

Towards Usability-Improving Design Patterns for Mobile Client-Server Computing

Bettina Biel, Volker Gruhn

University of Leipzig
Department of Computer Science
Klostergasse 3, 04109 Leipzig, Germany
[biel,gruhn]@ebus.informatik.uni-leipzig.de

www.lpz-ebusiness.de

9 June 2006

Abstract: The usability of a computer system can be improved by design of the user interface, and, as importantly, by design of user-system interactions. Our research presents usability-improving design patterns for different mobile computing client/server-architectures. In mobile computing, different types of client/server-systems exist, because software components can be distributed in different ways. Such general conditions of the architecture affect the pattern collection in a way that requires them to be shaped differently. This "work-in-progress" paper presents first steps towards a pattern system for usability-improving design patterns for mobile client-server computing.

1. Introduction

Usability is a non-functional quality attribute of a computer system. It describes from the users' point of view how easy it is to start working with a computer system, how efficient and reliable one can accomplish tasks and how pleased one feels about the whole system. Usability can be improved by skillful design of the user interface, and, as importantly, by the careful design of user-system interactions. Considering these interactions, we propose usability enhancing design patterns for the use by software engineers – in the area of mobile client/server systems.

In mobile computing, software components can be distributed in different ways, and c/s computing systems can be classified according to the distribution of their architecture components. Looking at how e.g. presentation logic, application logic, and data storage are distributed among clients and servers, one can differentiate between hybrid-online (does not always need a connection), always-online (connection dependent) and hybrid-offline systems in general. For example, a thin always-online client only presents a graphical user interface and totally depends on the server for all application logic. Therefore, all usability must be implemented on the server. While the usability challenges for a rich client are the same, they need to be solved in a different way since more logic is located on the client [2].

Due to these general conditions, architectural constraints will affect the patterns in a way that the usability-improving design pattern collection must take different solutions to one problem for different mobile architectures. To determine suitable architecture distribution classes, we can pose some basic architecture influencing questions, whose answers lead to architecture decisions, namely the frequency of communication, the location of data storage (client, server) and the distribution of logic. We will use that classification for structuring the pattern catalog.

For the use of the patterns by software engineers and software developers, the patterns must motivate why a technological solution is usability enhancing and especially provide information on programming issues as fine-grained as needed. To limit the scope of the patterns, we focus on interaction and leave interface issues aside (for example type of buttons, colors) and exclude device-specific input/output mechanisms.

We define the usability-improving design patterns for mobile computing as descriptions of architecture components that are customized to solve interaction-related (in contrast to interface-related) usability problems in a particular mobile c/s system.

Recapitulating, in the general context of mobile computing, the problems of the patterns are *technology* originated while solutions support software engineers in their ambition to provide a technological basis for enhancing mobile computing c/s systems' usability.

This paper will now cover related work and first patterns that are related to multi-channel access in always-online architectures.

2. Related Work

HCI-pattern collections and languages can be catalogued according to their interface and interaction relation. There is little discussion of mobile problems and solutions, i.e. most of the research focuses on window-based and browser-based GUI (the world-wide-web).

Regarding mobile computing interfaces, an interface pattern collection by Welie [3] and a more complex interface and interaction design pattern collection by Chung et al. [4] exist. While the first presents few interface best practices, the second presents a catalog of 45 pre-patterns that discusses application genres, physical-virtual spaces, interaction and systems techniques for managing privacy, and techniques for fluid interactions. They provide the broadest collection of patterns. As they give no detailed information on programming issues, as intended by our work, we will refer to that work for the motivation of certain patterns.

More detailed usability patterns were designed to confirm the relationship between usability and software architecture, by Bass et al. [5], Folmer and Bosch [6], and Folmer and Welie [7].

Bass et al. [5] identified usability scenarios and related patterns for improving the quality attribute usability. In some cases, there is still a need to prove architectural relevance and in some cases it is necessary to combine solutions, so the work can still be refined.

Folmer and Bosch [8] developed a framework to deduce usability patterns from usability attributes. Their set of summarizing attributes of the most commonly cited usability definition's criteria are *learnability*, *efficiency of use*, *reliability in use* and *satisfaction*. A layer in between patterns and attributes are *usability properties* which represent heuristics and design principles that were found to directly influence the system's usability; namely *accessibility*, *adaptability*, *consistency*, *explicit user control*, *guidance*, *minimal cognitive load*, *natural mapping*, and *providing feedback*. As architecture properties, they serve as interlink from abstract quality aims to properties and to patterns. Though the patterns are too general for our purposes, we can refer to them, refine some, and use their framework as a theoretical background.

Folmer and Welie [7] extended some of those general patterns to a detailed level that clarifies the relation of the patterns to software architecture. So, our work follows their approaches and their research, with a focus on the field of mobile computing they have not concentrated on.

After this short overview of related work, the next section presents how the patterns exactly relate to usability, the thumbnails for the first patterns and the *Dialog Flow Manager* pattern.

3. Pattern Collection

3.1. Framework

In the section Related Work (section 2) we already presented Folmer and Bosch's framework that we relate our patterns to. Hence, the patterns are derived from usability properties [8] and linked directly to the usability attributes *learnability*, *efficiency in use*, *reliability in use*, and *satisfaction*. Patterns can be derived from one or more properties. They can improve or impair one or more attributes, and a specific combination of patterns improves or impairs one or more usability attributes, too; i.e. the patterns interact with each other and then they can have different effects on attributes.¹

The next paragraph one presents thumbnails shows the architectural and technological common context of the patterns, while the following. After this, the pattern *Dialog Flow Manager* is described completely.

3.2. Common Context

The common context of the patterns is a web application with a strict implementation of the Model View Controller (MVC) pattern in an always-online architecture.

Common architectural constraints in the pattern's *always-online environment* consist of:

- *Frequency of communication*: frequently, every mask must be sent from the server to the client
- *Location of data storage*: all data is held on the server, the client simply acts as graphical user interface, the system depends totally on the network
- *Distribution of logic*: completely by the server

Further technical premises regard common participants of this MVC-solution are channel servlets and actions. The channel servlet takes care of one presentation channel and receives a request from the client, extracts events and forwards them.

¹ That topic relates to a discussion on the effectiveness of patterns and how we can evaluate the patterns; here it should just be mentioned that it is important that an implementation of a pattern only means a technological support of usability *on principle*.

3.3. Thumbnails

An insight into the purpose and the connection of the patterns is presented by the introductory thumbnails table.

Pattern Name	Problem	Solution
Dialog Flow Manager	In contrast to window-based applications, web applications are stateless, page-oriented and use a request-response mechanism. Systems should be able to encapsulate tasks and remember how far users have come in performing a task, though. How do you ensure that nested dialogs can still be handled in an intuitive way?	Introduce a central dialog manager that is responsible for managing the dialog flow.
Server-Side Transcoding	Different mobile devices have different input/output capabilities and thus require/produce data in different formats. To prepare content for only few devices would exclude many users, but development and maintenance of the different versions would be very time-consuming. How do you adapt dynamic content automatically to device properties?	Implement server-side generation of content using an annotated abstract specification about the contents and instructions for transcoding.
Server-Side Pagination	Mobile devices come in different sizes. A task that can be finished in one step on a desktop client may have to be subdivided into several steps for a small-screen mobile device. Instead of defining all possible paginations manually, how do you split the task	Implement server-side pagination of content using an annotated abstract specification about specific contents and its priorities, sizes and grouping.

Pattern Name	Problem	Solution
	automatically?	
Multichannel Access	Mobile devices' capabilities range too widely to manually provide and maintain content for all of them. A web application should be accessible via each channel and present content that is tailor-made. How do you know which device is accessing the application and provide the interpretable masks for the device following the dialog flow in smaller steps?	Use a profile database to look-up device properties, break up the dialog flow into a suitable number of steps and generate the masks afterwards. ²

3.4. Pattern Example

Name: Dialog Flow Manager

Example: For example, a user reads an article in a online forum, wants to reply to it and therefore clicks the reply-button. After the click, the system checks whether the user is logged on. If not, it presents a dialog mask that asks the user to log on or to register. Wanting to write a reply as soon as possible, a new user has to register. After completing the registering process, the user expects to be presented with the reply mask. But instead, he is redirected to the home page, in order to login, find the article again and finally write the reply.

This behaviour is contradictory to the user's expectations from window-based applications. It leads to confusion and annoys the user. Therefore it affects the usability attribute *user satisfaction* negatively. Since work has to be done twice, the usability attribute *efficiency* is also degraded. For predictability, a system should be able to encapsulate tasks and remember how far users have come in performing a task.

There are different ways to control such dialog flows. Commonly known is the solution, that every executed action is aware of it's predecessor and successor. Dialog flow decisions can be made, although only in this particular context. The problem is, if nested tasks are encapsulated into several contexts, the pre- and successor must be stored dynamically.

² The Multichannel pattern presents a solution that can address all of the above mentioned problems and hence will introduce a more complex solution.

Problem: In contrast to window-based applications, web applications are stateless, page-oriented and use a request-response mechanism. Systems should be able to encapsulate complex tasks and remember how far users have come in performing a task. How do you ensure that nested dialogs can still be handled in an intuitive way?

Forces:

- When complex dialog flows are implemented using pre/successor-solutions, different alternatives have to be implemented manually, which scales up poorly.
- It is necessary to specify a precise model of dialog flows before programming starts. That is, it is not possible to just start writing lines of code, like it was possible when implementing an ad hoc pre/successor-solution.
- Unlike a state-less pre/successor-solution, a dynamic solution demands the implementation of server-side states, with all corresponding problems.
- The finer-grained the actions, the more communication between client and server becomes necessary. In a mobile computing always-online environment such a fine-grain dialog flow implementation means a lot of communication (time, security, connectivity problems increase). Therefore, too fine-grain specification causes performance overhead. Hence, granularity of actions must be defined in a well-balanced way.
- It should be possible to integrate a dynamic solution in an existing pre/successor-solution – for extension purposes or if only parts of an application need dynamic processes.

Solution: Introduce a central dialog manager that is responsible for managing the dialog flow.

An application's dialog flow (all possible dialog paths through an application, consisting of GUI-masks and business-logic calling actions) is divided into modules (units for one task). These modules can be nested, i.e. all modules can refer to sub-modules.

The specification of the dialog flow in the modules and between the modules is not contained in the sourcecode, or in the actions. It is stored separately using a specific notation that is parsed and processed at run-time.

Only a central dialog manager can access that dialog flow information and manages stacks that store a user's dialog state (i.e. the current dialog mask and the set of possible transitions to other dialog masks). Hence it controls and manages a user's dialog flow at run-time according to the definition and is always aware of the state of the user's dialog.

Here, the stack is used. If a sub-module is called, the manager stores the current and previous states of the user's dialog. When it has terminated the previous state (in the previous module) is restored.

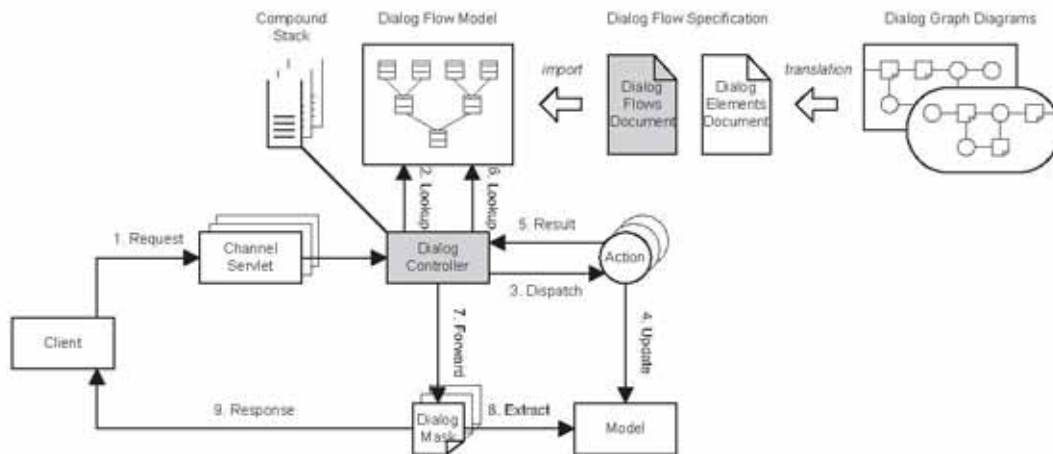


Figure 1: Coarse structure of the Dialog Control Framework [1]

The diagram in the Figure 1 presents a know uses' implementation [1].

Example resolved: In the example above, the user browses an online forum and clicks on a link to read an article. He wants to reply to it and therefore clicks the reply-button.

The definition of the current module *forum* refers to a module *user authorization*, because users must be logged on. The dialog flow manager calls a GUI-mask that asks the user to log on or to register, as he is not logged in yet.

The user is new and has to register.

Therefore, the module *user authorization* refers to the module *create new account*. The dialog state on the server has now changed from the *forum* via the *user authorization* module to the *create new account* module. On the stack, there are the three user's states stored: on top the *create new account* state information, then the *user authorization* information, then the information regarding the *forum*.

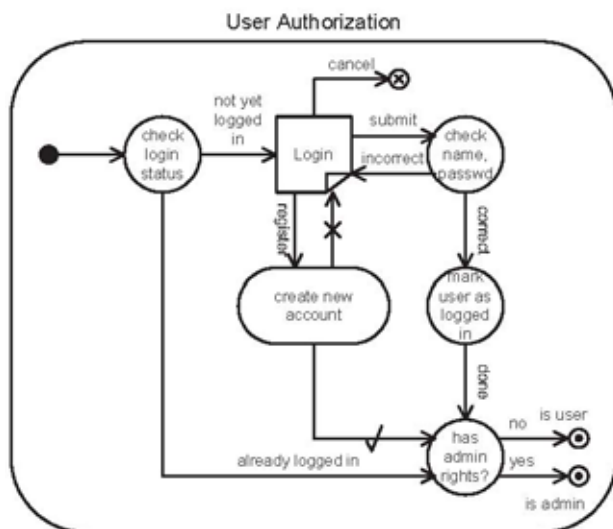


Figure 2: User Authorization Module [1]

The user registers.

The *create new account* module is terminated successfully, and the first state on the stack is deleted. Now the *user authorization* module is on top again. As the user is logged on now, the module can be terminated successfully, too, leaving the *forum* module on top. Here, the information about the last presented GUI-mask and its successor is stored and called again.

The description of the dialog flow can be done using the Dialog Flow Notation

[1], that has been developed for this purpose. In the Figure 2, a *user authorization module* refers to a module *create new account*.

Consequences:

- Complex dialog flows can be implemented using the presented solution, though the solution might not scale up that good because of the stacks for each user. Therefore, the stored information must be minimal.
- The specification of a precise model of the application's dialog flow means a lot of preparation work (that has to be carried out in any case). But once defined and implemented, dialog flows are easy to maintain.
- The implementation of server-side states leads to different problems because browser functionalities enable users to evoke different server-side problems. Users can execute operations that the presentation logic can neither allow nor forbid, by closing the browser window or requesting an external page outside the application, by requesting a page through a previously set bookmark or a link from an external page, by cloning the browser window to receive two user interfaces for the same session, or by clicking the Reload/Back or Forward buttons. The presentation logic often can not instantly detect that one of the operations was carried out, but only conclude it from unexpected requests. This can cause serious problems for the application. Not just because they are unpredictable and uncontrollable, but rather because they break the synchrony between the server and client-side state of the dialog system.

Usability Context and Consequences: This pattern is derived from the software architecture property *accessibility* [6].

Its usability attributes are affected in different ways:

- [+] *Learnability*: The improvement of predictability and the reduction of cognitive load ease the (prompt) construction of a mental model of the dialog.
- [+] *Reliability in use*: The user makes less mistakes.
- [+] *Efficiency in use*: Users finish a task faster because they do not have to do things twice. Learnability also relates to this usability attribute.
- [0] *Satisfaction*: The user does not have to do things twice, and it is easier to work with the application in general. Therefore, satisfaction will not be decreased.
- [-] *Satisfaction and Efficiency in use, IF...* dialog flows and actions are defined too fine-grained, more communication between the system components is necessary, so there can be a performance overhead. Especially in an always-online-environment that will cost time. This could decrease *satisfaction*, if users subjectively feel they have to wait too long.

Evidence/Known Uses: Spring Framework [9], ARGUS [10]

Literature: [1]

Management: *Author:* Bettina Biel, *Credits:* Matthias Book, *Creation-date:* 23-01-2006, *Last-modified:* 9-Jun-2006, *Revision-number:* Version 3

4. Conclusion

We presented the first steps towards a pattern system for usability-improving design patterns for mobile client-server computing. One has to keep in mind that applied patterns are no guarantee for a good usability level, for it always matters how a certain implementation was carried out in detail and how implemented solutions interact with each other.

Our future work comprises the identification and development of more patterns, for example the search in huge amounts of data in a c/s system, feedback related patterns, undo/redo/back and adaptability related patterns. We will organise them in a pattern system and validate the approach by using the patterns in architectural analysis.

5. Acknowledgments

I would like to thank Tim Wellhausen who shepherded this paper for EuroPLoP 2006, because his comments and questions helped to improve the work.

The Chair of Applied Telematics/e-Business is endowed by Deutsche Telekom AG.

6. Bibliography

1. Book, Matthias and Gruhn, Volker, *Modeling Web-Based Dialog Flows for Automatic Dialog Control*, in *19th IEEE International Conference on Automated Software Engineering (ASE 2004)*. 2004, IEEE Computer Society Press.
2. Book, Matthias, Gruhn, Volker, Hülder, Malte, and Schäfer, Clemens, *Der Einfluss verschiedener Mobilitätsgrade auf die Architektur von Informationssystemen*, in *5. Konferenz Mobile Commerce Technologien und Anwendungen (MCTA2005)*. 2005: Augsburg, Germany.
3. Welie, Martijn van, *Patterns in Interaction Design*. 2005.
4. Chung, Eric S., Hong, Jason I., Lin, James, Prabaker, Madhu K., Landay, James A., and Liu, Alan L., *Development and Evaluation of Emerging Design Patterns for Ubiquitous Computing*, in *DIS2004*. 2004.
5. Len Bass, Bonnie E. John, Jesse Kates, *Achieving Usability Through Software Architecture*. 2001, Software Engineering Institute (SEI), Carnegie Mellon University: Pittsburgh, USA.

6. Folmer, Eelke, Gulp, Jilles van, and Bosch, Jan. *A framework for capturing the relationship between usability and software architecture*. in *Software Process: Improvement and Practice*. 2003.
7. Folmer, Eelke, Welie, Martijn van, and Bosch, Jan, *Bridging Patterns - an approach to bridge gaps between SE and HCI*. *Journal of Information and Software Technology*, 2005.
8. Folmer, Eelke and Bosch, Jan, *Architecting for Usability*. *Journal of Systems and Software*, 2004(70-1): p. 61-78.
9. Framework, Spring. *Spring Framework*. 2006 [last visited 16.05.2006]; Available from: <http://www.springframework.org>.
10. Book, Matthias and Gruhn, Volker, *Experiences with a Dialog-Driven Process Model for Web Application Development*, in *29th Annual International Computer Software and Applications Conference (COMPSAC 2005), Workshops and Fast Abstracts: Workshop on Model Driven Agile Development II - Component-Based Agile Development*. 2005, IEEE Computer Society Press.