure 21).

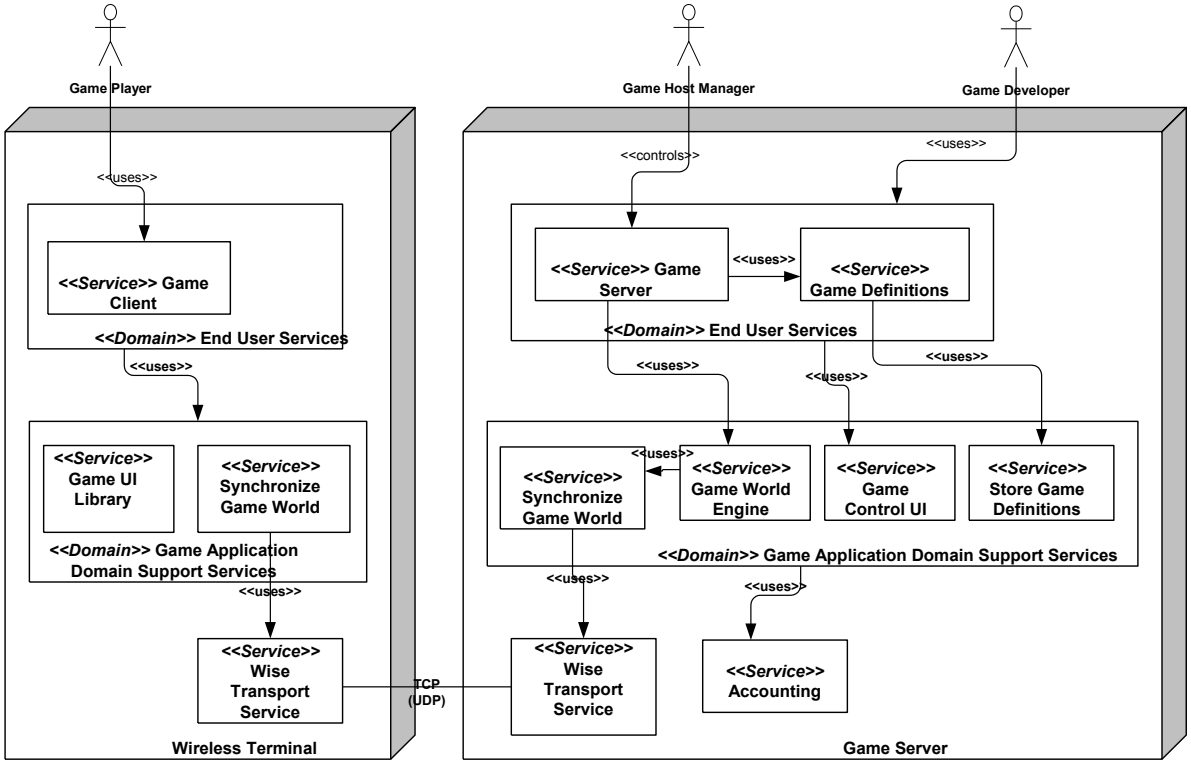

Figure 21 Game client and server detailed Conceptual Deployment Model

The objects presented in the figure are described in more detail in Table 4:

Table 4: Conceptual model element descriptions

Conceptual Element	Description
The Game Client (i.e., Player game world)	The game status of region of game world as visible to a single player in wireless terminal.
Game UI library	A gaming-oriented lightweight GUI library.
The Game Server (i.e., Server Game World)	The status of the whole game world at the server. The information model of the game world is based on the domain information model of the game. Provides a GUI to control and monitor the game in progress.
Synchronize game world.	Handles session management and synchronization of game status between player game worlds and the server game world via wireless connection. This is based on an application level messaging protocol that exchanges domain information model based information, using broadcasting and a publish-subscribe design pattern.
Game Definitions	The game definitions (maps, items, monsters, etc.) for the treasure hunters game.
Game World Engine	Calculates the results of player actions for player, the game world and other players.
Store Game Definitions	Stores adventure game definitions (maps, items, monsters, etc.) in various formats (ASCII file formats like XML, Java classes, etc.). Monster behavior is implemented using strategy patterns.
Wise Transport service	A messaging protocol via a wireless TCP or UDP connection.
Accounting	Authentication of players and storage of player profile information. A proxy service is provided by the service management component.

6.3 Conclusions

The wireless services discussed in this chapter represent good examples of the revolutionary applications that will be available on new generation mobile devices: the first service enables stock tracking and exchange in real time anywhere; the second service allows several players to share the same game environment and introduces the multi-player paradigm to the world of games for mobile devices.

In both cases, only a fat-client approach could provide the level of flexibility needed for the development of eye-catching applications characterized by a high level of interaction with the user.

Of course, the data gathered from the development of both pilot services came from specific contexts and cannot be considered universally valid. On the other hand, little data concerning typical effort baselines has been published and, due to the novelty of the field, no data is available that is directly related to the engineering of wireless Internet services. Applying the baselines presented in this chapter requires a careful comparison of the context surrounding the data discussed and the context surrounding the service to be engineered. Whenever possible, organization-specific historical data should be compared with the baselines presented here and reasons for divergences should be determined; also, the potential impact of the resulting influence factors on the new project should be investigated and constantly monitored.

Nevertheless, some aspects seem to be of particular importance, since they were observed in both pilot projects and viewed in a similar way by the involved parties. In all cases observed, less effort was spent on design than on coding. Also, most of the effort was spent on coding-

related activities. This fact was understood as a consequence of the uncertainty inherent in this kind of projects and the many unexpected issues that could only be discovered during coding. Feasibility studies took up to 8% of the overall effort, and this data should be considered an underestimation of the real project behavior, since during the first iteration the effort spent on feasibility studies was collected as coding. Our experience showed that feasibility studies represent a valid means for investigating requirements and documenting related decisions also in the case of the wireless Internet. The percentage of effort spent on testing (between 7% and 12% of the overall effort) would probably turn out to be too low in the case of a mature service to be considered ready for the market. Also, consider that testing turned out to be the most difficult phase to estimate and plan due to the extremely time-consuming testing on real devices and the unreliability of emulators.

In general, the choice of an iterative and incremental life cycle was considered the only one possible in a field governed by uncertainty. Also, explicit process modeling together with data collection performed in accordance with the resulting process descriptions represented a viable and fruitful way for gathering high-quality lessons learned, since the facts analyzed during post-mortem sessions were based on measurable evidence. This helped to get a funded understanding of project behavior.

6.4 References

[1] Basili, V.R, Quantitative Evaluation of Software Engineering Methodology, in Proceedings of the First Pan-Pacific Computer Conference, Melbourne, Australia (1985).

[2] Basili, V.R., Caldiera, G., Rombach H.D.: The Experience Factory, in Encyclopedia of Software Engineering (John J. Marciniak, Ed.), John Wiley & Sons, Inc., Vol. 1, pp. 469-476 (1994).

[3] Becker-Kornstaedt, U., Boggio, D., Muench, J., Ocampo, A., Palladino, G.: Empirically Driven Design of Software Development Processes for Wireless Internet Services. Proceedings of the Fourth International Conference on Product-Focused Software Processes Improvement (PROFES) (2002).

[4] Becker-Kornstaedt, U., Hamann, D., Kempkens, R., Rösch, P., Verlage, M., Webby, R., Zettel, J.: Support for the Process Engineer: The Spearmint Approach to Software Process Definition and Process Guidance. Proceedings of the Eleventh Conference on Advanced Information Systems Engineering (CAISE '99), pp. 119-133. Lecture Notes in Computer Science, Springer-Verlag. Berlin Heidelberg New York (1999).

[5] Bella, Fabio; Münch, Jürgen; Ocampo, Alexis: Baselining Wireless Internet Service Development - An Experience Report. In: Brito e Abreu, Fernando (Ed.) u.a.:5th Conference for Quality in Information and Communications Technology, QUATIC 2004 - Proceedings. Porto: Instituto Portugês da Qualidade, 2004, 161-169: Ill., Lit.

[6] Bella, Fabio; Münch, Jürgen; Ocampo, Alexis: Capturing Evidence From Wireless Internet Services Development. In: O'Brien, Liam (Ed.) u.a.:11th International Workshop on Software Technology and Engineering Practice. STEP'2003 - Proceedings. Los Alamitos : IEEE Computer Society, 2004, 33-39 : Ill., Lit.

[7] Bella, Fabio; Münch, Jürgen; Ocampo, Alexis: Observation-based Development of Software Process Baselines: An Experience Report. In: Arbeitskreis Software-Qualität Franken e.V.:CONQUEST 2004. 8th Conference on Quality Engineering in Software Technology - Proceedings. Erlangen, 2004, 31-43: Ill., Lit.

[8] Boehm, B.W.: A Spiral Model for Software Development and Enhancement, IEEE Computer, Vol 21, No 5, pp. 61-72 (1988).

[9] Humphrey, W. S.: A Discipline for Software Engineering (SEI Series in Software Engineering). Carnegie Mellon University, ISBN: 0-201-54610-8. Addison-Wesley Publishing Company (1995).

[10] Jedlitschka, A.; Nick, M.: Software Engineering Knowledge Repositories. In: Conradi, Reidar (Ed.) u.a.: Empirical Methods and Studies in Software Engineering: Experiences from ESERNET. Berlin: Springer-Verlag, 2003, 55-80: Ill., Lit. (Lecture Notes in Computer Science 2765).

[11] Kerth, N.L.: Project Retrospectives: A Handbook for Team Reviews. Dorset House Publishing, ISBN: 0-932633-44-7, New York (2001).

[12] Nielsen, J., Mack, R.L.: Usability Inspection Methods / John Wiley & Sons, Inc; (1994).

[13] Ocampo, A.; Boggio, D.; Münch, J.; Palladino, G.: Toward a Reference Process for Developing Wireless Internet Services. In: IEEE Transactions on Software Engineering 29 (2003), 12, 1122-1134: Ill., Lit.

[14] Solingen, Rini van; Berghout, Egon: The Goal/ Question/ Metric Method. A Practical Guide for Quality Improvement of Software Development. London : McGraw-Hill, 1999