A New Method to Analyze Graphical User Interfaces of Android Applications

Hong Anh Le[1](✉) and Ninh Thuan Truong[2]

[1] Hanoi University of Mining and Geology,
18 Pho Vien, Bac Tu Liem, Hanoi, Vietnam
lehonganh@humg.edu.vn

[2] VNU - University of Engineering and Technology,
144 Xuan Thuy, Cau Giay, Hanoi, Vietnam
thuantn@vnu.edu.vn

Abstract. In recent years, the number of Android smartphones increase dramatically and new applications are added numerously in Google store. Android developers usually have to deal with the difficulties such as limited capacity battery, screen design, and limited resources. Among them, specifying graphical user interfaces (GUI) of an application is one of the most important issues. This paper presents a new method to analyze GUI specifications of an Android application. We employ Event-B formal method and its refinement mechanism to formalize the specifications and to check if the constraints are satisfied. A running example of a Note application is given to illustrate the proposed method in detail.

Keywords: GUI specification · Event-B · Android applications

1 Introduction

With the rapid development of hardware technologies, smartphones become more powerful and much cheaper than ten years ago. Smartphones provide many advanced utilities to users and are in hands of a lot of people around the world. In the smartphone market, the devices using Android operating system contribute to around 65% in 2016. Currently, Google play store, the biggest market, contains more than 1.5 billions applications.

In software development process, specifying GUI is one of the most important step. Specifically, GUI of an mobile application becomes more critical because of various screen resolutions and limited resources. The developers who are based on the collection of GUI requirements design the screens and activities of their components before they actually write the code of the application. The GUI requirement documents often consist of UI objects and their semantic executions. One problem arises is that GUI specifications come with informal representation. Software developers, therefore, might misunderstand or can not realize the defects of GUI specifications. The earlier these faults are detected, the smaller cost of development is.

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2017
P. Cong Vinh et al. (Eds.): ICCASA 2016, LNICST 193, pp. 111–120, 2017.
DOI: 10.1007/978-3-319-56357-2_12

Formal methods have been used for describing, validating, and verifying GUI specifications of software systems. Authors [10] proposed a formal language, Fruit (Functional Reactive User Interface Toolkit), to write concise specification of GUI programs. Some model-based approaches [4,12,13] using different kinds of existing methods, e.g. Petri Nets, Z, Spec#, to formally describe GUI requirements. Mobile developers design GUI with components provided in the rich Android GUI framework. These existing methods are not specified enough for analyzing Android GUI designs. Hence, formally checking the design to find defects is an emerging issue and is needed to be more investigated.

This paper present a new method which employs Event-B [2] and its refinement mechanism to model and verify Android GUI requirements. Event-B, which uses set theory as modeling notations and mathematical proofs, is suitable for system modeling. The Event-B refinement allows to develop the system from abstract level to more concrete and precise levels. Moreover, we can make use of many platforms supported for Event-B such as RODIN to specify and to prove desired properties either automatically or iteratively. After verification phase at design level, EventB2Java or EventB2C can be used to generate executable programs from Event-B models. The contributions of the paper are (1) presents a formalization of Android GUI requirements in Event-B notations, (2) proposes refinement-based approach of GUI modeling which is suitable for mobile software development, and (3) shows that desired properties can be verified in the RODIN and the resulted Event-B models can be translated into executable implementations.

The rest of the paper is organized as follows. Section 2 provides a brief introduction of GUI design, Event-B, and the RODIN platform. The main work of the paper is presented in Sect. 3. In the next section, we apply the proposed method to analyze a Note application. Section 5 outlines some related work. Finally, Sect. 6 concludes the paper and gives some research directions.

2 Background

In this section, we outline some principles of GUI design in software system as well as in Android smartphone. After that, Event-B and its support tool RODIN is sketched.

2.1 GUI Design

The user interface is one of the most important parts of a software system. It defines how users interact with the system, e.g. using hands, keyboards, and voice, and how the system delivers the results to users, e.g. via screens and voice. Designing GUI is a subset of Human-Computer interaction studies.

GUI of Android applications contains a collection of graphical elements, e.g., Layout, Text, Button, etc., provided by the framework. Android framework provides several ways to intercept events from users' interaction with View or Activity class. An Activity provides a window screen for users to interact with the application. An application usually contains multiple activities in which a main activity is defined for the first appeared screen when lunching the application. An activity has four essential states such as *running* (the activity is in the foreground), *paused* (it has lost focus but still visible), *stopped* (the activity retains all its data but is invisible), and *destroyed* (the activity is killed from the memory).

2.2 Event-B

Event-B is a kind of formal method which combines mathematical techniques from the set theory and the first order logic. It is an evolution of B-method. Event-B is suitable for modeling large and reactive systems. The basic structure of an Event B model consists of a MACHINE and a CONTEXT.

Contexts form the static part of the model while machines form the dynamic part. Contexts can extend (or be extended by) other context and are referred (seen) by machines. Being considered as the static part of the model, the context is used to store, for instance, the types and constants used during the development of the system. The machine contains the dynamic part of the model. It describes the system state, the operations to interact with the environment together with the properties, conditions and constraints on the model. A Machine is defined by a set of clauses which is able to refine another Machine. We briefly introduce main concepts of an machine as follows:

- Variables: represents the state variables of the model of the specification.
- Invariants: describes by first order logic expressions, the properties of the attributes defined in the variable clauses. Typing information, functional and safety properties are described in this clause. These properties are true in the whole model. Invariants need to be preserved by events clauses.
- Events: $E(v)$ present transitions between states. Each event has the form $evt =$ **any** x **where** $G(x, v)$ **then** $A(x, v, v')$ **end**, where x are local variables of the event, $G(x, v)$ is a guard condition and $A(x, v, v')$ is an action. An event is enabled when its guard condition is satisfied. The event action consists of one or more assignments. We have three kinds of assignments for expressing the actions associated with an event: (1) a deterministic multiple assignment ($v := E(t, v)$), (2) an empty assignment (skip), or (3) a non-deterministic multiple assignment ($v : |P(t, v, x')$).

A Context consists of the following items:

- Sets: describes a set of abstract and enumerated types.
- Constants: represents the constants used by the model.
- Axioms: describes with first order logic expressions, the properties of the attributes defined in the CONSTANTS clause. Types and constraints are described in this clause.

To deal with complexity in modeling systems, Event-B provides a refinement mechanism that allows us to build the system gradually by adding more details to get a more precise model. A concrete Event-B machine can refine at most one abstract machine. A refined machine usually has more variables than its abstraction as we have new variables to represent more details of the model. In superposition refinement, the abstract variables are retained in the concrete machine, with possibly some additional variables. In data refinement, the abstract variables v are replaced by concrete ones w. Subsequently, the connections between them are represented by the relationship between v and w, i.e. gluing invariants $J(v, w)$.

2.3 RODIN

Rodin, an extension of the Eclipse platform, allows to create Event-B models with an editor. It also automatically generates the proof obligations of a model that can be discharged automatically or interactively. The architecture of the tool is illustrated in Fig. 1. Event-B UI provides users interfaces to edit Event-B models. Event-B Core has three components: static checker (checking the syntax of Event-B models), the proof obligation generator (producing simplified proof obligations that make them easier to discharge automatically), and the proof obligation manager (managing proof obligations and the associated proofs). The Rodin Core consists of two components: the Rodin repository (managing persistence of data elements) and the Rodin builder (scheduling jobs depending on changes made to files in the Rodin repository).

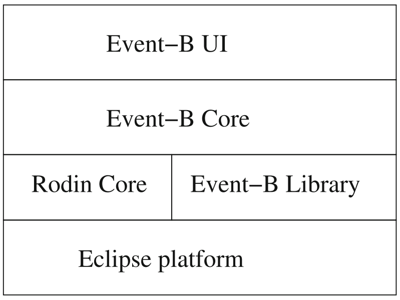

Fig. 1. Rodin tool architecture

3 Refinement-Based Approach to Analyzing Android GUI Designs

In this section, we first introduce the overall picture of our proposed method. After that, The formalization of Android GUI elements is given. Following this, we show how to model this formalization in Event-B notations and verify the desired properties with RODIN platform.

3.1 The Approach Overview

In the early phase of software development process, stakeholders often work together to initially define the GUI of the software. They then can add more precise and detail descriptions to GUI designs of the application. GUI design specifications of a general software system as well as Android applications describe the structure of each screen which is composed of UI elements and explain expected results or logic flows when users interact with these elements. The proposed approach consists of several steps depicted in Fig. 2.

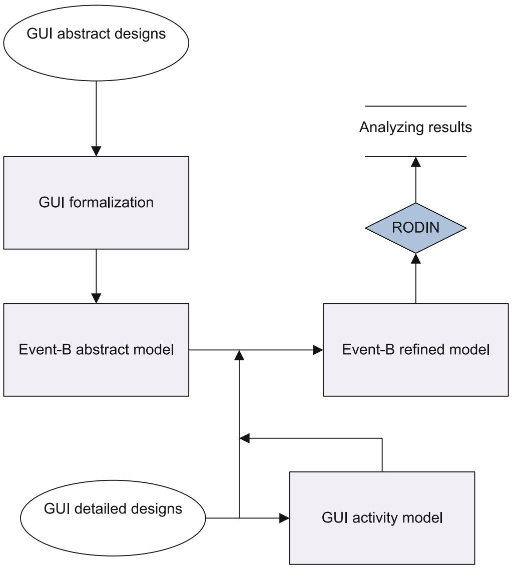

Fig. 2. The approach overview

The first steps in the proposed method is formalizing informal abstract GUI requirements and encoding them in Event-B notations. Along with development iterative phases, GUI designs are being more precise and detailed. These requirements, based on the refinement mechanism, are added to create a concrete Event-B one. The archived Event-B models in each steps can be verified via RODIN platform and its support plug-ins.

3.2 Formalization of Android GUI Elements

An Android application is different from an Android service as it has the GUI for users' interactions. Its GUI design describe each screen of the application which consists of UI elements, events, and corresponding actions. We define an application P as a 3-tuple $\langle W, S, C \rangle$, where

– W: a set of screens which are provided by *Activity* class in the framework.
– S: denotes the states of each screen including three essential ones of an Android activity.
– C: states the global constraints between screens.

Each screen composed of a collection of UI objects, events, and the corresponding actions. A screen $s \in S$ is formally defined as a 3-tuple $\langle Go, Ev, Ac\rangle$, where

– Go: denotes a set of graphical objects used in the screen. An object $o \in Go = \langle Pr, St\rangle$, where Pr and St represents its dynamic properties and state values respectively.
– Ev: states the intercept events when users interact with the screen.
– Ac: represents the consequences after the corresponding events happen.

Following the above definitions, we propose some rules to encode GUI requirements in Event-B notation as follows.

1. A screen s is mapped to an Event-B machine M.
2. Dynamic properties Pr of an UI object $o \in Go$ are translated to Event-B variables.
3. State values St of an UI object $o \in Go$ are translated to Event-B sets, constants, or axioms.
4. Events Ev are mapped naturally to events of the machine M.
5. Actions Ac associated with an event $e \in Ev$ are translated to assignments in THEN clauses of e.

3.3 Refinement for Analyzing GUI Specifications

In Sect. 3.2, we formalize the individual screen of an Android application with Event-B notations. One issue raised in verifying GUI specification is that analyzing multiple screens and verify the constraints between them. For simplicity, let assume that the application starts from the main activity, then goes into several branches. We continue to exploit Event-B refinement to formalize these (Fig. 3).

Following the rules presented in Sect. 3.2, we start by translating the main activity into the first abstract Event-B model. Then, we can visit other n screens from it. These, using the main activity's states, are modeled by refining the first abstract model. The remaining screens are modeled analogously.

4 A Running Example: Note Application

4.1 GUI Specifications

Note is a king of basic and popular Android applications providing users functionalities to create and view their personal memorial notes. It basically consists of four activities including *MainActivity*, *CreateActivity*, *EditActivity*, and

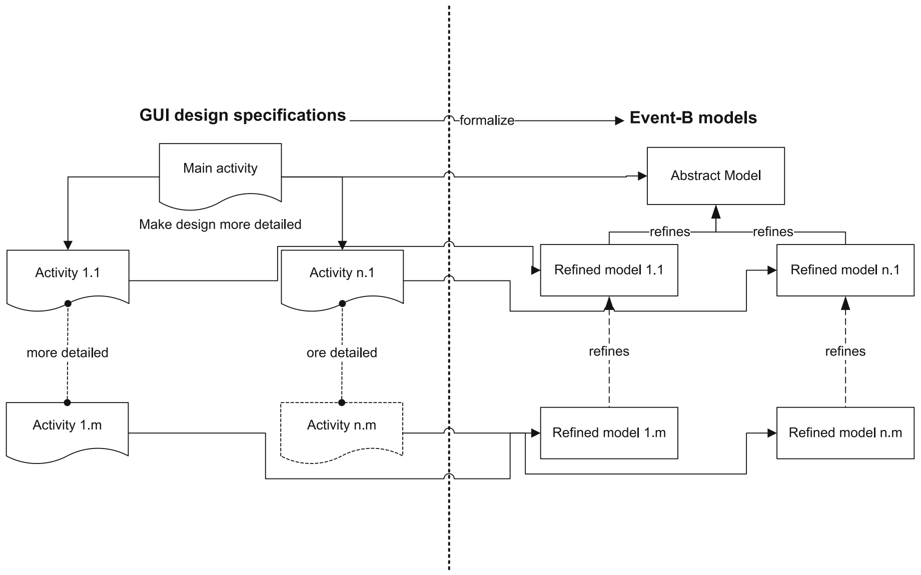

Fig. 3. Refinement for modeling UI designs

ViewActivity where *MainActivity* is the start-up screen. From the main screen, users can visit *CreateActivity*, *EditActivity*, or *ViewActivity* screens.

The *CreateActivity* screen has two events which are attached to two buttons such as *Save* and *Cancel*. The former allows users to save the note and go back to the previous screen. When users press the latter, they will ignore the current note and go back to the previous screen. The *EditActivity* screen acts in the similar way. The *ViewActivity* has only *Close* button to return the previous screen.

The application has to meet the requirement that only one of three screens *EditActivity*, *ViewActivity*, and *CreateActivity* activates at the same time. If *MainActivity* is active, then all other screens are destroyed.

4.2 Modeling and Verifying GUI Designs

We apply the proposed refinement-based approach to modeling the GUI specifications of *Note* application as follows.

Step 1. Initial model
The main activity is modeled as a context C_0 containing a set S representing four essential states of an Android activity and an abstract machine M_0 which *sees* C_0 and has four BOOL variables *main*, *createAtive*, *editActive*, and *viewActive* representing the active status of four activities.

Step 2. Refinement
CreateActivity is added into the design. This activity is modeled by a refined machine *M_CREATE* which refines machine *M_MAIN* and has a more variable *memo* and two events *Save* and *Cancel*.

Step 3. Refinement
Similarly, *EditActivity* is formalized by a refined machine *M_EDIT* which refines machine *M_MAIN* and has a more variable *memo* and two events *Save* and *Cancel*.

Step 4. Refinement
ViewActivity is modeled by a refined machine *M_VIEW* which refines machine *M_MAIN* and has a more variable *memo* and one event *Cancel*.

The paradigm of the translated model is depicted in Fig. 4.

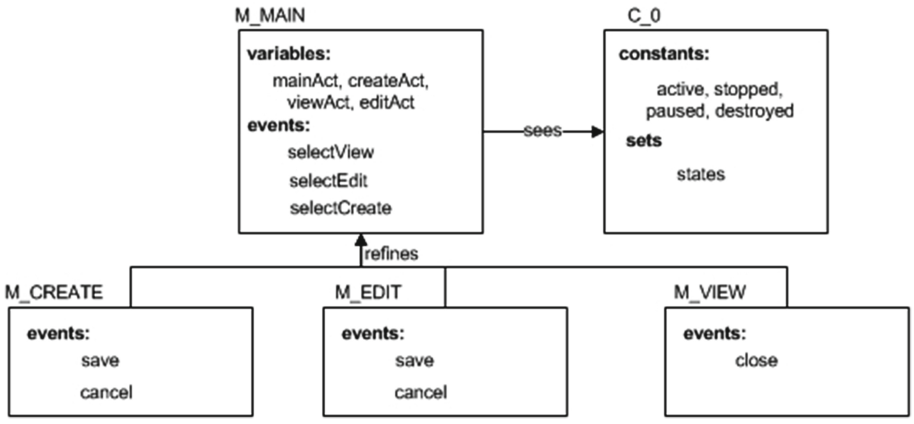

Fig. 4. Structure of the target model

The behavior constraints are formalized as two invariants:

$INV1 : (editState = active \implies \neg(viewState = active \vee createsState = active) \wedge (viewState = active \implies \neg(editState = active \vee createsState = active) \wedge (createState = active \implies \neg(viewState = active \vee editState = active)$ and

$INV2 : mainState = active \implies viewState = destroyed \wedge createState = destroyed \wedge editState = destroyed$

Verification results are depicted in Table 1. One discharged PO of *M_View* shows that in the screen *ViewActivity*, if users click *Save* button, it leads to the violation of the behavior constraint[1].

[1] The RODIN archive can be downloaded at http://125.212.233.56/NoteExample.zip.

Table 1. PO obligations statistics in RODIN

Machine	Total POs	Auto	Manual	Discharged
M_Main	19	19	0	0
M_Create	14	14	0	0
M_Edit	14	14	0	0
M_View	14	13	0	1

5 Related Work

GUI modeling is an emerging issue extracting many researchers. Many papers have been dedicated to formal modeling and checking UI specifications with different techniques. Bowen [5] provides formal notations of X11 programs' UI based on Z language. The author precisely specifies operations such as create, manipulate, and destroy windows to avoid ambiguity UI requirements. This method, however, does not provide a way to verify the desired properties. Clement, in [8], adopts VDM and its proof obligations to develop Window interfaces. The approach also makes use of VDM support tools to generate the implementation of the system in Prolog language. The difference between this paper and their paper is that we can generate the executable program in several languages (e.g., Java, C++) from the target Event-B model. Palanque *et al.* present a formalism called Interactive Cooperative Objects, which is based on high-level Petri nets. Their proposed approach is suitable for discrete interaction between users and application. It can check several properties of interface requirements such as absence of a deadlock, integrity constraints, etc. Judy Bowen *et al.*, in [4], integrate both user-centric design methods and Z formal method to propose a formal model of UI. The works presented in [3,9] also use Event-B proof obligations as the foundation to validate HCI requirements. These works need a hypothesis that the requirements are described in Concur Task Tree (CTT) models.

The common issue of these papers, in our opinions, is that they are unspecific enough to apply for Android applications in which dialogs inherit *Acivity* class. One of advantages of our method is that we can apply the tool, in [7], to generate Android codes from Event-B models.

6 Conclusions and Future Work

GUI requirement analysis plays an important role in mobile application development. Many work has been dedicated to GUI verification at early phase to reduce the cost. In this paper, we propose a new refinement-based method for modeling and verifying Android GUI designs. The defects of the design can be realized at any refinement step. Desired properties can be checked based on the generated proof obligations and almost be proved automatically. One limitation of this method is that Event-B primitives data type are not rich enough to describe all Android UI elements. This, however, can be overcome by incorporating with

Theory Plugin [6] to define more data structures. We also intend to expand this work with analyzing concurrency issues of Android application interfaces.

Acknowledgments. This work is partly supported by the project no. 102.03–2014.40 granted by Vietnam National Foundation for Science and Technology Development (Nafosted).

References

1. Event-B and the Rodin platform (2012). http://www.event-b.org
2. Abrial, J.-R.: Modeling in Event-B: System and Software Engineering, 1st edn. Cambridge University Press, New York (2010)
3. Ait-Ameur, Y., Baron, M.: Formal and experimental validation approaches in hci systems design based on a shared event b model. Int. J. Softw. Tools Technol. Transf. **8**(6), 547–563 (2006)
4. Bowen, J., Reeves, S.: Formal models for informal GUI designs. Electron. Notes Theor. Comput. Sci. **183**, 57–72 (2007)
5. Bowen, J.P.: X: why z? Comput. Graph. Forum **11**, 221–234 (1990)
6. Butler, M., Maamria, I.: Practical theory extension in event-b. In: Liu, Z., Woodcock, J., Zhu, H. (eds.) Theories of Programming and Formal Methods. LNCS, vol. 8051, pp. 67–81. Springer, Heidelberg (2013). doi:10.1007/978-3-642-39698-4_5
7. Cataño, N., Rivera, V.: EventB2Java: a code generator for event-b. In: Rayadurgam, S., Tkachuk, O. (eds.) NFM 2016. LNCS, vol. 9690, pp. 166–171. Springer, Cham (2016). doi:10.1007/978-3-319-40648-0_13
8. Clement, T.: The formal development of a windows interface. In: Proceedings of the 3rd BCS-FACS Conference on Northern Formal Methods (3FACS 1998), p. 6, Swinton, UK. British Computer Society (1998)
9. Cortier, A., d'Ausbourg, B., Aït-Ameur, Y.: Formal validation of java/swing user interfaces with the event b method. In: Jacko, J.A. (ed.) HCI 2007. LNCS, vol. 4550, pp. 1062–1071. Springer, Heidelberg (2007). doi:10.1007/978-3-540-73105-4_116
10. Courtney, A.A.: Modeling user interfaces in a functional language. Ph.D. thesis, New Haven, CT, USA. AAI3125177 (2004)
11. Le, H.A., Nakajima, S., Truong, N.T.: Formal analysis of imprecise system requirements with event-b. SpringerPlus **5**(1), 1000 (2016)
12. Palanque, P., Paternó, F. (eds.): Formal Methods in Human-Computer Interaction, 1st edn. Springer, New York (1998)
13. Palanque, P.A., Bastide, R.: Petri net based design of user-driven interfaces using the interactive cooperative objects formalism. In: Paternó, F. (ed.) Interactive Systems: Design, Specification, and Verification. Focus on Computer Graphics: Tutorials and Perspectives in Computer Graphics. Springer, Heidelberg (1995)