Navigation Flow Modeling as a Basis for the Automatic Generation of Android APPs

HSI-MIN CHEN¹, TSUNG-CI LIN¹, LIEN-WU CHEN¹, BAO-AN NGUYEN¹, AND YI-CHUNG CHEN²

¹Department of Information Engineering and Computer Science
Feng Chia University
Taichung, Taiwan
²Department of Industrial Engineering and Management
National Yunlin University of Science and Technology
Yunlin, Taiwan
E-mail: {hsiminc; M9802118; lwuchen}@mail.fcu.edu.tw; baoanth@gmail.com; chenyich@yuntech.edu.tw

According to the marketing survey in 2017, Android has gained the world's highest market share of smart phone operating systems. However, with the increase of APP functionalities requested by smartphone users, program logics, data structures and operation flows of APPs become more and more complicated. If there are no appropriate methods to facilitating complex App developments, it will result in that these APPs are not able to deliver on schedule, difficult to maintain and with low software quality. Therefore, this research proposes a development method for Android APPs from the viewpoint of navigation flows. A navigation flow is referred to as a series of screens switched in order to perform a function of a mobile application. We leverage navigation flows to address the issues regarding the pressure of time to market and the mismatch between design specifications and products. Thus, we came up with an APP development method based on navigation flows and MDA technologies. The proposed method can automatically transform navigation flows specified in wireframes to Android project code, meanwhile taking Android project structure into account. The productivity and maintainability of Android APPs can be improved by the proposed method.

Keywords: wireless sensor networks, localization, mobile beacon, mobile anchor, RSSI

1. INTRODUCTION

IDC Research, a world-renowned market research organization, reported a survey that more than 344 million smart phones sold in the first quarter of 2017 [1]. Among these smart phones, Android [2] operating system accounts for 85% of the global smart phone market, while Apple’s iOS [3] operating system accounts for 14.7%. Through this survey, it is obvious to see that the choice of Android operating system is still a big one in recent years.

However, with the increase of APP functionalities requested by smartphone users, program logics, data structures and operation flows of APPs become more and more complicated. If there are no appropriate methods to facilitating complex APP developments, it will result in that these APPs are not able to deliver on schedule, difficult to maintain and with low software quality. In general, today’s APP developments encounter
the following challenges:

- **Tight development schedule:** According to Nine Hertz's survey from 100 iOS, Android, and Web engineers [4], "18 weeks" is the average time for an APP to go from development to launch. In addition, the number of APPs on Google Play is more than 3 million. In a such short development schedule, how to rapidly implement user requirements on APPs has become an important factor to enhance product competitiveness. Without good software development methods, APP development productivity would not be improved effectively, thereby losing the competition of leading the market.

- **Inconsistency between APP design and implementation:** Version releases of APPs for the changes of user's requirement is more frequent than the ones of traditional desktop applications. Under such situation where requirement changes and program updates of APPs are proceeded constantly, if we can't manage the relationship between intended design and its implementation effectively, the gap between them will become more and more widened. In the end, APP products are hard to maintain and with low quality.

   To address the above challenges, we first studied APP development processes adopted by APP companies. We found that there is a common activity involved in these developments, which is drawing wireframes [5]. In order to capture and describe user experience (UX), APP designers are used to leveraging wireframe technique to clarify the requirements of user interfaces and get a consensus of user experience with product stakeholders. Wireframes can be drawn by hand with paper and pencil, or by tools. The main purpose of wireframes is to allow stakeholders to get agreements through easy-to-understand images. In addition to the UI components and layouts that constitute each screen of an APP, an APP wireframe also specifies screen navigation flows (SNF) from one screen to another, which show that the user can perform one of the functions provided by the APP after navigating a series of screens.

   In this research, we propose an Android APP software development method centered on "navigation flow". Because APP designers can specify navigation flows in the design model at the early stage of APP developments, we can extract the flows from the design model. Then, Model-Driven Architecture (MDA) [6,7] techniques are applied to generate the corresponding code skeleton automatically based on the models of given navigation flows. The proposed APP development method can greatly reduce the time spent by APP developers in writing code of APPs, thus solving the problem of tight development schedule. On the other hand, the corresponding Android code can be generated directly from the wireframe design model, which can also explain the issue that APP implementations do not conform to their design. This automatic APP generation method can maintain consistency between APP design and implementation. When new functions are added, or defects are fixed, in the future, APP modifications can start from design model, through which APP maintainability can be achieved.

   The remainder of this paper is organized as follows. Section 2 reviews related literature. Section 3 presents proposed App development approach. Conclusions are drawn in Section 4.
2. RELATED WORK

2.1 Android APPs Modeling

The literature reports several approaches regarding the modeling of Android Apps. Modeling Android Apps can come from various perspectives, such as static structures and dynamic behaviors. To model Android Apps, UML (Unified Modeling Language) [8] is the widely used tool that provides a set of diagrams to capture the concepts of systems’ structure, behavior and interaction. Ko et. al. [9] extended the notations and syntax of standard UML class diagram by defining meta-models for static structure elements, dynamic element lifecycle and user interface components. Through these extensions of meta-models, App designers can specify Android Apps more precisely. Similarly, Parada et al. [10] employed UML class diagram to describe structure view of Apps and used sequence view to describe behavior view. App designers can add application-specific classes to system pre-defined classes to present static relationships among these classes and model dynamic interactions in a class method via sequence diagrams.

In addition to UML modeling techniques, a variety of Android App modeling approaches, adopting self-defined modeling languages, were presented as well. Yang et. al. [11] defined WTG (Window Transition Graphs) to describe series of GUI windows, relevant events and callback functions. The authors adopted WTG to model the stack of windows, the changes of window stack and the callbacks for these changes. The modeling results can serve as inputs to examine the static analysis of Android APPs. Most of APP modeling approaches used graphic representation to describe static structures and dynamic behaviors. Compared to text-based design specifications, modeling by graphic representations can help relevant stakeholders understand entire App designs from various perspectives and enable communications with ease.

2.2 MDA-based development approaches for Android Apps

In the last decade, MDA technologies have been applied in software developments widely. In general, MDA is leveraged to generated source code from concept models by performing several transformation processes. The highest abstract model is referred as to PIM (Platform Independent Model) which is a system model independent of specific platforms/languages. Then, PIM models are transformed to PSM (Platform Specific Model) models by applying defined transformation rules. Finally, the PSM models are further transformed to source code or text documents. With the help of MDA, the goal of automatic code generations can be achieved.

Lachgar et al. [12] came up with a Technology Neutral Domain-specific language as PIM used to model UI components of Android APPs. After that, transform the PIM models to various PSM models designed for the platforms of WinPhone, iOS and Web. Besides, Parada et al. [10] adopted UML class diagram to describe structure views of Apps and employed sequence diagram to present behavior views of Apps. Subsequently, the authors used GenCode [13] to generate Android-related code.

Compared to the above studies, our proposed method takes wireframes as source models that can obtain from UI/UX designers directly at early design phase without spending extra effort to learn and draw various models.
3. THE PROPOSED APPROACH

In this section, we outline the proposed method to generating Android project code from the designs of wireframes. Figure 1 depicts the steps involved in the code generation process. We will elaborate the detail of each step as follows.

![Diagram](image.png)

Figure 1. The MDA-based code generation process

![Diagram](image.png)

Figure 2. An example of wireframe design
3.1 Model Screen Navigation Flow

Different from previous studies, we first extract screen navigation flows from wireframes that specify both UI screens and navigations between these screens of Apps. Figure 2 shows an example of a wireframe design for the TaxiBar APP [14]. Wireframes can be drawn by hand with a pen and paper, or by tools. The main purpose is to allow participants to reach consensus through easy-to-understand images. An APP wireframe can describe navigation flows from one screen to another in addition to UI components and layouts that make up each screen of the APP. It is used to explain that a user can perform functions provided by an APP after following the operations of navigation flows. Taking Figure 3 as an example, the APP designer first designed the application login screen. Besides describing the UI components and screen layouts, this screen will also pull a navigation flow from the login button (blue arrow line) to the next APP main screen. It indicates that when a user clicks the login button, if the user’s authentication is successful, the APP will navigate from the login screen to the APP’s main screen. After APP designers confirm the design specifications for each screen of an APP, the design specifications can be submitted to subsequent developers in charge of the development of the APP based on the design specifications.

As a sequence of screen switches forms a screen navigation flow through which users can perform a function provided by an App, we use UML State Machine Diagram [15] to formally model these screens and the switches between them. UML State Machine Diagram is composed by three elements, i.e., states, events and transitions. States describe the possible ones presented in a system. When a certain event is fired, one of system states will transit to another state. In general, State Machine Diagram is helpful to model dynamic behaviors of systems.

Table 1. The semantic mappings between State Machine and APP wireframe

<table>
<thead>
<tr>
<th>Notation</th>
<th>Semantic in State Machine</th>
<th>Semantic in APP wireframe</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Initial State" /></td>
<td>Initial State</td>
<td>Enter an App</td>
</tr>
<tr>
<td><img src="image" alt="Final State" /></td>
<td>Final State</td>
<td>Exit an APP</td>
</tr>
<tr>
<td><img src="image" alt="Simple State" /></td>
<td>Simple State</td>
<td>An APP Screen</td>
</tr>
<tr>
<td><img src="image" alt="Transition" /></td>
<td>Transition</td>
<td>Switch between two screens</td>
</tr>
</tbody>
</table>

To model screen navigation flows by State Machine Diagram, we define the mappings of notation and semantic between these two concepts in Table 1. App developers can transform screen navigation flow designed in wireframes to formal State Machine diagrams by using these mapping rules. When App designers model screen navigation flows of an App, they can use the notations of Initial and Final States to describe the start and end of an App. Each screen presented in wireframes can be mapped as a Simple State modeled in State Machine diagrams. Based on the behavior depicted in wireframes,
when users perform an operation on the UI components of an App, e.g. tapping a button, the current screen will be switched to another one. This switch behavior can be modeled by a transition that associates two Simple States, i.e., screens in wireframes, with an arrow pointing to the next one. Besides, App designers can attach information, including events, conditions and actions, to a transition. Figure 3 explains the formal State Machine model derived from the wireframe shown in Figure 1.

3.1 Extend XMI Schema

As UML is a standard modeling language defined by OMG (Object Management Group), many software vendors have provided several UML modeling tools to help designers design software systems from various perspectives by different diagrams. However, in the past, these tools were not compatible with each other, as the design results generated by these tools cannot be exchanged. To enable the compatibility, OMG defined XMI (XML Metadata Interchange) [16] that can facilitate the exchange of the UML diagrams made by different UML modeling diagrams.

In addition to enable compatibility among modeling tools, XMI has an extensible feature that allows users to define supplementary information and structures to existing meta-models for different purposes. For example, modeling tools can add coordinate information of graphic UI components displayed on editors to existing standard meta-models and saved them in XMI files. Based on this extensible feature of XMI, we add the information required by generating App to the standard meta-model of State Machine Diagram.

To navigate APP screens, users can trigger an event on a UI component to switch to another screen. For example, after users tap a “login” button, the current login screen is navigated to the main screen if the users are authorized. However, in the specification of standard State Machine Diagram, we cannot model UI components in diagrams directly. To address this issue, we exploit XMI to extend the meta-model of State Machine Diagram to which the information of required UI components is supplemented. List 1 depicts an example of XMI extension for State Machine Diagram.
List 1. Example of XML Extension for Statement Machine Diagram

```xml
<transition xmi:id="trans_1" source="Login" target="Main">
  <eAnnotations xmi:id="ui_annotati
  on_1" source="UI Component">
    <details xmi:id="ui_btn_login" key="Type" value="Button"/>
    <details xmi:id="ui_btn_login_name" key="VariableName" value="loginBtn"/>
  </eAnnotations>
  <effect xmi:type="uml:FunctionBehavior" xmi:id="act_1" name="doTransition"/>
  <trigger xmi:id="trg_1" name="onClick" event="onClick_1"/>
</transition>
```

3.3 Design Model-to-Model Transformation Rules

After modeling screen navigation flows from wireframes with the formal modeling language, i.e., State Machine Diagram, the next step is to transform the models of State Machine Diagram to the ones of Android. At this step, we do not transform the models directly into Android project code. The reason is that, with compared to the upgrade frequency of Android programming syntax and APIs, the structures of Android programs are updated less often. Thus, at this step, we transform the State Machine models to the models, i.e. PSM (Platform Specific Model) models, defined specific to the Android platform. There are two actions, defining the meta-model of Android models and figuring out transformation rules, involved in this step. Android meta-model is used to describe the structure of Android models.

![Android meta-model](image)

As shown in Figure 4, we exploit reverse engineering mechanism [17] to obtain the Android meta-model, which is related to screen navigation flows and required to generate Android project code, from the code of Android APIs. The defined Android meta-model is described by means of UML Class Diagram. In the Android meta-model, an activity is composed by a set of views that can be UI widgets or layout components. For the UI widgets, for instance, Button, ListView, ImageView, TextView, etc., which extend View, they can register listeners to catch events triggered by users and then decide
whether to perform navigations. Besides, the Intent class describes the source and destination activities for the navigation event triggered.

Table 2. Model-to-Model transformation rules

<table>
<thead>
<tr>
<th>Rule #</th>
<th>XMI tag in State Machine Model</th>
<th>Attribute</th>
<th>XMI tag in Android Model</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>state</td>
<td>-</td>
<td>Activity</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>UI Component</td>
<td>Type</td>
<td>View</td>
<td>type</td>
</tr>
<tr>
<td>3</td>
<td>UI Component</td>
<td>VariableName</td>
<td>View</td>
<td>variableName</td>
</tr>
<tr>
<td>4</td>
<td>trigger</td>
<td>-</td>
<td>Listener</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>transition</td>
<td>-</td>
<td>Intent</td>
<td>-</td>
</tr>
</tbody>
</table>

After defining the Android meta-model, we come up with a set of Model-to-Model transformation rules, as depicted in Table 2, that can convert State Machine models to Android models. To implement these transformation rules, ATL (ATLAS Transformation Language) [18] is exploited in this research. List 2 shows an example that specifies the rule to transform the elements, including state, UI component, trigger and transitions, defined in State Machine meta-model to the corresponding ones in Android meta-model.

List 2. Example of an ATL transform rule

```
rule Element {
  from
    s :StateMachine!State
  using{
    uiType : Sequence(String) = s.getUIType();
    uiVariable : Sequence(String) = s.getUIVariable();
    uiListener : Sequence(String) = s.getUILocator();
    uiTargetName : Sequence(String) = s.getTargetName();
  }
  to
    ToActivity : AndroidModel!Activity(
      name <- s.name,
      View <- ToUi
    ),
    ToUi : distinct AndroidModel!UI_Component foreach(e in uiType)(
      type <- e,
      variableName <- uiVariable,
      Listener <- ToListener
    ),
    ToListener : distinct AndroidModel!Listener foreach(e in uiListener)(
      listenerName <- e,
      Intent <- ToIntent
    ),
    ToIntent : distinct AndroidModel!Intent foreach(e in uiTargetName)(
      targetName <- e,
      sourceName <- s.name
    )
}
Followed by applying the transformation rules to State machine models, the corresponding Android models are generated in XMI format. List 3 shows an example of the Android model derived from the State Machine model described in List 1 after applying the Model-to-Model transformation rules.

### List 3. Example of Android model derived from List 1

```xml
<Activity name="Login">
  <View type="Button" variableName="loginBtn">
    <Listener listenerName="onClick">
      <Intent targetName="Main" sourceName="Login"/>
    </Listener>
  </View>
</Activity>
```

### 3.3 Design Model-to-Text Transformation Rules

Subsequently, the generated Android models are further transformed to source code that is specific to Android platform. At this step, we developed a set of Model-to-Text transformation rules used to convert Android models to executable Android project code.

#### Table 3. Model-to-Text transformation rules.

<table>
<thead>
<tr>
<th>Rule #</th>
<th>XMI tag in State Android Model</th>
<th>Attribute</th>
<th>Android Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Activity</td>
<td>-</td>
<td>Activity Name</td>
</tr>
<tr>
<td>2</td>
<td>View</td>
<td>Type</td>
<td>UI Type</td>
</tr>
<tr>
<td>3</td>
<td>View</td>
<td>VariableName</td>
<td>uiComponent</td>
</tr>
<tr>
<td>4</td>
<td>Listener</td>
<td>-</td>
<td>Listener</td>
</tr>
<tr>
<td>5</td>
<td>Intent</td>
<td>-</td>
<td>Intent, Source Activity, Target Activity</td>
</tr>
</tbody>
</table>

Different from the previous step where the transformations between models are executed by a script-based rule language, we leveraged code templates to specify Model-to-Text Transformations at this step. Model-to-Text Transformation rules are embedded in code templates. To implement Model-to-Text Transformation rules in code templates, we exploit Acceleo model transformation tool [19] to write rules in templates and then obtain executable Android project code from Android models. List 4 shows an example of Acceleo code template, and List 5 shows an example of the Android Java code generated from List 2.

#### List 4. Example of Acceleo code template

```plaintext
[comment encoding = UTF-8 /]
[module generateActivity(http://org/model/AndroidModel')]
[template public generateActivity(anActivity : Activity,aApplication : Application)]
[file (anActivity.name.concat('.java'), false, 'UTF-8')] package [aApplication.packageName;]/
import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
```
List 5. The generated Android Activity code

```java
package com.example.myapplication;

import android.app.Activity;
import android.os.Bundle;
import android.content.Intent;
import android.widget.*;
import android.view.View;

public class Login extends Activity {
    Button loginBtn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.Loginactivity);
    }
}
```

loginBtn=(Button)findViewById(R.id.loginBtn);
loginBtn.setOnClickListener(new Button.OnClickListener(){
  public void onClick(View v) {
    Intent intent = new Intent();
    intent.setClass(Login.this, Main.class);
    startActivity(intent);
  }
});

Figure 5. Android project templates

To generate Android project code that can be imported to the Android IDE tool, i.e. Android Studio [15], other project-related artifacts are also required in addition to Android Java code. Thus, for these required artifacts, we developed the templates, as shown in Figure 5, to generate corresponding project code. By applying these templates, models of screen navigation flows can be transformed into Android project code and required artifacts. The generated Android project code can be directly built and deployed to Android smart phones without any manual intervention.

4. EXPERIMENTS

In this section, we describe the experimental results of our system. Firstly, we designed a synthetic experiment in which the proposed system was used to generate code from 25 input state machine models which vary in number of states and transitions. Secondly, we conducted a manual experiment in which 15 developers were assigned to implement an Android project including 5 screens and 4 transitions. Time to complete the two experiments was captured and compared in order to measure the effectively of our method. The experiments are discussed in detail as follows:

4.1. Synthetic experiment for code generation
In this experiment, we used multiple input UML models with variation on number of states and transitions. Each input model has two parameters: number of layers \((LN)\), and number of transitions per each state \((STN)\). Let considering the input state machine model as a tree, \(LN\) denotes the depth of the tree, and \(STN\) denotes the breadth of each node, except the leaf-nodes. An example model with \(LN=2\) and \(STN=2\) is shown in Figure 6. The total number of states of the model is computed by:

\[
\text{Total number of states} = 1 + STN^1 + \ldots + STN^{LN}
\]

where 1 is the initial state, \(STN^i\) is the number of state at layer \(i\th\) layer.

![Figure 6. An example testing state machine model with LN=2 and STN=2](image)

To conduct this experiment, we designed multiple testing state machine models with depth and breadth vary from 1 to 5. Consequently, we have 25 test cases in which the simplest case has only 2 two states, and the most complex case includes 3906 states. The experiment was executed on a standalone computer equipped by an Intel I5-3470 CPU, 16GB of main memory, and a 7200 rpm HDD (1TB).

![Figure 7. Code generating time for synthetic testing models](image)
The experimental results in Figure 7 show that the execution time was less than 2 seconds in almost test cases; the most complex case was completed in only 12 seconds, which is much faster than manual coding.

4.2. Manual coding experiment

In order to measure the implementation time of a simple project without using our method, we designed a simple Android project with 5 states and 4 transitions, as illustrated in Figure 8. A team of 15 coders was assigned to program the project, implementation time of each coder was carefully recorded (Figure 9). The average time to complete the task was 1021 seconds (standard deviation = 299.8 seconds). The data shown in Figure 7 and Figure 9 reveals that our system, in average, can save 15 minutes for a simple task within 5 states, or 3 minutes per state. By comparing time consumed in two experiments, we can show that the proposed MDA method have significant effects on shortening APP development time and increasing development efficiency.
5. CONCLUSIONS

This research proposed an MDA-base App development method that takes screen navigation flows as input and automatically generates Android project code with required artifacts. By means of this method, APP development time can be reduced. Besides, it can ensure that the behavior of generated Android project code can conform to the original designs, i.e. the model of screen navigation flows. Thus, with the help of the proposed method, APP developers can rapidly obtain the skeletons of APPs, including UI components and the code exploited to enable screen navigations, by automatic code generation from wireframes. By contrast, its limitation is that APP developers must implement the logic part of each screen as it is not described in wireframes.

In the future, we will leverage the models of screens navigation flows as test cases to validate whether their corresponding implementation adhere to these modeled navigation flows. On the other hand, we will tailor the proposed App development approach to apply in other App platforms, such as iOS.

REFERENCES


Hsi-Min Chen (陳錫民) received the B.S. and Ph.D. degrees in Computer Science and Information Engineering from National Central University, Taiwan, in 2000 and 2010, respectively. He is currently an Associate Professor with the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. His research interests include software engineering, object-oriented technology, service computing, and distributed computing.

Tsung-Chi Lin (林宗祺) received the B.S. degree in the Department of Information Engineering and Computer Science, Feng Chia University, Taiwan. He is currently a master student in the same department at Feng Chia University. His research interests include software engineering, mobile application technology, and education technology.
Lien-Wu Chen (陳烈武) received the Ph.D. degree in computer science and information engineering from the National Chiao Tung University, Hsinchu, Taiwan, in December 2008. From 2012 to 2015, he was an Assistant Professor with the Department of Information Engineering and Computer Science, Feng Chia University. Since August 2015, he is currently an Associate Professor. His research interests include wireless communication and mobile computing, especially in the Internet of Vehicles, Internet of Things, and Internet of People.

Bao-An Nguyen (阮寶恩) received the M.S. degree in Information Engineering and Computer Science from Feng Chia University, Taiwan, in 2011. He is currently pursuing the Ph.D degree in the same department. His research interests include data mining, software engineering and education technology.

Yi-Chung Chen (陳奕中) received the B.S. and M.S. degrees in Electrical Engineering from National Cheng Kung University, Tainan, Taiwan, in 2007 and 2008, and the Ph.D. degrees in Department of Computer Science and Information Engineering from National Cheng Kung University, Tainan, Taiwan, in 2014. He is currently an Assistant Professor in the Department of Industrial Engineering and Management, National Yunlin University of Science and Technology. His research interests include spatio-temporal databases, recommendation systems, social network analyses, artificial intelligences, and techniques of Industry 4.0.