MODELING QUALITY OF ADAPTIVE MOBILE USER INTERFACES

Rodolfo Canelon[1], Francisca Losavio[2], Alfredo Matteo[3]

[1] Dpto. Sistemas, Decanato de Ciencias y Tecnología,
Universidad «Lisandro Alvarado», Barquisimeto, Venezuela.
[2] Laboratorio LaTecS, Centro ISYS, Universidad Central de Venezuela.
[3] Laboratorio TOOLS, Centro ISYS, Universidad Central de Venezuela.

Recibido: agosto de 2006 Recibido en forma final revisado: febrero de 2007

ABSTRACT

The use of mobile devices is now worldwide. Mobile and pervasive computing deals with the development of software applications for these devices in wireless environments. The development cost is high since there are no market standards for these devices and the application is often designed ad hoc. The user interface in particular is constrained by limited resources such as memory, display size and resolution, and it varies for each type of commercial display. The goal of this work is to characterize the adaptive user interface component of a mobile application using a quality model for evaluating the quality provided by its architectural solutions. Quality properties are given for each architectural solution; as a consequence, the reliability of the application is improved, and the development costs are lowered. The identification of functional and non-functional requirements for the adaptive user interface plays an important role in our approach, since they are directly related with the quality properties which drive the architectural choices. Our approach is applied to a case study in the e-banking domain.

Keywords: mobile computing, adaptive user interfaces, mobile agents, mediator, quality model, software architecture.

CARACTERIZACIÓN DE LA ADAPTABILIDAD DE LA COMPONENTE DE INTERFAZ DE USUARIO PARA UNA APLICACIÓN MÓVIL USANDO UN MODELO DE CALIDAD

RESUMEN

El empleo de dispositivos móviles esta siendo extendido por todo el mundo. La computación móvil estudia el desarrollo de aplicaciones de software para estos dispositivos en ambientes inalámbricos. Los costos de desarrollo son altos y no existen estándares de mercado para estos dispositivos y frecuentemente las aplicaciones son diseñadas ad hoc. La interfaz de usuario en particular, esta restringida por la limitación de recursos como memoria, el tamaño y resolución de la pantalla, y esto varía para cada tipo de dispositivo comercial. El objetivo de este trabajo es caracterizar la adaptabilidad de la componente de interfaz de usuario para una aplicación móvil usando un modelo de calidad que permita evaluar la calidad de las soluciones arquitectónicas propuestas. Se incorporan las propiedades de calidad para cada solución arquitectural de manera de garantizar la confiabilidad de la aplicación y la disminución en los costos de desarrollo. La identificación de los requisitos funcionales y no funcionales para la interfaz de usuario adaptable juega un rol importante en nuestro estudio, siendo directamente relacionadas con las propiedades de calidad las cuales guían la escogencia arquitectónica. Nuestra solución arquitectural es aplicada a un caso estudio, en el dominio de banca móvil.

Palabras clave: computación móvil, interfaces de usuarios adaptables, agentes móviles, mediadores, modelos de calidad, arquitectura de software.

INTRODUCTION

Mobile computing differs from the standard «desktop» computing since it imposes several challenges for the development of low cost, efficient and reliable software applications, which must run on heterogeneous mobile devices, such as Personal Digital Assistant (PDA) and mobile phones, with limited resources and transient connections. Handheld mobile devices are used in complex scenarios such as visual exploration, environmental monitoring, freeway-traffic management, fire fighting and natural disaster damage assessment, which require that a number of new challenges that are affecting the whole software engineering lifecycle be addressed. This new set of challenges is characterized as Prism (programming in the small and many), which refers to software development for highly distributed, dynamic, mobile, heterogeneous computations on large numbers of small resource constrained platforms (Medvidovic, 2003). In general, these «mobile applications» are built by recoding existing desktop versions. In particular, user interfaces must deal with miniaturization issues (display size and resolution) and device heterogeneity, increasing the development costs to assure their portability. The absence of traditional input/output devices increases the difficulty of the design, limiting the amount of information that can be handled. Application mobility (Eisenstein, 2000], is defined as the ability of an application to adapt itself to different mobile devices. The context of use (Eisenstein, 2000), is the environment associated with the mobile device. It is defined by a set of parameters (platform, network, QoS, etc.) for each environment. It affects the user interface usability. An adaptable or adaptive application reconfigures itself dynamically according to the context of use. In particular, the problem of adaptive user interface design has been addressed by several frameworks, an innovative method for component-based software engineering, backed up by a formalism for modeling components and by a generic component architecture, all tailored to the needs of embedded systems. Two other fields of application development are integrated in the component oriented-approach: the building of user interfaces for embedded systems and the debugging of embedded systems. In the MUSA (Menkhaus, 2005) framework the user interface is modeled as an event graph using the EG-XML descriptive language to describe dialogs and events (Musa, 2002). A study of (Sousa, 2002) indicated that a promising approach is to apply software architecture principles to the development of software systems within the Prism context.

The main goal of this work is to characterize architectural solutions for the adaptive user interface problem. The problem is described by its functional and non-functional requirements. The quality properties related with the requirements are specified by a standard quality model (Iso/Iec, 2001) and they are used to justify the architectural choices. Only the logic view of the architecture, in the sense of Krutchen, is considered (Krutchen, 1999). We consider this view because it focuses on the main functionality of the system, to provide a first broad structure of the system. This is also a view that is very much used in practice to establish a common understanding among the stakeholders. An implementation of one of the solutions, the mobile agent-based architecture is also presented, where Java and XML are the constraints for the technological platform.

The structure of this paper, besides this introduction and the conclusion is the following: section 2 characterizes the modeling elements of the user interface component in a mobile context and the architecture used. Section 3 discusses in details the Mediator architectural solution for adaptive interfaces. The UML model of this architecture is presented with some implementation considerations and examples. Finally, section 4 presents the characterization of the problem and its architectural solution by functional and non functional requirements and their quality properties. These are expressed by a quality model and are used to justify the architectural choices.

CHARACTERIZATION OF THE USER INTERFACE COMPONENT FOR MOBILE DEVICES

Contexts of use

The graphical user interface (GUI) used in desktop computing provide interaction based on the WYSIWYG (What You See Is What You Get) and WIMP (Eisenstein, 2001) (Windows, Icons, Menus and Pointing devices) principles. However these issues do not always apply to the user interface for mobile devices, faced with limited resources and different input/output systems. For example, most mobile devices do not have pointing devices and menus are used instead. Moreover, resource limitation imposes reduced functionality for the application and the user interface, which is not necessarily graphic. In consequence, the term user interface (UI) will be used for the mobile context, instead of GUI. Mobile devices are very different. In this work we will refer mainly to PDAs and mobile phones. The main standard functionalities of PDA are agenda, office document, music and video playback. Mobile phones are more oriented towards voice communication and text messages. However, nowadays most of these functionalities are common to both devices. UI must be adaptive to these devices, where the human hand determines the size of the display. There is also a limit on the optimal size of a text that can be displayed in order to be captured by the human eye. A multiple window system is impractical and applications run always on full display mode. Hence, new input devices are adopted, such as specific function buttons, touch pads or touch displays. UI is in general designed ad hoc for each specific device. Heterogeneity increases the development costs of applications, in particular of the interface component.

Main modeling elements

The classical user interface development models are (Coutaz, 1991):

- The platform model**,** describes the computer system (platform) where the interface is running.

- The presentation model, describes the interface structure (windows system and graphical components).

- The tasks model presents the tasks that the user must perform to execute the software functionality. It is a hierarchical structure made up of goals and pre-post conditions on the tasks.

User interface modeling languages must be declarative and independent from the implementation. The most used modeling platform independent languages for mobile devices are MIMIC, UIML, IML, and XIML model, the elementary interface unit is an interaction object (IO), which is an element that allows the user of an application to visualize or manipulate information or execute an interactive task. An abstract interaction object (AIO) represents a platform independent interaction object. A concrete interaction object (CIO) is an AIO implementation. An AIO is associated with several CIOs, inheriting behavior from the AIO. The presentation model is a tree-like structure of interaction objects. When the model is interpreted, the appropriate CIO is built automatically, according to the platform (Eisenstein, 2000). Each context of use imposes constraints on the input/output capacity (Luyten, 2001 and Menkhaus, 2002). The display resolution expressed in pixels must not be confounded with the display surface. Two displays can have different surfaces and the same resolution. An optimal distribution of the interface presentation can work for a device and not for another. The interface adaptability depends on the number of the context of use. Three parameters determine the space required for display: individual IO, location of the IO within the window, distribution of the IO. A Logical Window (LW) is a constituted by composition of AIO. LWs are limited by the display constraints of the device (Eisenstein, 2001; Menkhaus, 2002; Eisenstein, 2000). A Presentation Unit (PU) is the complete presentation environment required for the interaction. Each PU can be decomposed into LWs; it is composed by at least a main window from which other windows may be accessed. Notice that this configuration is supported by the PAC (Presentation, Abstraction, and Control) pattern, which is a mediator-based architectural pattern (Coutaz, 1991 and Losavio, 1994).

Architectures for user interface in mobile context

Several architectures have been proposed to handle the constraints imposed by the mobile devices. Two of them are presented in what follows.

Architecture 1: determine the convenient size of IOs

Two solutions are proposed:

- The size of the IO is reduced to reach a level so that its usability is preserved.
- The IO is substituted by another IO with the same functionality but requiring less space.

A decision tree: is used to select the size and properties of an IO as a function of the device context (Eisenstein, 2000). The decision tree does not solve the problem of the efficient location and distribution of IOs within one or multiple windows. The following architecture gives an automatic solution to this problem.

Architecture 2: determine automatic location and distribution of AIOs within one or multiple windows

A mediator-based pattern is used to solve this problem (Eisenstein, 2000 and 2001; Menkhaus, 2002; Luyten, 2001) (figure 1). Mediator is an abstract class defining a communication interface between Colleague classes and controlling the data transmission. It enforces the interactions among colleagues and it contains the basic methods to manage the flow of information between independent colleagues (Mediator, 2002). However, it does not solve the problem of the automatic adaptation to the different contexts of use.

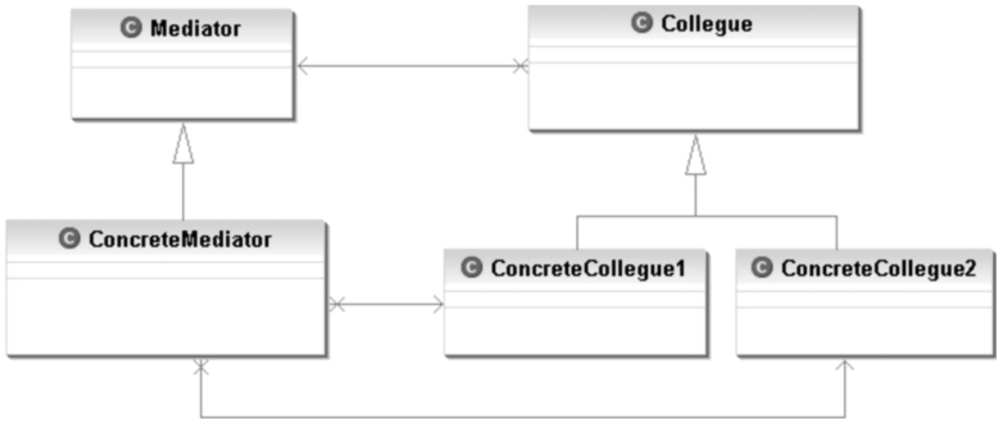


Figure 1. General structure of *Mediator* [Mediator02].

Architecture 3: Adaptability with mobile agents

Architectures based on software agents have been widely used for distributed applications (Genesereth, 1994; Gomez, 2001). Agents are software entities with the following characteristics: facility to capture and apply knowledge and autonomous decision processing capacity for problem solving (intelligence). Static agents can be executed only in the machine where they are initiated. Mobile agents, instead, can be transported from one machine to another, allowing direct interaction with the agents of the system where the required object is located. The agent can:

- gather information from the architecture and the environment;

- be triggered by the architecture and the environment in the form of an exception generated by the application;

- make proper decisions using rule-based intelligent mechanism;

- communicate with others agent components controlling other relevant aspects of the architecture;

- notify other agents that one or more modification have occurred in the configuration that it controls;

- Ensure, in collaboration together with others agents, some quality aspects of a system by controlling systematically components communication properties such as security, reliability, etc; perform some action on the architecture to manage the changes required by some modification.

Agent-based (Structure and Participants)

An agent emits a certain message sentmessage (reach, persistence), using two parameters (Schelfthout, 2003):

- Reach is the intensity of the message.

- Persistence is the time the message «hangs in the air».

The agents that should get the message are not statically reachable. They move in the environment, and they only «hear» the message when they are present at the right time, and close enough to hear it.

Basically, the environment consists of a «Communication Broker» and a «Location Broker». According to the message's parameters *obtainTargets: void*, the Location Broker selects the agent that hears the message, while the Communication Broker actually delivers it with message: void; resultSet : void. Due to the collaboration between the Location Broker and the Communication Broker, the right set of agents will receive the message in the right period of time (Schelfthout, 2003) (figure 2).

A mobile agent-based architecture (Mediator) based on the mediator pattern and on the mobile agent pattern is proposed in figure 3 to achieve adaptability. This architecture uses the planning processes established by architectures 1 and 2, with an optimal adaptation to the contexts of use. It provides solution to the following adaptability problems:

- Determine the convenient size of IOs.

- Determine the automatic location and distribution of AIOs within one or multiple windows.

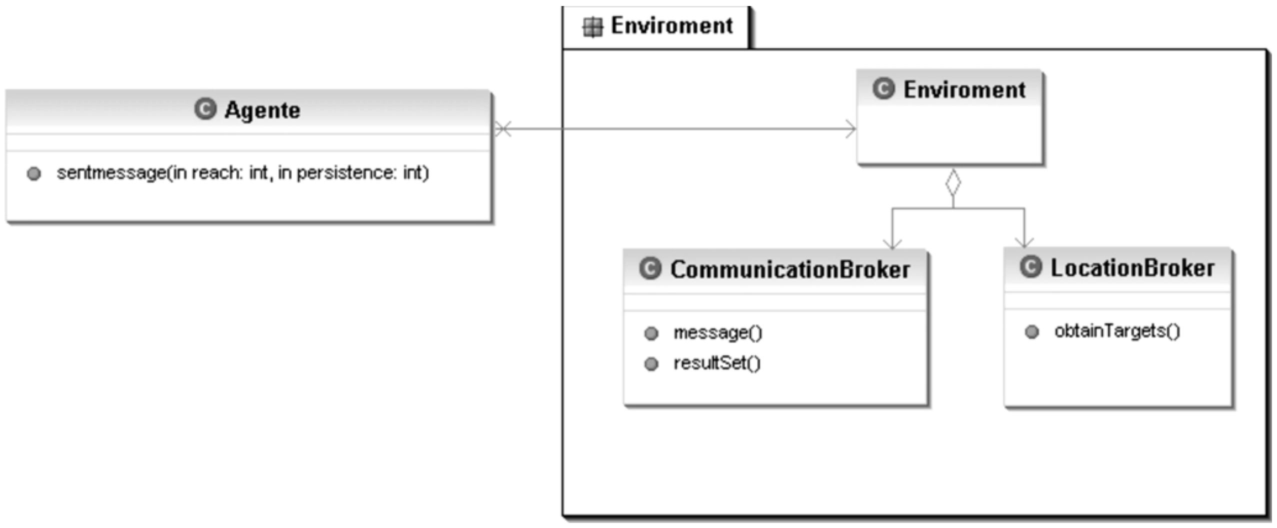


Figure 2. Structure of the communicator pattern between the mobile agent and the environment.

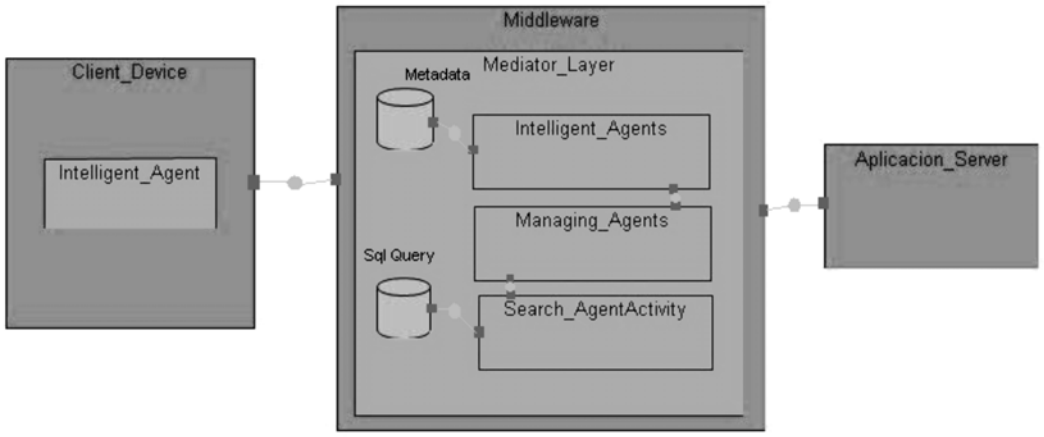


Figure 3. The *MEDIATOR* architecture.

- Determine the adaptation to the different contexts of use at run-time (dynamic adaptability to the mobile environment).

The Mediator architecture is a mobile agent with the following capabilities:

- Exploit the information on the resource limits of the device.

- Compute the maximal usable display resolution for each presentation structure.

- Determine the optimal context of use, as a function of its parameters.

- It is part of the middleware for the mobile network.

The basic structure of the Mediator mobile agent is the following:

Mediator is an instance of ConcreteMediator (Figure 1) and is part of a Multi Agent System (MAS) hub of 3 layers:

- Intelligent Agents (User Interface Agents)

- Managing Agent (Coordination Monitoring)

- Search Agents/Activity

Intelligent Agent/User Interface Layer

The Intelligent Agent (also called User Interface Agent) is a static agent responsible for displaying the PU on the device. The clients use the Intelligent Agents which are constituted by AIOs, CIOs, or concrete LWs. They are defined as the set of provided actions and required events. For each intelligent agent we attach an Event/Condition/Action-rules mechanism in order to react with the other layer (Managing agents) and the environment and they are responsible of performing the user interface.

Managing Agents (Coordination Monitoring) Layer

The Managing Agent is a static agent responsible for the generation and transmission of the queries through the Search Agents.

– **Search Agents Layer:**

It is responsible for the environment information, the client information, sent from the Managing Agent to the destination device. It communicates with other Search Agents to exchange intermediate data. It sends the result obtained to the Managing Agent.

– **Environment:**

- The Metadata: The metadata dictionary in the environment stores all data description and associated information from several information sources that are used by Intelligent Agents and Managing Agents (Ngamnij, 2003). It concerns the data model for each device, the domain family and the configuration of each information source that are required by the Search Agents.

- Constraint functions: They are declarative approaches with explanation capabilities, heuristic control, generation and pruning of alternatives. The CIOs run on the client device and are associated to events activating the triggers on the environment database.

- Allocating: use thread allocation to PU, AIOs, CIOs agents and the communication protocols.

This architecture is dynamically adaptive to the changes of the device display and to the platform changes. The end-user has only to precise the available display size in terms of resolution measures. The automatic generation of the presentation structure is performed on a set of alternative presentation structures and on a set of constraints on the AIOs groups. The alternative presentation models will be adapted to a specific mobile platform by decomposing recursively the tree structure with the composite and simple AIOs. The relation between the presentation model and the tasks model is described in (Eisenstein, 2001; Menkhaus, 2002; Eisenstein, 2000). It is assumed that the same set of tasks is performed for each device. The profile of each device contains the elements of the execution platform, for example the display size, the colors supported etc., are mapped on the high level elements of the task model. The tasks are mapped on the elements of the presentation model to be executed.

The Mediator mobile agent, based on the architectures described previously, determines an optimal user interface (best fitted to the runtime platform constraints). In Section 4, this assumption will be justified (Gamma, 1995; Mediator, 2002) on the basis of the quality properties accomplished. This pattern promotes loose coupling by keeping the objects from directly referring or even contact each other. If all signals or messages are first sent through a mediator, to change the behavior of the entire system would basically consist of changing the methods within the mediator class. The mediator is a class whose objects at runtime are responsible for controlling and coordinating the interactions of a group of other objects (Raj, 2002). The level of reuse for highly communicative objects will increase with the localization and changing of common tasks into Mediator subclasses rather than Colleague classes (Raj, 2002).

Benefits of Mediator

1. Provide optimal interfaces to the different mobile platforms at runtime.

2. Promote a high level of reusability. Code reuse for the migration of mobile applications to mobile platforms device, specifically when domain applications are huge.

3. Exploit the information on the resource limits of the device, due to the centralized flow of information following established constraints on the mobile platform in use.

4. Reduce the application load introducing mediation, translation and interaction services (Gamma, 1959). the mediation between the platforms and the application is agent function and translation to the mobile platforms the optimal interfaces.

5. Improve the system structure. Splitting up the system in this way creates a better understanding of the objects in the system: how they interact and how they are structured, since all objects are bundled into one class (Raj, 2002). Different optimal views of presentations for each platform are obtained.

Mediator vs. Related patterns

Similar to the observer pattern, where related objects need to communicate; in the observer a change in one object (the subject) will sometimes require other objects (observers) to be updated and the update is explicitly coded in the subject; this relation requires knowledge about how the observers should be updated. In Mediator, a change in one object generates changes only in the inferior subclasses, minimizing the knowledge shared between the colleague objects. The observer is applicable with good results when the updates are not independent. Encapsulating these aspects in separate objects will increase the chance of reusing them independently. Mediator or does not need to consider PU dependencies among the colleague objects. Mediator exhibits properties of both bridges and wrappers. The major distinction from bridges and wrappers, however is that Mediator incorporates a planning function that in effect results in runtime determination of the translation (recall that bridges establish this translation when the bridge is constructed) (Bass, 1998). The translation process needs to exploit the information on the device resource limits and compute the maximal usable display resolution for the presentation structure of the platform in use. The planning function (located into Intelligent Agent) with its parameters makes the choice of optimal user interfaces.

IMPLEMENTATION CONSIDERATIONS

ConcreteMediator is a subclass of Mediator. Mediator is an instance of ConcreteMediator as shown in figure 4, where an example with two concrete presentations UPConcrete1 and UPConcrete2 is presented. The ConcreteColleague number of classes will depend on the number of presentation models for each specific platform.

Mediator basic behavior

Communication between Mediator and the mobile device: XML and Java are well suited interface modeling languages because they are declarative and independent from the platforms (Eisenstein, 2000) as shown in figure 5: XML will be used as follows in the communication process:

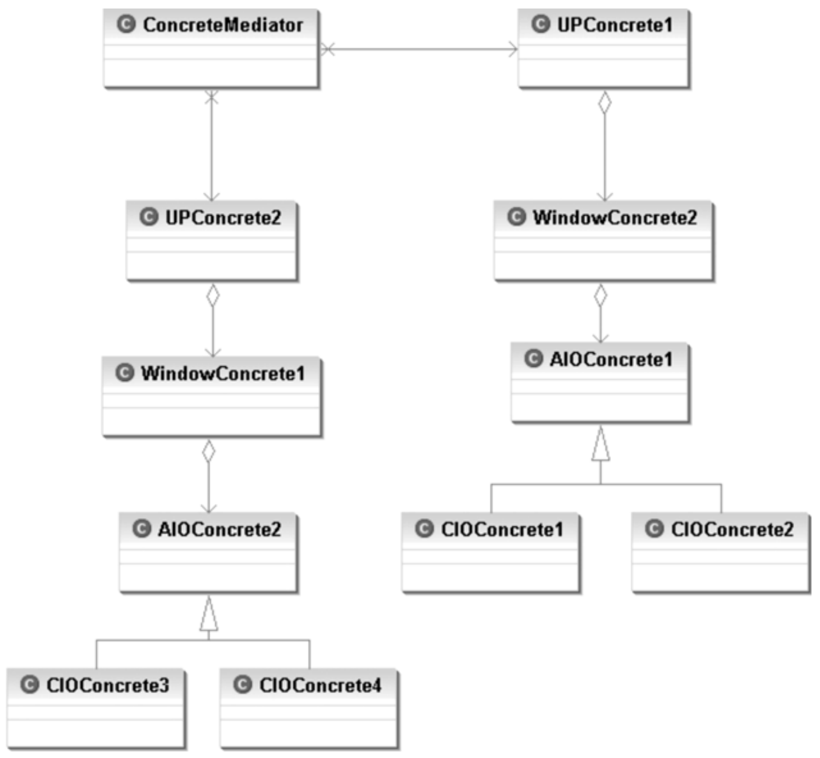
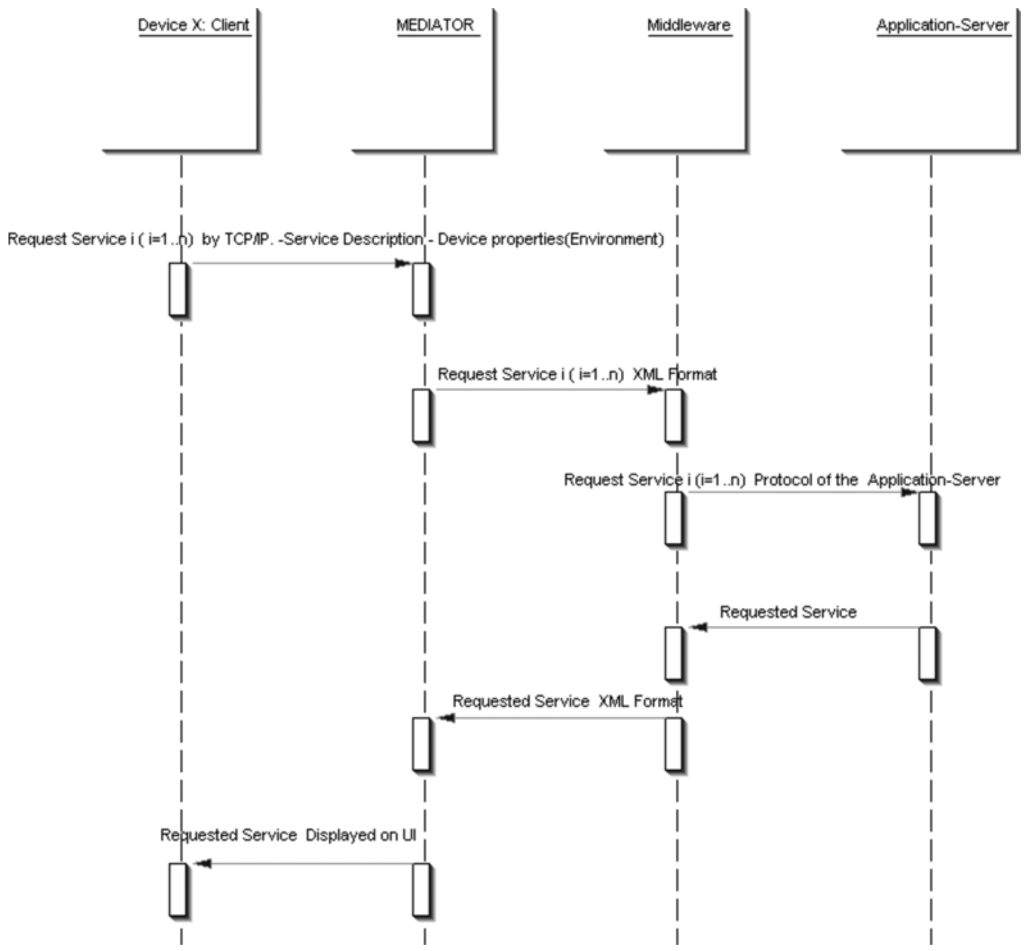


Figure 4. *Mediator* pattern showing two concrete presentations.



Device X: Client
MEDIATOR
Middleware
Application-Server
Request Service i (i=1..n) by TCP/IP. -Service Description - Device properties(Environment)
Request Service i (i=1..n) XML Format
Request Service i (i=1..n) Protocol of the Application-Server
Requested Service
Requested Service XML Format
Requested Service Displayed on UI

Figure 5. UML Sequence Diagram of Communication between Mediator and device.

JUSTIFYING ARCHITECTURAL SOLUTIONS FOR THE ADAPTIVE USER INTERFACE

In the previous section we have discussed the basic elements involved in the UI component and some of the architectural solutions. The architecture based on the mediator pattern has been proposed in the literature as an acceptable solution adopted by practitioners on a previous experience (Vanderdonckt, 2001). In what follows, this choice will be justified studying the quality properties enhanced by this solution. In our approach, the architecture is considered as a problem-solution pair (Lévy, 2004), hence we have to identify the problem part by defining it in terms of its functional and non functional requirements, define the quality model (Iso/Iec, 2001) related with the problem domain and study the choices that have been made to select the architectural configuration as a solution. These choices are driven by the quality properties related with the problem's requirements. Our approach is now also considered by the Aspect Oriented Software Design (AOSD) paradigm where the crosscutting concerns which are in general quality properties derived from software requirements (Brito, 2003), must be taken into account very early in order to improve the overall system structure. The architecture is a software artifact where these issues must be taken into account.

Architectures are the baseline on which a software system is articulated. They are solutions to particular problems described in detail, without ambiguity, and organized so that they can be understood and reused (Gamma, 1995). The transition from one abstraction level to another is not straightforward. The Model Driven Architecture (MDA) approach (Mda, 2001) is an attempt to fill this gap. Methods have been developed for architectural design (Bosch, 2000; Clements, 2002) and this step is now included in general software development frameworks (Clements, 2002). On the other hand, all these methods consider that non functional requirements are crucial for architectural design, especially when the application must respond to critical issues and to a changing environment. However, their specification is very often only superficially considered.

In what follows, the problem is expressed in terms of its functional and non functional requirements.

Problem: build context adaptive user interface

The considered problem consists in building user interfaces that reconfigure dynamically to the context of use of a mobile device in a wireless network environment.

From the previous section, the main concerns faced by user interface designers in wireless application development (MOTI, 2004; IBM, 1999) are:

- Limited bandwidth: A wireless device typically has much less bandwidth available for transmitting and receiving data than a wired device.
Intermittent connection: The connection to a wireless device is typically unreliable. A persistent point-to-point connection is difficult, if not impossible.

- Limited battery life: A wireless device is typically (also) a mobile device. Since mobility dictates compactness in size, and since there is no wired power connection, batteries are the only means of power supply. Even the longest lasting batteries offer a very limited amount of power.

- Limited memory on client device: Once again, because of the mobile nature of wireless devices and their requirements to remain a small size, room for memory is limited. Memory is also limited by the available power source (batteries) on the device.

- Limited CPU: Because of the size of the devices and the battery life, processing information on the device is very expensive. Very few operations should be performed on the device and they should be only performed where there is strong justification for them.

- Limited user interface devices: A keyboard and/or a mouse are normally not available for a wireless or a mobile device. Also, the display is almost always very small. This makes viewing and data entry more difficult.

These concerns indicate that minimization of resource consumption is a mandatory concern that can be measured using the efficiency quality property.

In consequence, from the analysis of the above concerns we are able to state the following non functional requirements for the software system.

Non functional requirements

- Resource use minimization (battery life, data storage, display size, etc.).

- Data must be transmitted completely, consistently and correctly, with transient connections (in spite of network failures).

- Limited data transmission time. Convenient range of response time.

- Handle changes in communications among dynamic or static components.

- Notice that the above non functional requirements are constrained related to the problem domain environment, where the software system will be executing. They indirectly affect the system architecture.

- Centralized solution, due to the presence of the middleware.
- Handle reconfiguration of interfaces.

These two non functional requirements are directly related with the system architecture.

Business rules

They identify requirements such as company policy or rules that must be considered by the software system.

- Established network communication protocols and bandwidth.

Requirements may be seen in general as main «goals» that the system must accomplish. In fact, due to a changing environment, the context may change in such a way that the putting into operation of the goal is no longer valid. In this sense requirements may be seen as trade-offs between the goals and the actual reality. For example, a goal for a service may be to provide for a highly interactive user experience. If the context is favorable (high bandwidth, large color display, Java Virtual Machine available, etc.), the goal is translated into the service «use a Java applet to represent the shopping basket» (Finkestein, 2004).

Functional requirements

- Users install/uninstall applications (services) on the mobile device. A service may be installed or uninstalled over a device by a user because he explicitly requests it. The data displayed by the required service must be attractive to the user. Services must be related to the user rather than to a specific device, or specific location. (User Oriented Services)

- Code and data are downloaded/uploaded from/to the server and stored locally on the mobile device. Data must be completely and correctly transmitted**.** Messages and data from the server are downloaded and stored locally. In a pervasive computing environment, there are various ways of accessing different types of data according to different users needs. A combination of code and data mobility should enable to construct a flexible data model. (Data Management).

When the service is running, the adaptive user interface is automatically and contextually reconfigured for this environment. Failure can be caused by the user resource scarcity (battery, memory, etc.) or by disconnection (connection fails). Client-dependent adaptability such as a changing environment requires the dynamicity of the user interface component in applications, i.e. to dynamically change according to the user's device configuration. Automatic contextual reconfiguration is a process of adding new components, removing existing components, or altering the connections between components due to context changes.

Quality model and architectural solutions

According to the domain of context adaptive user interface in a mobile environment (Lévy, 2004) and the requirements stated above, the following internal and external quality properties, specified according to (Iso/Iec, 2001), characterize this domain and constitute its quality model: Reliability (availability, consistency), Functionality (security for data access control), Efficiency (performance with respect to resources utilization, response time), Maintainability (changeability with respect to the execution environment, scalability to adopt the device resources), Portability (adaptability, replaceability), Usability (attractiveness, operability).

Table 1 shows the direct relation between the adaptive user interface quality properties, expressed in the quality model and the non functional requirements, that is to say the quality that each requirement should accomplish. Notice that the metrics for each quality property are given at architectural design level, i.e. at a high abstraction level. They must be provided very early in the development process and are meant as broad guidelines to document decisions that can be shared by all stakeholders for a common understanding on the software project.

Table 2 shows possible architectural solutions or design patterns used to solve each functionality responding to a precise quality property.

Table 1. Non functional requirements, quality properties and proposed architectural solutions for each requirement

NON FUNCTIONAL REQUIREMENTS	**QUALITY PROPERTY**	**ARCHITECTURAL SOLUTION**
Resource use minimization (battery life, data storage, display size, etc.).	Efficiency (performance with respect to resource utilization); quantitative property Attribute: resource consumption for each device Metric: percentage	Network Communication Protocol, bandwidth , direct measure during execution
Data must be transmitted completely, consistently and correctly, with transient connections	Reliability (consistency: provide a mechanism, e.g. the management of replicated data update on each device); qualitative property Attribute: presence of a mechanism Metric: Boolean [0..1]	Mechanism: Middleware including *Mediator*
Limited data transmission time. Convenient range of response time.	Efficiency (performance with respect to response time due to battery scarcity and transient connections); quantitative property Attribute: latency Metric: percentage	Network Communication Protocol, bandwidth, direct measure during execution
Handle changes in communications among dynamic or static components and in components	Maintainability (changeability: extensibility of the execution environment); quantitative property Attribute: size Metric: Measure of complexity, e.g. number of dynamic components in a period of time Portability (replaceability) quantitative property Attribute: size Metric: Measure of complexity	Reflection pattern (Schmidt, 2000) to observe the state of a component, to allow dynamic changes in structure and behavior, direct measure, during execution
Centralized solution with respect to data, due to presence of the middleware	Reliability (availability); qualitative property Attribute: presence of a mechanism Metric: Boolean [0..1]	Mechanism: Middleware including *Mediator*
Dynamic reconfiguration of interfaces	Maintainability (changeability); quantitative property Attribute: size Metric: Measure of complexity Portability (adaptability) qualitative property Attribute: presence of a mechanism Metric: Boolean [0..1] Reliability (consistency); qualitative property Attribute: presence of a mechanism Metric: Boolean [0..1]	Reflection pattern (Schmidt, 2000) to observe the state of a component, to allow dynamic changes in structure and behavior, direct measure, during execution Mechanism Middleware including *MEDIATOR*

CASE STUDY

Device-based mobile applications are developed using technologies such as Java 2 Platform Micro Edition (J2ME) of Sun Microsystems, which support a range of PDA (Personal Digital Assistant), smart phones and other mobile devices. In terms of advantages, device-based mobile applications provide sophisticated interaction styles beyond the simple navigation model of web based applications. They also offer a more immediate experience since they are not so heavily bound by request/response cycles inherent in web-based design. Furthermore, such applications can be used offline, with information synchronized with the server upon connection and disconnection. Disadvantages include the need for more sophisticated devices, more costly development and deployment (since existing web-based applications cannot be readily ported to device based ones), additional user configuration, and problems with client-side incompatibilities. Note that device-based mobile applications can be used in conjunction with web-based mobile applications. The case study selected to illustrate our approach is taken from a real application in the e-banking domain. We only present here the case of an account summary services problem. In particular, let us consider that a bank client has a few accounts and want an account summary, according to his location and preferences. The user wants interaction with the services from different devices viewing the account summary in a quickly, easy and secure way. The use case model for the AccountSummary functionality is shown in figure 6.

Table 2. Quality properties and proposed architectural solution for each functionality

FUNCTIONALITY	QUALITY PROPERTY (INTERNAL/EXTERNAL)	ARCHITECTURAL SOLUTION
Data Management, on a flexible data model	Reliability (availability, consistency) Attribute: presence of a mechanism Metric: Boolean [0..1] Efficiency (performance with respect to limited resource space, performance with respect to limited response time) Attribute: resource consumption for each device Metric: percentage Maintainability (changeability), quantitative property Attribute: size Metric: Measure of complexity	LocalProxy (Roth, 2001) A local instance provides an interface to a local or remote service. Example : Locally running web proxy PushObject (Roth, 2001) PushObject A device sends a specific object without request. Examples : SMS, OBEX RequestObject (Roth, 2001) A device requests a specific object (e.g. a web page) from another device. Example : WAP CannedCode (Roth, 2001) CannedCode A device sends code, which is executed on another device. The code must not be executable on the sender's device. Example: WMLscript, web filters
User-Oriented Services	Maintainability (scalability) quantitative property Attribute: size Metric: Measure of complexity Usability (attractiveness, operability) Attribute: presence of a mechanism Metric: Boolean [0..1] Reliability (fault-tolerance) Attribute: presence of a mechanism Metric: Boolean [0..1]	Reflection (Schmidt, 2000)] WrapperFaçade (Schmidt, 2000) Encapsulate functions and data of non object-oriented APIs into concise, portable, maintainable and robust interface classes. ComponentConfigurator[(Schmidt, 2000) Allow the component reconfiguration in run time, avoiding modifying, recompiling or relinking the application statically. Interceptor (Schmidt, 2000) Allow to transparently add services when certain events occur. Variants: Chain-of-responsibility, Publisher/subscriber and Subject/observer. Extension Interface (Schmidt, 2000) Allow to export multiple interfaces by a component to prevent the increase of the number of interfaces and the breaking of the client code. The component functionality is extended and modified. *Mediator*

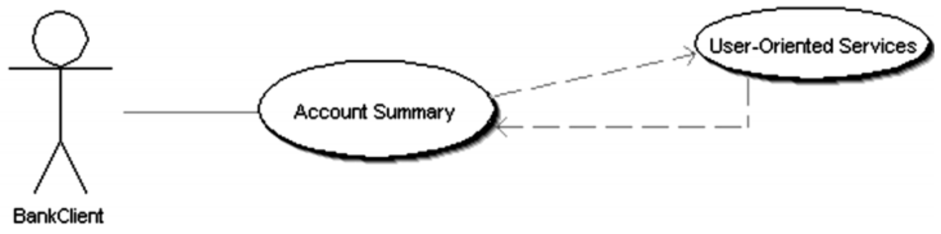


Figure 6. Use case Model for Account Summary.

Functional requirements for AccountSummary

1. Users install/uninstall AccountSummary Services on the mobile device. A service may be installed or uninstalled over a device by a user because he explicitly requests it. (User-Oriented Services)

2. The data must be attractively displayed by the required AccountSummary Service (User-Oriented Services)

3. Code and data are downloaded/uploaded from/to the server and stored locally on the mobile device. (Data Management)

4. Data must be completely and correctly transmitted. (Data Management)

Non functional requirements for AccountSummary

1. Resource use minimization (battery life, data storage, display size, etc.).

2. Data must be transmitted completely, consistently and correctly, with transient connections (in spite of network failures).

3. Limited data transmission time. Convenient range of response time.

4. Handle changes in communications among dynamic or static components.

5. Centralized solution, due to the presence of the middleware.

6. Dynamic reconfiguration of interfaces.

7. The AccountSummary Service must be secure from the user requirement in the problem definition.

The quality properties related to the global configuration shown in figure 7 are due to the user requirements stated above: Efficiency, Portability Security and Usability. To provide an architectural solution to the problem of the AccountSummary, several components are introduced to respond to specific quality properties: the component User-Oriented Services must respond to Usability (attractiveness, operability), Maintainability (scalability), Reliability (fault-tolerance) and Security; Data management must respond to Reliability (consistency) and Efficiency (response time).

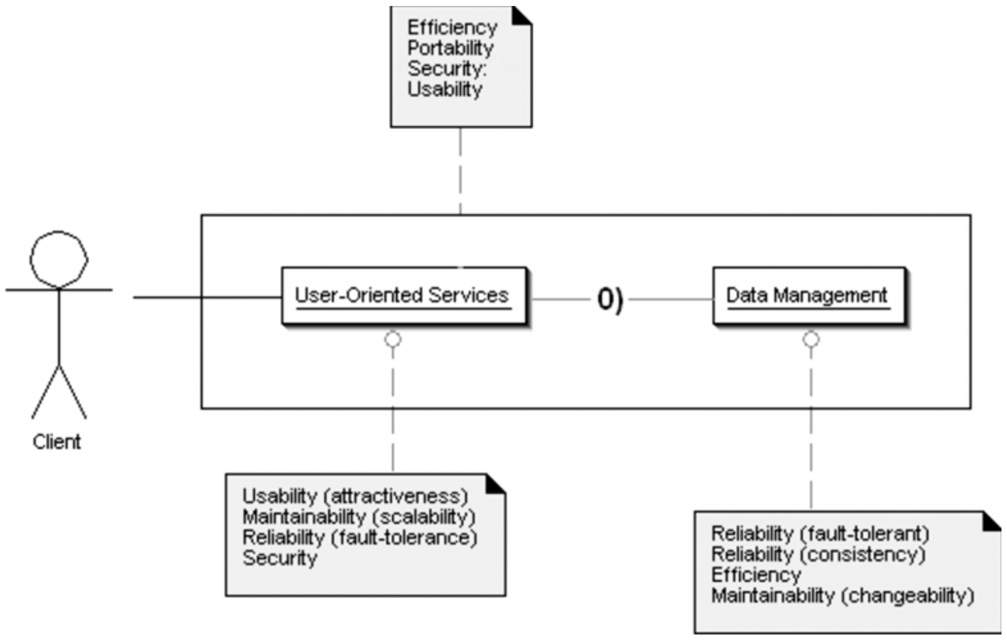

Efficiency
Portability
Security:
Usability
User-Oriented Services
0)
Data Management
Client
Usability (attractiveness)
Maintainability (scalability)
Reliability (fault-tolerance)
Security
Reliability (fault-tolerant)
Reliability (consistency)
Efficiency
Maintainability (changeability)

Figure 7. Summary Account in Mobile Device.

When the User-Oriented Services is running, the adaptive user interface and the Security Server have the environment parameters (figure 8). The GUI/Client-Side component is introduced in response to Usability (attractiveness) and the Security Server component is introduced in response to Security and Reliability (fault-tolerance). The Middleware component is then introduced to guarantee Efficiency and Reliability (availability) in the communication, due to the Centralized solution model. Moreover it fulfills the Maintainability (Scalability, Changeability) property. The Data Bank Server component is a solution introduced for data management for reliability (consistency); however, it involves also security, which is a crosscutting concern since it is also required by the GUI-Client Side, as shown in figure 8.

The mediator mobile agent is dynamically adaptive to the changes of the device display and the platform. According to Table 1, Dynamic Reconfiguration of Interfaces must accomplish Portability (Adaptability, Replaceability), Maintainability (Scalability, Changeability). Since the Data Bank Server requires Security, a new Security component is introduced, as shown in figure 9.

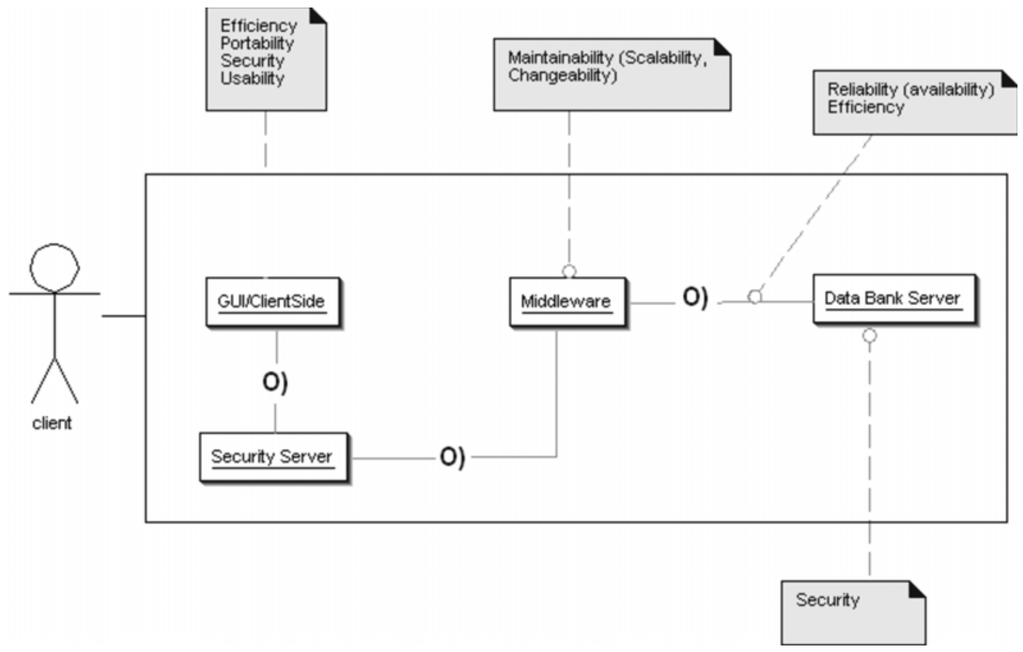


Figure 8. Balance account view.

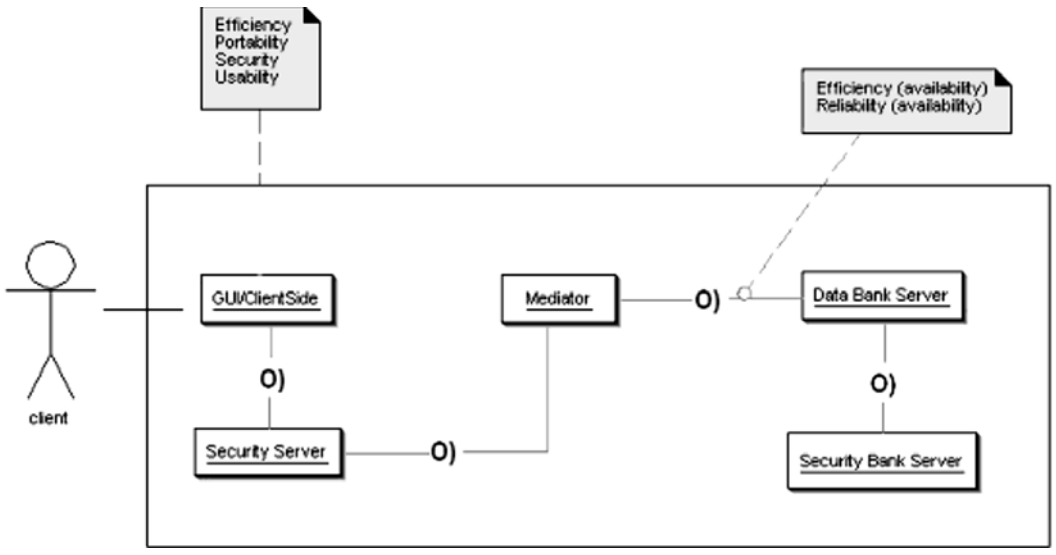


Figure 9. Balance account view with GUI.

To obtain the final architectural configuration for the AccountSummary Services problem, we have to justify the quality properties specified in the global configuration tag and the properties on the communication between Mediator and Bank Data Server (figure 9). On the one hand, with regard to the efficiency and reliability of the communication, the respective protocol will be responsible. On the other hand, in the end, architectural portability is guaranteed by the Mediator middleware, efficiency by the communication protocol, security by the client-side Security Server and the Bank Security Server and finally usability by the GUI client-side component. Notice that, according to an AOSD approach (Brito, 2003) we can think of having only one Security component concentrating the security concerns of the Client Side and Bank Data Server, to optimize the final architectural configuration.

CONCLUSION

An architectural solution for adaptive user interfaces based on the mediator design pattern has been proposed, characterized and justified on the basis of the quality properties issued from the system requirements. These properties are specified by a standard quality model (Iso/Iec, 2001), where high level metrics for architectural design have been defined; they can be used to justify the architectural choices on more solid bases than the usual experience bases. The components and connections of the different architectural configurations incrementally obtained, have been selected and justified on these bases. This documentation favors a common understanding for the stakeholders involved in the software project. A future research trend is to generalize our approach to context aware applications including explicitly AOSD issues.

REFERENCES

BASS L., (1998): Clements P., Kazman R. Software Architecture in Practice. pp. 337-344.

BOND G., (2002): «Research Methods Lecture». http://www.psynt.iupui.edu/gbond/gbond/I643%20Design%20241/index.htm.

BOSCH J., (2000): «Design and Use of Software Architecture», Addison Wesley, Harlow, England.

BRITO I, MOREIRA A., (2003): Towards a Composition Process for Aspect-Oriented Requirements, Early Aspects Workshop (AOSD Conference), Boston, USA.

CLEMENTS P., KAZMAN R., KLEIN M., (2002): «Evaluating Software Architecture Methods and Case Studies». SEI Series in Software Engineering. Addison-Wesley.

COUTAZ J., BASS L., (1991): «Developing Software for the User Interface» Addison Wesley Cummings Publishing Company.

EISENSTEIN J., VANDERDONCKT J., PUERTA A. «Adapting to Mobile Contexts with User-Interface Modeling». Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'00). http://www.ximl.org/documents/XIMLMobile.pdf

EISENSTEIN J., VANDERDONCKT J., PUERTA A., (2001): «Applying Model-Based Techniques to the Development of Uis for Mobile Computers». http://www.ximl.org/documents/XIMLMultiChannel.pdf. January

FINKESTEIN A., SAVIGNI A., (2004): «A Framework for Requirements Engineering for Context Aware Services». http://www.cin.ufpe.br/~straw01/epapers/paper14/paper14.html.

GAMMA E., HELM R., JOHNSON R., VLISSIDES J., (1995): «Design patterns Elements of Reusable Object Oriented Software». Addison Wesley, New York.

GENESERETH M., KETCHPEL R., (1994): S.P. Software agents. Communications of the ACM 37(7) (1994) pp. 48–53.

GÓMEZ R., (2001): Agentes Móviles y CORBA. http://www.fie.us.es/~ramon/tesis/CORBA/Seminario-MASIF/. Barcelona.

IBM SYSTEMS JOURNAL «Pervasive Computing», (1999),Volume 38, Number 4, .

KRUTCHEN KRUTCHEN P., (1999): The Rational Unified Process, Addison Wesley, Reading, Massachusetts.

ISO/IEC ISO/IEC 9126-1. Software Engineering - Product Quality. Part 1: Quality Model, 2001.

LÉVY N., LOSAVIO F., (2004): «Architectural Choices for Dependable Systems». WADS, 26th ICSE, May.

LOSAVIO F., MATTEO A., ORDAZ O., MEZA O., GONTIER W., (1994): «An implementation of the PAC Architecture using ObjectOriented Techniques», IFIP Transactions, Volume: 2 , No. 1 Elsevier Science B.H. (North Holland) pp. 149-155.

LUYTEN K., CONINX K., (2001): «An XML-based runtime user interface description languaje for mobile computing devices». http://xml.coverpages.org/LuytenDSVIS2001.pdf

MEDIATOR DESIGN PATTERN, (2002): http://sern.ucalgary.ca/courses/SENG/443/W02/assignments/Mediator

MEDVIDOVIC N., MIKIC-RAKIC M., MEHTA N., MALEK R., (2003): «Software Architectural support for handheld computing», IEEE Computer, Special Issue on Handheld Computing, vol. 36, no. 9, pages 66-73 (September 2003). Acceptance rate 5 of 87.

MENKHAUS G., PREE W., (2005): «User Interface Tailoring for Multi-Platform Service Access». IUI International conference on intelligent user interface. San Diego California, January.

MENKHAUS G., PREE W., (2002): «A Hibrid Approach to Adaptative User Interface Generation». Journal of Computing and Information Technology (CIT), 10(3).

MDA DOCUMENT number ormsc/2001-07-01, «Architecture Board ORMSC», July 9, 2001.

MOTI N., CHOW Y., KWAN V., WANG C., LAU F., (2004). «A Component-based Software Architecture for Pervasive Computing», Intelligent Virtual World: Technologies and Applications in Distributed Virtual Environments, World Scientific Publishing Co. USA. http://www.softwareresearch.net/site/publications/C048.pdf. April 1, 2002.

NGAMNIJ A., (2003): «A reference architecture for integrating heterogeneous information sources using XML and Agent Model». Advance virtual and intelligent Computing (AVIC) research center. University of Thailand.

ROTH J., (2001): «Patterns of mobile interaction». Mobile Conference Human Computer Interface.

SCHELFTHOUT K., CONINX T, HELLEBOOGH A, HOLVOET T, STEEGMANS E, WEYNS D., (2002): «Agent Implementation Patterns». Proceedings of the OOPSLA Workshop on Agent-Oriented Methodologies (Debenham, J. and Henderson-Sellers, B. and Jennings, N. and Odell, J., eds.), pp. 119-130.

SCHMIDT D, STAL M, ROHNERT H, BUSCHMANN F., (2000): «Pattern-oriented software architectural». Pattern for concurrent and networked objects.Willey.

SEESCOA., (2003): «A Concept-Based Approach to Software Design». Proceedings of the 7th International Conference on Software Engineering and Applications. Marina del Rey,

SOUSA J., GARLAN A., (2002): An architectural framework for user mobility in Ubiquitous computing Environments, Proc. 3rd. Working IEEE/IFIP Conf. Software Architecture, Kluwer Academic, pp.29-43.

VANDERDONCKT J., FLORINS M., OGER F., (2001): ModelBased Design of Mobile User Interfaces. http://www.cs.strath.ac.uk/~mdd/mobilehci01/procs/vanderdonckt_cr.pdf