User Interface Master Detail Pattern on Android

Thanh-Diane Nguyen, Jean Vanderdonckt
Louvain Interaction Lab., Louvain School of Management (LSM), Université catholique de Louvain, Place des Doyens, 1
B-1348 Louvain-la-Neuve (Belgium) – {thanh-diane.nguyen, jean.vanderdonckt}@uclouvain.be

ABSTRACT
The purpose of this work is to understand some existing user interface (UI) patterns and to adapt them to the constraints of mobile devices running on the Android system. We focus mainly on the Master/Detail pattern and on the surrounding patterns. The contributions are multiple: our background study consists of a brief summary of the principles of some existing user interface patterns. Based on it, we provide an adapted version of each pattern targeted to mobile phones through a framework called MandroiD. We will also present a basic case study application that takes advantage of the framework. This application is developed with Android guidelines in mind. Indeed, one of our goals is to provide the reader with some knowledge about Android applications development. Limitations of general mobile devices (e.g., the small screen) require of "reducing" homogeneous elements. MandroiD overcome theses constraints. A statistical analysis is conducted on the developed mini-application. Evaluation of it shows a general satisfaction concerning the ergonomy of the application by various users.

Author Keywords
User interface; patterns; mobile development; Android.

ACM Classification Keywords
D2.2 [**Software Engineering**]: Design Tools and Techniques – *Modules and interfaces; user interfaces*. D2.m [**Software Engineering**]: Miscellaneous – *Rapid Prototyping*; *reusable software*. H5.2 [**Information interfaces and presentation**]: User Interfaces – *Graphical user interfaces (GUI); style guides; user-centered design*. H5.3 [**Information interfaces and presentation**]: Group and Organization Interfaces – *Evaluation/methodology*.

General Terms
Algorithms; Design; Languages.

INTRODUCTION
As we can observe in our everyday life, mobile devices evolved with the introduction of high level development capabilities. Modern computing and interfaces design activities have to take into account the constraints of mobile devices [1,2,11], which often have a small-sized screen and no physical controller, such as a keyboard and/or mouse.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EICS'12, June 25–26, 2012, Copenhagen, Denmark.
Copyright 2012 ACM 978-1-4503-1168-7/12/06...$10.00.

The problem of today's literature is that most of the interface descriptive and generative patterns [9] were designed for desktop environments [8,10,15] and therefore lack of support for mobile-related operations related to generative patterns. For example, the size of the screen is not a concern (nothing is said about small-sized screens with low resolutions). Ubiquitous computing is not supported by those patterns [11,14]. Design patterns can be used to capture essential problems of different "sizes". Moreover, the using of pattern for documenting design knowledge "divides a large problem area into a structured set of manageable problems" [3]. The purpose of this work is to provide adapted versions, with an evaluation, of some existing design patterns based on Object Oriented Method that can be very useful to developers and end users.

We decided to focus on Android-based mobile systems instead of iPhone devices (iOS-based). We motivate our choice by the fact that Android development is accessible and free, with a great support from the community. Furthermore, Android could not require any add-learning of specific language: Android applications are written in Java, which is a widespread programming language known by all developers. iPhone application is relied on Objective-C which can be less learned in academic classes by its material requirement. Nevertheless, the guidelines introduced in this document are valid for both systems.

Structure
This paper is organized as follows. The first part focuses on a background study of some patterns introduced in [13]. We have chosen three main patterns: the Master/Detail, the Order and the Filter patterns. This choice is motivated by the fact that there are very common patterns that are, according to our experience, generally poorly supported in mobile computing. Furthermore, some of them (e.g. the Master/Detail pattern) involve a recursive design whose conception is a very interesting challenge. In the second part, we propose a framework that could be used by programmers for implementing UIs with that kind of patterns.

Afterwards, we will take a basic application as a case study illustrating the framework. The objective is to prove that it is usable for real applications, such as a car-configurator application targeted for customers of car dealers (e.g. Audi, BMW, etc.). Finally, we present a statistical analysis which assesses the overall quality of the developed interfaces, according to some criteria that have been evaluated by external and non-technical users.

RELATED WORK
In this section, we explore some existing design patterns that need to be implemented on Android systems.

This background study is based on [12,13], which proposes a **Presentation Model**. This approach is "*a methodological guide representing the user interface in an abstract and design independent way*".

We do not explore all the patterns presented there, but we focus on a combination of some patterns which are [8, 10]: the Master/Detail, the Filter and the Order criterion against a given Population. After this exploration, we discuss some guidelines for designing user interfaces on mobile devices. Again, we restricted our study to some of the most important rules for conceiving high quality interfaces.

The definition of Filter and Order criterion patterns are obvious. **Filter pattern** is used for defining custom criteria that allow selecting some parts of a population. **Order criterion pattern** is mainly used for sorting elements of a population in ascending or descending order. In the tool built in this work, we stay generic by creating a generic filter object that could be extended according to any specific requirement, and by letting developer create any kind of ordering criteria. These auxiliary patterns are also similar to the current available collection of pattern-catalogues in the HCI domain [7, 16].

Population

A population unit is an abstraction defined for representing set of elements. Typically, populations are implemented as lists or arrays.

Master/Detail

The Master/Detail pattern is illustrated in Figure 1. It is the most interesting pattern to present because it combines the concepts defined previously: the Master part, located on the left of Figure 1, consists of a Population unit, which can also be combined with filters and ordering criterion. The Detail part, on the right, can be any graphical component required for presenting the details of the selected element of the list. In the framework presented in the next section, we still stay generic so that any user-defined component can be a member of the list: even non-trivial elements are allowed. Similarly, the detail can be any specific component, even a nested Master/Detail structure.

Android style guidelines

Developing on Android means creating and using Activities, which correspond to the "windows" of the applications on desktop computers. Although this mechanism is aimed at allowing modular designs, it should be used with parsimony because activities are *stacked* in the system. The end-user can navigate between activities by pressing the *Back* button of his device. Consequently, minimizing the amount of activities started is a main goal of our framework.

Generally, and similarly to desktop applications, we have to keep the interactions between the user and the system as clear as possible in order to not create unnecessary confusion. This property has to be enforced on Android system because devices with this operating system, OS, could have neither physical keyboard nor mouse for navigating.

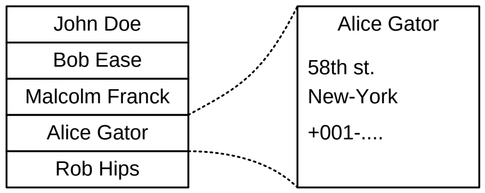


Figure 1. Master Detail.

Therefore, we try to keep the interactions limited to simple "click" actions and to predefined components (e.g. instead of entering the date manually, a widget should be used). The last guideline is more a recommendation on the Android philosophy which tends to ensure that only the essential information are shown, with the least superfluous data possible. Unlike for desktop environment, there is no free-space on the left and the right of wide screens for presenting additional and non-essential data. This observation has to be taken in consideration while conceiving graphical user interfaces.

MANDROID: A JAVA FRAMEWORK FOR IMPLEMENTING THE MASTER/DETAIL PATTERN

The purpose of this paper is not to compare different environments that generate GUIs for mobile devices, but rather to see how patterns for desktop (implementation in OlivaNova [10]) could be transferred to another system for other platform at what cost. In addition, the usability of resulting GUIs is work to be examined. Then, one result of this work is a Java framework called MandroiD. It is targeted for implementing interfaces through the Master/Detail pattern. Its name simply represents a combination of the initials of the pattern name with the android word. The architecture of the classes composing the framework is shown in Figure 2.

LayoutProvider

We start by defining an abstraction whose purpose is to declare a common behavior for displaying complex graphical object representation. We call this class `LayoutProvider` and define a method `getLayout()` that should be called by the interface creation procedure. Thanks to this class, we are able to define any kind of complex layout objects, and to reuse them in the remainder of the framework, in composite elements. A `View` is an Android object referencing graphical component. An `Activity` corresponds to a *window* of the application, it is needed because the definition of the `getLayout()` method in the subclasses generally build components that all require this reference when they are created. For instance, a user-component providing a date-chooser in a white rectangle block could be implemented as a `LayoutProvider`.

Population

The next brick of the framework is the `Population` class. It is used for representing a *list* of graphical elements to the user, and therefore it extends the `LayoutProvider` class. A list is composed of zero to many `ListElement` objects which are aimed at storing pairs of graphical elements and actions that are triggered when the element is clicked. The next brick of the framework is the `Population` class.

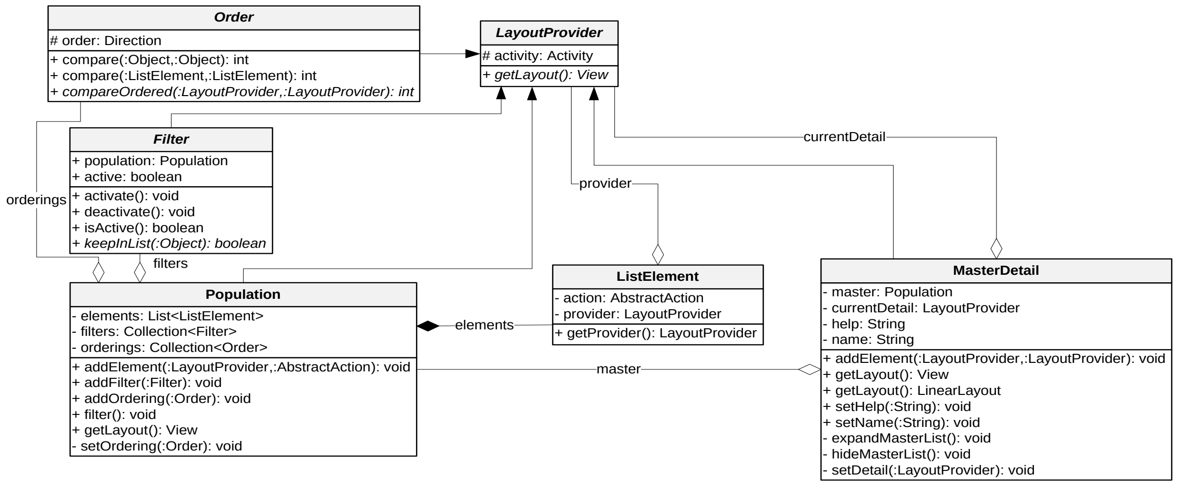

Order
# order: Direction
+ compare(:Object,:Object): int
+ compare(:ListElement,:ListElement): int
+ compareOrdered(:LayoutProvider,:LayoutProvider): int
LayoutProvider
# activity: Activity
+ getLayout(): View
Filter
+ population: Population
+ active: boolean
+ activate(): void
+ deactivate(): void
+ isActive(): boolean
+ keepInList(:Object): boolean
orderings
filters
currentDetail
provider
Population
- elements: List<ListElement>
- filters: Collection<Filter>
- orderings: Collection<Order>
+ addElement(:LayoutProvider,:AbstractAction): void
+ addFilter(:Filter): void
+ addOrdering(:Order): void
+ filter(): void
+ getLayout(): View
- setOrdering(:Order): void
elements
ListElement
- action: AbstractAction
- provider: LayoutProvider
+ getProvider(): LayoutProvider
master
MasterDetail
- master: Population
- currentDetail: LayoutProvider
- help: String
- name: String
+ addElement(:LayoutProvider,:LayoutProvider): void
+ getLayout(): View
+ getLayout(): LinearLayout
+ setHelp(:String): void
+ setName(:String): void
- expandMasterList(): void
- hideMasterList(): void
- setDetail(:LayoutProvider): void

Figure 2. The UML Class Diagram of Mandoid.

It is used for representing a *list* of graphical elements to the user, and therefore it extends the `LayoutProvider` class. A list is composed of zero to many `ListElement` objects which are aimed at storing pairs of graphical elements and actions that are triggered when the element is clicked.

Filters and Orderings

Filters and ordering criteria can be attached to populations through `Filter` and `Order` objects, respectively.They both provide layouts in order to be presented to the user, for activation and deactivation They are *abstract* classes, and consequently concrete filters have to be defined according to the needs of the application. Defining a new filter means implementing the `keepInList()` method which returns `true` if the given element should stay in the list, and `false` otherwise. Defining ordering criteria can be done by extending the `Order` class and implementing the `compareOrdered()` method. Once filters and ordering critera are attached to populations with the `addFilter()` and `addOrdering()` methods, the framework manages their display, their activation (through user-input) and deactivation.

Master/Detail

The most relevant point to present is the `MasterDetail` class. This class manages the display of elements according to the Master/Detail pattern which has been described in the previous section. A `MasterDetail` is defined, among others, by a `Population` which corresponds to the *master* part, and by a `LayoutProvider` corresponding to the *detail* part. The most interesting thing comes from the type of the detail part, which can be any `LayoutProvider` object, including a nested `MasterDetail`. The framework can then handle (potentially) infinite recursion. Pairs of master elements and corresponding details can be added with the `addElement()` method which is responsible of inserting the element in the list, and of creating the event handler that will update the detail part when the element is selected by the user.

The `expandMasterList()` and `hideMasterList()` are used internally for replacing the population by an "*expand*" control, in order to avoid the graphical structure becoming too big because of several nested master-details. In this section, we presented the internal architecture of the framework. The next section presents a case study application relying on MandroiD.

A DETAILED CAR REPOSITORY

For this case study, we built a basic application which takes advantage of our framework, MandroiD, for conceiving its graphical UI. The purpose of the application is to provide detailed information describing the configuration of each model of car of a dealer. The underlying intension is to be used by potential buyers who are interested in exploring all the details of their future car. On a strictly graphical point of view, the first screen of the application is the one asking the user to choose a dealer. Each screen contents follow general ergonomic rules [4]:

1. Elements of a window have to be align.
2. Create a screen balanced.
3. Unicity of elements provides better overview.
4. Insert regularity and harmony in the way of a set ordered elements from a central point.

The relevant patterns for this first step are the Master/Detail and the Order ones. First, the user is able to sort alphabetically the brands and, secondly, when a brand is selected, the detail (i.e. the next step of the car configuration) appears. If the user wants to sort the models in descending order, the result is in Figure 4.A.

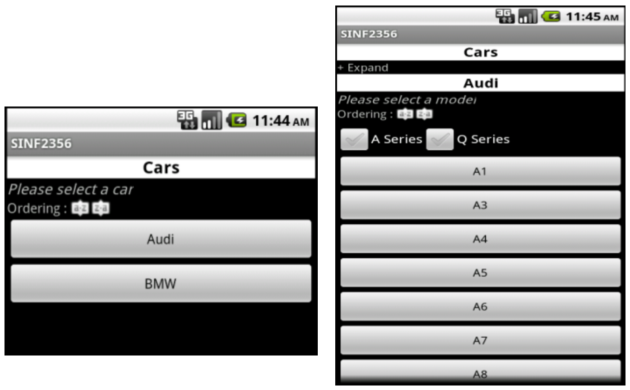


Figure 3. Dealer selection (3.A) - Model selection (3.B)

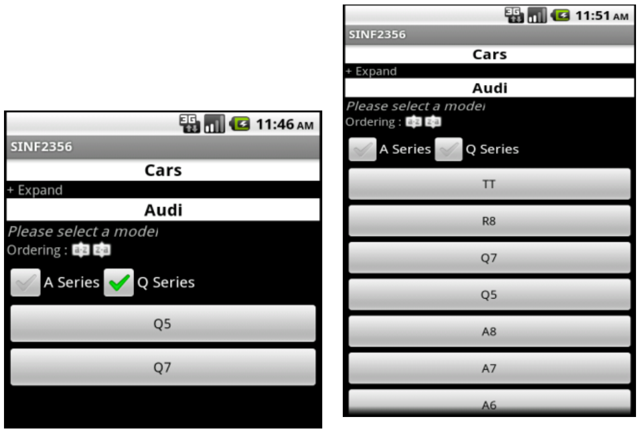
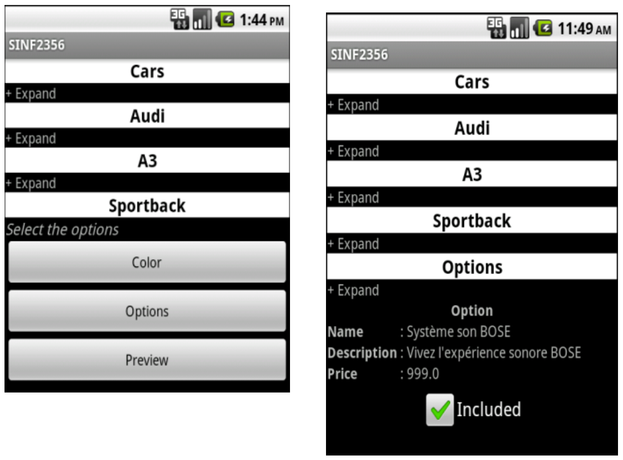


Figure 4. Model selection: DESC order (4.A) -filtering (4.B)

Then, the user has to select a model of car represented by standard button of Android System. Basically, this step is implemented the same way as the previous one, using Master/Detail and ordering, but it also contains the Filter pattern. The latter is used, in this case, to keep only a specific branch among the different models (i.e. the population). For instance, if the user selects the "Q Series" checkbox (see on Figure 4.B).

Once the model is selected, the resulting detail concerns the selection of the body style of the car. This step uses a nested Master/Detail pattern. Therefore, it is not illustrated. Next, the user can specify the options and the color that s/he wants as shown on the Figure 5.A. The color and options buttons (i.e. masters) render the same kind of view (i.e. detail) when clicked. So, we only focus on the "Options" one. Typically, the detail of this button is a list of options, which, once again, use the Master/Detail pattern. When an option is selected, a screen allowing the user to select it appears.

To get back to the options list, the "+ Expand" link can be clicked. This link is present each time the Master/Detail pattern is used in order to get back to the master. Finally, a preview of the car is available.



Name : Système son BOSE
Description : Vivez l'expérience sonore BOSE
Price : 999.0
Included

Figure 5. Options selection (5.A)-Option inclusion (5.B)

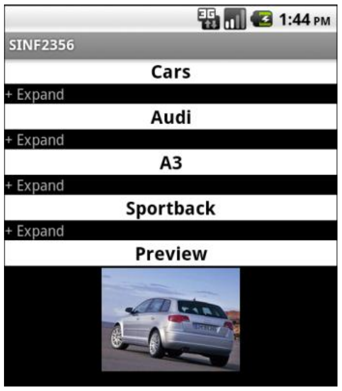


Figure 6. Preview

On a technical point of view, the filling of the application is done automatically thanks to our XML parser compatible with Android. Indeed, all packages available in standard Java are not part of the Android SDK and we had to develop a tool to help us parsing textual data in order to make the application more flexible. In this case, the missing package was *javax.xml*.

Thanks to the developed tool, the data is fetched from a XML-file and then presented on the user interface. This strategy enables to update the data about cars and even add new models and/or brands (without having to recompile the application). The idea behind the algorithm is the following: each time we meet a node in the XML-file we check its value and create the corresponding elements with the attributes specified in the XML-file. Example: a node with value "model" causes the creation of a Master element. Every node that follows and whose value is different from "model" concerns the model previously created (we go through the XML-file line by line). Then, depending on the values of the next nodes, masters and details elements are created and added to previous elements. If the value is equal to "ordering" or "filter", the corresponding patterns are initialized on the population of the appropriate master. This XML parser helped us to maintain our application clean and well structured. Those two points are very important to enforce the quality of the user interface and to efficiently work in team.

STATISTICAL ANALYSIS

This statistical analysis is based on the Post-Study System Usability Questionnaire (PSSUQ) [5]. This method provides a set of questions (see Table 1) that users have to answer after processing our case scenario [17]. Each question consists of a 5-point Likert scale [6]. The questions are grouped in 5 categories:

- Usability of the system (SYSUSE)
- Quality of the information (INFOQUAL)
- Quality of the interaction (INTERQUAL)
- Overall of the system (OVERALL)
- Ergonomy (ERGONOMY)

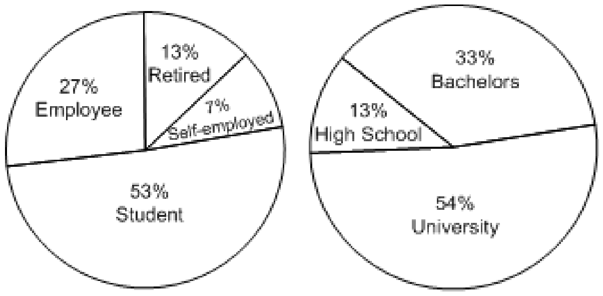


Figure 7. Occupation of testers (7.A) - Level of studies of testers (7.B)

The scenario is the following:

1. Find the options available on Audi Q7.
2. Look at the beautiful shape of Audi A5 Sportback.
3. What is the price of the Audi TT Roadster's GPS?
4. What are the colors available of BMW serie 1?

For this analysis, we did not take all the questions of PSSUQ as-is because they were not applicable for our application. Each question has to be answered with an evaluation number from 1 to 7. 1 means: "I totally disagree" and 7 means: "I totally agree". PSSUQ is accurate because the questions it provides are suitable for scenario-based usability test. To collect the data that serve to this analysis, we create first a set of action items that users have to do, and then ask them to answer to question set. The testers we found are friends or family of us. We found 15 peoples, 53% of them are woman and 47% are man. Figure 7.A and Figure 7.B show the current occupation of testers as well as their level of education.

As a result, we can see that the set of testers are mainly student but other categories are represented as well. We can also put out that our testers have high level of study. Figure 8 shows the results of the answers of the testers.

The first observation is that the average score of every category is high. The master details pattern is interesting while programming on mobile device. Nevertheless, the standard deviation of the fourth first categories is big because our application needs to be improved with new features. The standard deviation of ERGONOMY is not high though, thus suggesting that participants are generally satisfied concerning the ergonomy of the application.

This is one of the most important observations because it shows that our implementation of nested master/details does not result in losing the user in complex hierarchies, thanks to the "expand" mechanism which keeps the "path" of his current location clearly visible at any time.

This was challenging because of the limited screen-size of the devices. The black background behind white texts may also be discussed, but that is the default configuration for applications on Android systems. This analysis showed that it did not confuse any user.

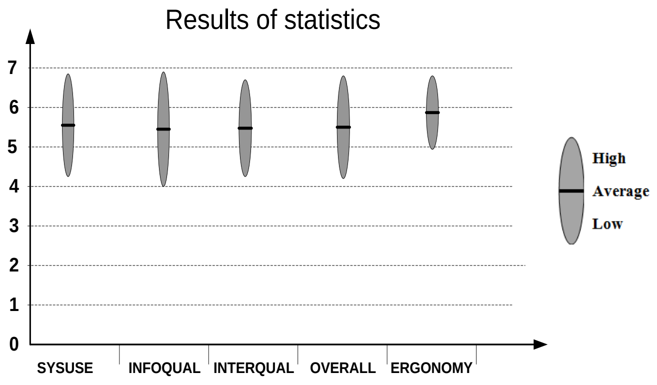


Figure 8. Results of statistics

CONCLUSION

During this work, we explored some existing design patterns and adapted them to the criteria of the mobile devices running on Android systems. This paper is aimed at determining to what extent a java framework could support automated generation of graphical UIs or mobile devices based on pattern approach.

The main contribution of this work is a framework called MandroiD, which supports generative patterns for mobile devices. It provides specific constructs for building three commonly used patterns; the most impressive is the Master/Detail because it introduces recursive structure in graphical interfaces. The underlying problem was the limitations of general mobile devices, which have a small screen on which a minimal set of information is available at any time. We achieve the goal of minimizing the accessible information set thanks to an adequate use of "reducing" and "expanding" controls of the list, so that the user keeps the focus on the part of the application s/he is using.

We also proved that the framework is usable in practice, firstly by providing an application taking advantage of it, and secondly with the interface evaluation part which shown that although some points could be improved, the implemented patterns are convenient for being used by most of the users.

The future work can be to extend the evaluation with a larger community of developers using the framework. This evaluation will show their feeling on development practices with this framework and any suggestion to improve it. Another future work can be a comparison of different other implementation approach of Mandroid on other frameworks or systems with an evaluation of their performances.

ACKNOWLEDGMENTS

The authors would like to thank L. Carlier, Y. Hanson, J. Janssens, D. Jeusette, and A. Moulai for implementing Mandroid and for testing it with various users. Our thanks also go to all the volunteers that filled in the surveys

Question ID	Question statement	Statistics per questions				
		Average	Median	Average of deviations	Standard deviation	Confidence
1	Overall, I am satisfied with how easy it is to use this system	5,67	6,00	0,84	1,11	0,56309217
2	It was simple to use this system.	5,40	6,00	0,91	1,12	0,567407115
3	I could effectively complete the tasks and scenarios using this system	5,20	5,00	1,15	1,47	0,745719046
4	I was able to complete the tasks and scenarios quickly using this system.	5,80	6,00	0,80	1,08	0,54772223
5	I was able to efficiently complete the tasks and scenarios using this system.	5,47	6,00	0,97	1,19	0,600812099
6	I felt comfortable using this system.	5,27	5,00	1,22	1,53	0,776168992
7	It was easy to learn to use this system.	5,80	6,00	0,69	1,01	0,513239047
8	I believe I could become productive quickly using this system.	5,47	6,00	0,97	1,19	0,600812099
9	Whenever I made a mistake using the system, I could recover easily and quickly	5,33	6,00	1,24	1,54	0,780868343
10	The information (on-line help, on-screen messages and other documentation) provided with the system was clear	5,67	6,00	0,89	1,05	0,529610677
11	It was easy to find the information I needed	5,40	6,00	0,99	1,18	0,598778888
12	The information provided for the system was easy to understand	5,13	5,00	1,21	1,51	0,761897047
13	The information was effective in helping me complete the tasks and scenarios.	5,07	5,00	1,27	1,67	0,843916033
14	The organization of information on the system screens was clear.	5,60	6,00	0,93	1,24	0,62858689
15	The interface of this system was pleasant.	5,27	5,00	0,95	1,22	0,618810449
16	I liked using the interface of this system.	5,53	5,00	1,10	1,25	0,630523989
17	This system has all the functions and capabilities I expect it to have.	5,47	5,00	0,90	1,06	0,536474169
18	Overall, I am satisfied with this system.	5,07	5,00	0,89	1,16	0,588507476
19	I always know where I am and how to go where I want	5,93	6,00	0,63	0,88	0,447213328
20	Colors are chosen in order to let information visible	5,60	6,00	0,69	0,83	0,419057927

Table 1 : Questions and scores

REFERENCES

1. Abrahão, S., Iborra, E., and Vanderdonckt, J. "Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool." In E. Law, E. Hvannberg, and G. Cockton (eds.), *Maturing Usability: Quality in Software, Interaction and Value*. Vol. 10, Springer, London, 2008, pp. 3-32.
2. Aquino, N., Vanderdonckt, J., Condori-Fernández, N., Dieste, Ó., and Pastor, Ó., "Usability Evaluation of Multi-Device/Platform User Interfaces Generated by Model-Driven Engineering." In *Proc. ESEM'2010*-ACM Press, New York, 2010, Article #30.
3. Erik G. Nilsson. "Design patterns for user interface for mobile applications.", *Adv. Eng. Softw*. 40, 12 (December 2009), Oslo, Norway, pp. 1318-1328.
4. Galitz, W., "The essential guide to user interface design, an introduction to GUI design principles and techniques." Wiley Computer Publishing, Indianapolis, Inc., 1997
5. Lewis, J. R. "*IBM* Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use." IBM, Human Factors Group, Boca Raton, FL, 1993.
6. Likert, R. "A technique for the measurement of attitudes." *Archives of Psychology 22*, 140 (1932), pp. 1–55.
7. Mobile UI Pattern (04/05/2012) http://mobile-patterns.com/
8. Molina, P.J., Meliá, S., and Pastor, O. "Just-UI: A User Interface Specification Model." In *Proc. of CADUI'2002.* Kluwer Academics, Dordrecht, 2002, pp. 63-74.
9. Molina, P.J., Meliá, S., and Pastor, O. "User Interface Conceptual Patterns." In *Proc DSV-IS'2002*. Springer-Verlag, Berlin, 2002, pp. 159-172.
10. Molina, P.J. "User interface generation with OlivaNova model execution system." In *Proc. of IUI'2004*. ACM Press, New York, 2004, pp. 358-359.
11. Mori, G., Paternò, F., Santoro, C. "Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions." *IEEE Transactions on Software Engineering* IEEE Press Piscataway, NJ, USA, 30, 8 (2004), 507–520.
12. Pastor, O. "Generating User Interfaces From Conceptual Models: A Model-Transformation Based Approach." In *Proc. CADUI'2006*. Kluwer Academics, Dordrecht, 2006, pp. 1-14.
13. Pastor, O. and Molina, J.C. "MDA in Practice: a Software Production Environment Based on Conceptual Modeling." Springer-Verlag, Berlin, 2007.
14. Vanderdonckt, J. "Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures." In S. Buraga and I. Juvina (eds.), *Proc. ROCHI'2008, (Iasi, 18-19 September 2008)*. Matrix ROM, Bucarest, 2008, pp. 1–10.
15. Vanderdonckt, J. and Montero, F., "Generative Pattern-Based Design of User Interfaces", *Proc.* PEICS'2010
16. Van Welie pattern catalogue (05/03/2012): http://www.welie.com/patterns/
17. Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., Wesslén, A. Experimentation in Software Engineering: An Introduction. Volume 6 of International Series in Software Engineering. Springer, 2000.