Evaluation of User-Interfaces for Mobile Application Development Environments

Florence Balagtas-Fernandez and Heinrich Hussmann

Media Informatics Group, Department of Computer Science,
University of Munich, Germany
{florence.balagtas,heinrich.hussmann}@ifi.lmu.de

Abstract. This paper discusses about the different user-interfaces of mobile development and modeling environments in order to extract important details in which the user-interfaces for such environments are designed. The goal of studying such environments is to come up with a simple interface which would help people with little or no experience in programming, develop their own mobile applications through modeling. The aim of this research is to find ways in order to present the user interface in a clear manner such that the balance between ease-of-use and ease of learning is achieved.

1 Introduction

Nowadays, the development of software applications is no longer bounded within the confines of people with programming skills. People are no longer limited to just being end-users of an application, but are encouraged to be the creators of their own applications as well. An example of this is the growth of the world wide web and how the creation of web pages are no longer restricted to people who have skills in writing HTML code and scripts. The introduction of WYSIWYG HTML Editors such as Microsoft Front Page and Google Page Editor has made this possible. By hiding the HTML code in the background and allowing components to be dragged and dropped on to a page makes it easy for novices to create their own web pages.

The same thing is happening now to the mobile industry. Mobile phone users are no longer limited to use pre-installed applications on their devices or buying ready-made mobile applications for their personal purposes. People now have the power to create their own applications given the right motivation, creativity, skills and tools. Mobile phone companies and organizations have now opened up their application programming interfaces (APIs) that would allow anyone to develop their own applications for their mobile devices. Examples of these are the Java Platform Micro Edition (Java ME) API[1] from Sun Microsystems, the Android API[2] from the Open Handset Alliance and the iPhone API from Apple[3]. However, even though many users may have ideas for novel applications for mobile phones, software development is simply too difficult for most people. It takes a large amount of skill and familiarity with how

[1] http://java.sun.com/javame/index.jsp
[2] http://code.google.com/android/
[3] http://developer.apple.com/iphone/

the framework is used before a person can create a decent amount of code for a simple application. Even setting up the programming environment is a complex task, let alone, trying to figure out how to use the APIs, compiling, running and deploying the application on the actual device. Other things that make developing applications for mobile devices more difficult as compared to desktop applications are factors such as device limitations (e.g. screen size, computing power, power consumption) [4], different operating systems for mobile devices, different data representation and additional device capabilities (e.g. Bluetooth, Wifi, GPS, Camera-enabled) which are not standard to all devices and therefore should be considered when developing a uniform application that can be run on different mobile devices.

In this research, we are investigating ways to make application development accessible to people with low or no programming skills. We propose applying model-driven development (MDD) which is an approach to creating complex software systems by first creating a high-level, platform-independent model of the system, and then generating specific code based on the model to the target platform [5]. In ordinary software development, models are just thought of as tools for getting system requirements and for documentation purposes, however, in MDD, the models are actually part of the implementation of the system. The basic idea of our work is to come up with a modeling environment that is specific for modeling mobile applications which targets non-experts as the main users. Non-expert users here are defined to be people who have little or no experience in programming for mobile platforms.

We want to present to the user one application that they could use to model their mobile applications without having to worry about low-level coding. In order to do this, we are developing a tool called Mobile Applications Modeler (Mobia). The focus of discussion for this paper is on the design of the user-interface for the Mobia modeling environment. The aim here is to find out which user interface design concepts are most suitable in order for non-expert users to develop their own mobile application with ease. The goal is to present the interface such that the balance between ease-of-use and ease-of-learning [10] is achieved. We have focused on non-expert users in this research and do not include expert users in general since these two types of users often differ in their experiences and needs [6]. Unlike existing modeling tools such as Magic Draw[4] and Eclipse with the Eclipse Modeling Framework (EMF)[5] plugins that are more general purpose modeling tools, we want to present to the user a more domain-specific modeling tool that specializes only on modeling mobile applications. The focus of this part of our research is on how to present the user interface of the Mobia modeler such that it is easy to use for non-expert users.

The rest of the paper is organized as follows: section 2 will discuss a few related researches to our work particularly in the area of model-driven development. In section 3 we will discuss the different user interfaces of existing development and modeling tools that are the basis of some of our designs. And then, the remainder of the paper will be a discussion of our approach such as, the design of our prototypes and evaluation results.

[4] http://www.magicdraw.com/

[5] http://www.eclipse.org/modeling

2 Related Work

Integrated development environments (IDEs) are tools which are made to ease the application development process. Most IDEs provide an environment that features a text editor, compiler, debugger and simulator to name a few, which are all integrated into one application. They have evolved throughout the years, adding more features (e.g. GUI designer, version control, etc.) that would help the developer in accomplishing their tasks in the most efficient way. For mobile applications development in particular, examples of IDEs that allows plugins for mobile application development are the Netbeans, Eclipse and XCode development environments. A problem with IDEs for mobile application development though is that, different mobile phones have different application programming interfaces and platforms. Thus, creating a common application that would run on different platforms of mobile phones tends to get tedious and redundant since developers have to develop different code for each of them. One solution to this problem is by applying the model-driven approach in which models are used to describe the application and through transformation tools, these models are transformed to code that would run on specific platforms [5].

An example research that applies MDD is the Multimedia Modeling Language (MML) which is a platform-independent language used for the model-driven development of multimedia applications [7]. MML models are transformed into Flash models which can then later on be loaded into the Flash authoring tool for further completion of the application [8]. The approach [7,8] is usually for teams wherein graphical designers and software designers need to work together in a certain project. Each group of users has their own expertise in terms of skills and tools they are using. However, when non-expert users are involved in the development process, this approach can be quite complicated. Extensive knowledge on how to make UML models is necessary in order to create the applications using this approach, and since the tools are not yet integrated, mastery in using these tools is a must [8]. Dunkel and Bruns [3] also presents a model-driven way of producing business applications for mobile devices with BAMOS (Base Architecture for Mobile Applications in Spontaneous networks) as the target platform. Their models are expressed in UML activity diagrams to specify control flow and the description of mobile services through a DSL they have defined using UML profiles. As with MML, the approach uses different tools which are not yet integrated [3]. Another research that applies the MDD process and targets non-experts as the primary developers is the Simple Mobile Services (SMS) project. This project aims to create service authoring tools and mobile services that are *simple to use, find and set up*[9]. They focus on non-expert users as the people assembling these mobile services on their own. SMS applies the MDA [5] approach in building their services [2]. Our approach is similar to SMS in a way that we target non-expert users as main users of our tool for developing mobile applications. However, while SMS focuses on mobile web-based services, our research focuses on mobile-based applications.

In the next section, we will discuss the different user-interface components present in various development and modeling environments. We want to find out which existing approaches in the UI and some new ones are most suitable for non-experts.

3 A Closer Look into the User Interfaces

In this section, we would like to compare user-interfaces of existing IDEs that supports mobile application development (Netbeans and Eclipse), and a modeling tool (MetaEdit+) that supports domain specific modeling of mobile applications. We want to explore what features these tools have, and which of these features are essential parts of an environment in which non-experts can benefit in it.

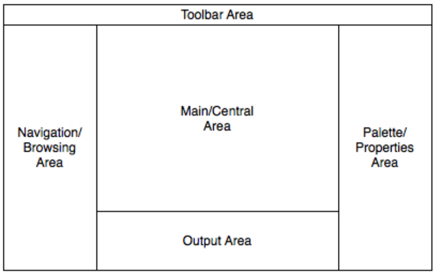


Fig. 1. General Parts of a Development/Modeling Environment

In studying these tools, we have identified four basic areas that are usually present in such environments. For the purpose of discussion in this paper, we will attach a general name to each of the areas, which may be identical or not to how they are labeled in such environments. Fig. 1. shows the typical default location of the main areas and their names. Table 1 contains the main areas and some of the possible contents that may appear in those areas.

Table 1. The different areas and their possible contents

Area	Possible Contents
Navigation/Browsing Area	Different components in a certain development project (e.g. files and folders, classes and packages)
Main/Central Area	The component in which the user is currently working on (e.g. source code, design for a user interface, data source)
Palette/Properties Area	Components that can be dragged and dropped to the main/central area (e.g. UI components, Datasets)
Toolbar Area	Button controls (e.g run, debug), editing controls (e.g. copy, paste)
Output Area	Program output, Compiler errors, Debugging messages, etc.

Shown in Fig. 2. is an overview of the Netbeans 6.5 environment. The components described in our general UI model for a development environment are present in the Netbeans environment. One additional feature of Netbeans is the ability to switch to different views in the main area depending on what the user is focusing on. The *source view* allows the user to make changes to the source code; the *screen view* allows drag and drop design of the mobile application's user interface; the *flow view* allows adding logic to the program by dragging flow arrows between the different screens; and the *analyzer view* shows unused resources and MIDP compliancy. Switching through the different views changes the contents of the palette area, depending on what components are needed in that certain view. Netbeans also has the ability to bind a screen component's data to information taken from a database.

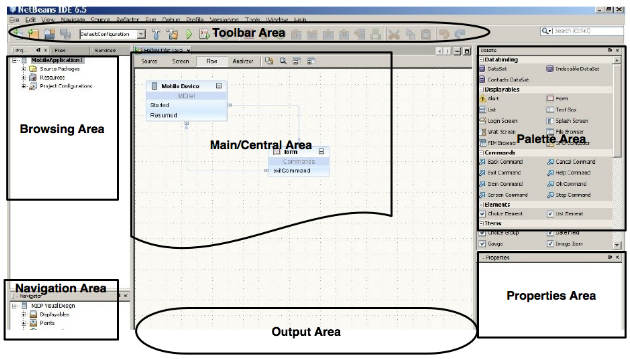
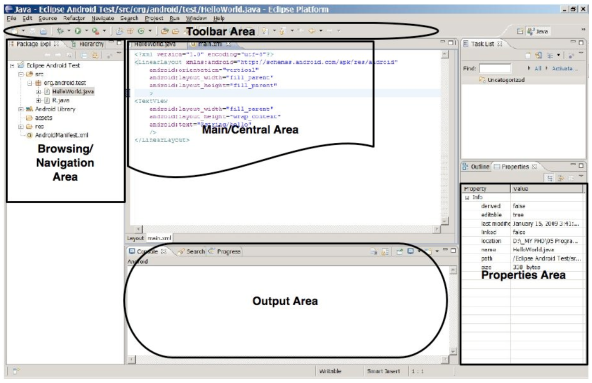


Fig. 2. Overview of the Netbeans Environment

The next IDE interface that we are going to discuss is the Eclipse IDE. There are several projects that aim to develop plugins for Eclipse to allow mobile applications development (e.g. EclipseME[6], Eclipse Plugin for Android[7]). For the purpose of this paper, we will focus on analyzing the interface for the Eclipse IDE used for developing Android applications since the basic components of the IDE are similar anyway. As shown in Fig. 3, the positioning of the components in the environment are similar to that of Netbeans. However, the features offered by the Eclipse environment are just a subset of Netbeans. As of the moment this paper is written, it does not feature a drag-and-drop GUI environment for developing Android applications, but through editing an XML file for the placement of the GUI components on the screen, or by directly adding lines to the source code for the GUI. Although, as expected in the future, developers might add more features for easy GUI development as the platform matures. DroidDraw[8] is one example application UI editor for the Android platform which generates an XML file that can be copied to the main code.

The MetaEdit+ Modeler is a DSM tool that allows the modeling of different domain specific applications (e.g. Mobile, Automotive, Telecom, Embedded, etc.). One supported domain is for modeling smartphone applications. Fig. 4 shows an overview of the basic user-interface of MetaEdit+ for modeling mobile applications. Unlike the first two IDEs we have described, the MetaEdit modeler features a simpler interface with several of its components positioned at different areas. The palette area contains fewer number of components as compared to the first two tools discussed. It features specialized constructs which is specific for such mobile platforms. The navigation area contains a list of components in the model. Below the navigation area is the properties area, which contains information about the current component in focus.

[6] http://eclipseme.org
[7] http://code.google.com/android/intro/installing.html
[8] http://www.droiddraw.org/



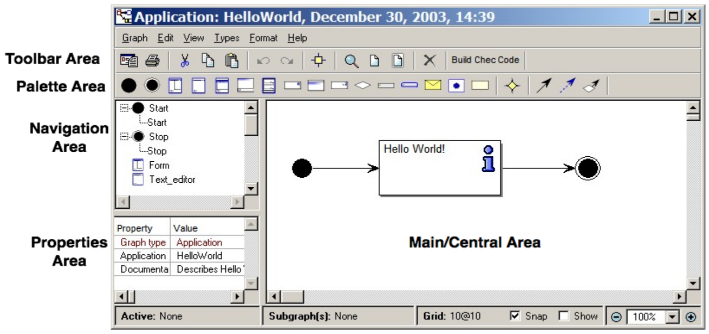
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

Property	Value
Info	
derived	false
editable	true
last modifie	January 15, 2009 3:41:...
linked	false
location	D:_MY PHD\05 Progra...
name	HelloWorld.java
path	/Eclipse Android Test/sr...
size	338 bytes

Fig. 3. Overview of the Eclipse Environment

From these three examples, we want to extract the most desirable feature of each that can be applied to the design of Mobia. The Netbeans environment for instance, features the ability to change to different views, which can allow the user to concentrate on one task at a time. However, it contains so much features that it can take awhile before the user can actually take advantage of such features. The MetaEdit tool on the other hand contains only a limited number of components. It contains specialized constructs that could easily be identified by the user. All the tasks such as designing the screen and adding flow to the program is modeled in one view. The disadvantage about this though, is that since it is very specialized, the user is restricted to the type of application they can create. The Eclipse environment also offers a very simple interface and not shows too much features. It is clearly a tool for expert developers, who basically know what source code to type in for the applications they are developing.

In the next section, we will discuss more about our approach in finding the ideal user-interface for the Mobia Modeler such that non-expert users will be able to use it. We apply some design patterns that is shown in the previous tools that we have described, and try to evaluate it in order to find out which features are most desirable for such an environment.



Property	Value
Graph type	Application
Application	HelloWorld
Documenta	Describes Hello

Fig. 4. Overview of the MetaEdit+ Modeler

4 The Mobia Modeler User Interface

The Mobia Modeler is a modeling tool specifically designed to allow modeling for mobile applications. The target users for Mobia are non-expert users who are people that have little or no experience at all in programming for mobile platforms. For this particular study, we feature a module of Mobia that is focused on modeling applications in the domain of mobile health monitoring. As of the moment, we want to focus on one type of domain, since different domains may offer different modeling constructs. For this module, the users will model a certain type of application that can be used for health monitoring and feature modeling constructs that represent data from different medical gadgets or medgets (e.g. ECG meter, Thermometer, etc.). In order to find the ideal interface for Mobia, we have created two prototypes using Flash, which offers two different UI designs. Just to clarify, these prototypes are simply focused on evaluating the different user interface designs and interactions and do not yet have code transformation features.

4.1 Mobia with One View

Shown in Fig. 5 is a screenshot of the first version of our Mobia prototype which we called Mobia One View. The first prototype offers one view for the user which means that, the user can design an instance of the mobile screens, add data and application control flow all in one view. The user can concentrate on designing a single screen by zooming into that area, and try to see an overview of the whole system by zooming out. The palette on the right side of the screen contains screen components that can be dragged on to the mobile screen. For our prototypes, we only feature a subset of the possible screen components that a mobile application can have. The right palette also contains data input which we call medget (short for medical gadget) input. The medget constructs in the medget input palette contains abstract representations of information that comes from health monitoring devices capable of sending their data to a mobile device. More information about the different representations of medget data will not be discussed in this paper, but in a separate paper [1].

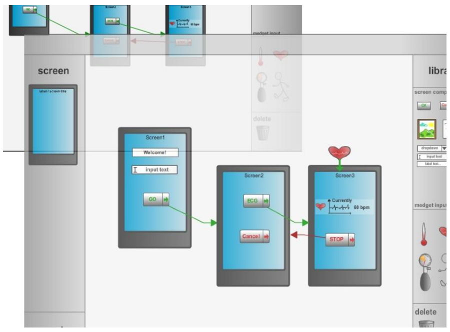


Fig. 5. Mobia with One View. (In the foreground) The main area is zoomed-in to see the screen designs better.

4.2 Mobia with Multiple Views

The second design approach that we did for Mobia is what we called multiple views which is shown in Fig. 6. This is similar to the Netbeans IDE in which the main area features different kinds of views depending on the specific task that the user is doing. The reason behind the choice of this type of design is that we want the user to focus on one task at a time.

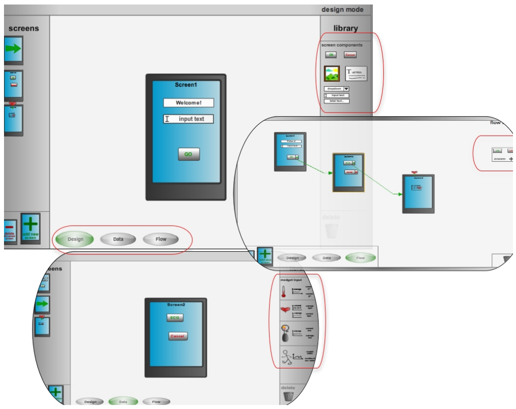


Fig. 6. Mobia with Multiple Views (Design, Data and Navigation View)

The default view is the *Design View* in which the user can design individual screens by dragging and dropping screen components from the palette to the screen. The left panel contains all the mobile screens for that application. Clicking on an individual screen in the left panel shows it in the main view to be further edited and designed. Screens can be added and deleted by pressing the add and delete buttons respectively. This design is borrowed from presentation programs such as Microsoft Powerpoint and OpenOffice Impress wherein each slide can be viewed from a panel and allows switching from one slide to another by clicking on the mini versions of the slides in the panel. The *Data View* is similar to the design view except for the fact that the palette contents on the right panel changes to medget data. In this view, the users can concentrate on how they want data taken from health monitoring devices be displayed on the screen. These medget components act as placeholders into which the real information from the devices will appear in the real application. The last view is the *Flow View*, which shows all the screens in the model and how the screen transitions from one to the next. The user can add basic control logic to the application by dragging on arrows and linking the screens together. In this view, a small component palette contains buttons in which the users can drag to the screens. The logic behind this design approach is that, in the application, only by pressing a control component such as a button can trigger going from one screen to the next.

5 User Study Evaluation and Results

Given the prototypes that we have described in the previous section, we want to find out which of the prototypes provides a simpler UI for the user and gets the task done quickly. For a more subjective evaluation, we also want to find out which design is more fun and easier to use. In order to do this, we have conducted a user study in which each user is given a task to accomplish using both prototypes. In order to measure efficiency, we get the time in which the user accomplishes a certain task. In order to eliminate the bias towards the second prototype, we alternate which prototype each participant uses first in doing the tasks. The participants were instructed not to ask any questions from the evaluators. The goal here is to allow the participants to explore the tool themselves and learn how to use it by themselves without any outside intervention. At the beginning of the user-study, the participant was asked to explore the prototypes and give comments. After they are done studying the tool in whichever method they choose, they are given two tasks which are to design the contents of the screens and then later on to add control flow to the screens. They were asked to create 3 screens with some screen components in them. After designing the screens, they were asked to add control flow in which allows switching to a different screen whenever a button is pressed.

Table 2. The average times accomplishing the tasks using the two prototypes

Version	**Average Time (Minutes)**	
	Screen Design Task	*Adding Control Flow Task*
Mobia One View	4.036 minutes	1.126 minutes
Mobia Multiple Views	5.833 minutes	2.223 minutes

There were 10 participants to our user study: 60% have backgrounds in the field of Computer Science and the rest from the fields of Educational Psychology, Archeology, Architecture and Social Welfare. Only 10% of the participants have a background in programming for mobile platforms which is also in the very basic level.

Table 2 shows the average times of when the users accomplished the tasks while, Table 3 shows the results for the subjective evaluation in terms of which prototype is easier and more fun to use. Based on the results shown in the tables, Mobia One View allows the users to do the tasks faster as compared to the multiple view. A factor that might contribute to this is the fact that in multiple views, the user has to switch from one to the next in order to add a different component or do another design task. Based on the subjective feedback of the participants, the Mobile one view poses an environment that is both easy and fun to use.

Table 3. Subjective evaluation for the Mobia Prototypes

Version	**Criteria (Percentage of Users)**	
	Easier to Use	*More fun to Use*
Mobia One View	60%	50%
Mobia Multiple Views	40%	40%
None	0%	10%

6 Summary and Future Work

In this paper, we have presented different design ideas for a mobile application modeling environment that targets non-experts as the main users. The design and results presented here are just the initial phase of our iterative approach to finding the ideal interface for a tool that would help accomplish tasks with ease.

Our future work aside from continuing to polish the user interface design of Mobia, is to come up with an underlying framework to support code transformation from the models. An approach to have a user-adaptive tool that changes according to each user's existing skills and preferences to enhance user experience and learning is also envisioned.

Acknowledgments

We would like to thank the German Academic Exchange Service (DAAD) for funding this research. We would also like to thank Ugur Örgün for helping with the prototypes and to the people who participated in our user study.

References

1. Balagtas-Fernandez, F., Hussmann, H.: Modeling Information From Wearable Sensors. In: MDDAUI 2009- Model Driven Development of Advanced User Interfaces 2009. CEUR Proceedings (2009)
2. Bartolomeo, G., Casalicchio, E., Salsano, S., Melazzi, N.B.: Design and Development Tools for Next Generation Mobile Services. In: International Conference on Software Engineering Advances, ICSEA 2007, p. 16 (2007)
3. Dunkel, J., Bruns, R.: Model-Driven Architecture for Mobile Applications. Business Information Systems, pp. 464–477 (2007)
4. Gaedke, M., Beigl, M., Gellersen, H.-W., Segor, C.: Web Content Delivery to Heterogeneous Mobile Platforms. In: ER 1998: Proceedings of the Workshops on Data Warehousing and Data Mining, pp. 205–217. Springer, London (1998)
5. Kleppe, A., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Pearson Education, Inc., Boston (2003)
6. Petre, M.: Why looking isn't always seeing: readership skills and graphical programming. Commun. ACM 38, 33–44 (1995)
7. Pleuss, A.: MML: A Language for Modeling Interactive Multimedia Applications. In: ISM 2005: Proceedings of the Seventh IEEE International Symposium on Multimedia, pp. 465–473. IEEE Computer Society, Washington, DC, USA (2005)
8. Pleuß, A., Vitzthum, A., Hussmann, H.: Integrating heterogeneous tools into model-centric development of interactive applications. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 241–255. Springer, Heidelberg (2007)
9. The SMS Project, `http://www.ist-sms.org`
10. Weiss, S.: Handheld Usability. John Wiley and Sons, Chichester (2002)