The Design of Mobile Apps: what and how to teach?

SYLVIA STUURMAN and BERNARD E. VAN GASTEL and HARRIE J. M. PASSIER
Open Universiteit Nederland

Mobile applications (or mobile apps or apps for short) gain importance, and will, as is our expectation, find a place in the curricula of Computer Science and Software Engineering. In books, courses and tutorials, not much attention has been given to the design of mobile applications.

In this paper, we describe the anatomy of mobile apps, using Android as an example. Based on this anatomy, we offer an inventarization of modeling techniques that can be applied to adequately design mobile apps. Some of these modeling techniques are already taught in most curricula, albeit in different courses. A modeling technique that is useful for several aspects of mobile apps is the Interaction Flow Modeling Language (IFML). This modeling technique would have to be introduced when one would like to teach students how to design apps.

We also describe which strategies can be followed when introducing mobile apps in a curriculum: as a subject of a course, together with knowledge of the concepts and the necessary modeling techniques, or as examples in different courses. We discuss advantages and disadvantages of both approaches.

Categories and Subject Descriptors: D.2.10 [**Software Engineering**]: Design—*Representation*; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education*; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Curriculum*

General Terms: Designing Mobile Applications, Computer Science curriculum

Additional Key Words and Phrases: Android, Universal Modeling Language, UML, Interaction Flow Modeling Language, IFML, Curriculum

1. INTRODUCTION

In courses on Software Design, the domain that is often (implicitly) presumed, is the domain of business applications, running on a server. This is reflected in the examples that are used in textbooks [Evans 2004; Larman 2012].

The omnipresence of mobile applications (or apps for short) shows that the dominance of server-side applications is decreasing. More and more, apps are introduced in Computer Science curricula. Though universities teach knowledge that is as independent of specific technologies as possible, it is inevitable to use specific technologies to teach certain subjects. Both teachers and students prefer recent technology. Because of these reasons, we foresee a shift from the implicit domain of business applications to the domain of mobile applications.

Other reasons are the fact that the user interface of mobile apps has more possibilities than input through the keyboard or the mouse, that mobile applications are often part of a client-server or a peer-to-peer application, that there is hardware involved with multiple sensors, that there is limited memory and there are restrictions with respect to power usage, that data handling is different than the traditional file-system-model, that interaction between mobile applications is common, and that there are programming issues such as the life cycle of applications, designing for multiple platforms, or security and privacy [Gordon 2013]. In short, mobile applications are rich and complex, and thus suitable to teach many aspects of Computer Science.

There are more reasons. Because of the competitive market it is extremely important in mobile applications to optimize for changeability and adaptability, and therefore mobile applications lend themselves to show the importance of this quality aspect to students. Mobile applications are event-driven, in contrast to server-side applications as they are taught. And not the least important, learning to program mobile applications motivates students.

The implicitly presumed domain of applications not only influences the examples that will be used, but also the modeling techniques that will be taught. Teaching suitable modeling techniques for the design of mobile applications is important because of the aspects we mentioned before, such as the importance of designing for changeability.

We pose two questions. The first question is: which modeling techniques does one need to design mobile applications? The second question is: if one would like to integrate mobile applications into the curriculum, two strategies may be used to do so. On the one hand, one may use apps as examples in existing courses; on the other hand, apps might become the focus point, to teach concepts and modeling techniques necessary to design apps. What are the advantages and disadvantages of both approaches?

With respect to the first question, we describe the anatomy of mobile apps, using Android as an example. Based on these concepts, we make an inventarization of modeling techniques. Our contribution is a proposal for a set of existing modeling techniques that are suitable to model different aspects of a modern mobile application. We do not restrict ourselves to UML, but also take other approaches into account. Knowing these techniques, a student could model the different aspects of a mobile application.

With respect to the second question, we describe which strategy one should choose if one sees designing apps as a complex task, which integrates knowledge from several areas and asks for several skills. We check existing curriculum guidelines for Computer Science and Software Engineering with respect to the two strategies, and discuss advantages and disadvantages.

This paper is organized as follows: In Section 2, we describe the anatomy of a mobile application. We will use Android apps as an example. Section 3 shows which current modeling techniques might be used to model the aspects of apps that we described in Section 2. In Section 4, we describe advantages and disadvantages of the two strategies to integrate modeling apps in the curriculum. In Section 5, we describe related work, and show the difference with our work. We summarize our conclusions in Section 6.

2. THE ANATOMY OF ANDROID APPS

Most mobile platforms support similar constructs though the syntax varies. We use the Android platform to illustrate aspects of mobile applications that deserve attention during the design phase. We choose Android because its open source nature is ideal to study the inner workings.

An Android app runs on the Android operating system in its own Java virtual machine, within its own process, in isolation from other apps.

2.1 Elements

Android apps consist of four types of building blocks:

Activities An activity is associated with a single screen. In texts on Android development, 'design' is often a synonym for the design of the graphical user interface of each screen of an app [1] Here, we focus more on the design of the functionality of each activity and on the functionality of its user interface.

Services Services run in the background: there is no associated screen. An activity may use a service. Services may perform long-running operations or perform work for remote processes.

Content providers A content provider manages data. Data may be stored locally (either private for the app or shared with several apps) or on the web. Data may be stored in a file system or in a database. When an application allows other applications to read data, access is through the content provider (unless the data has a simple structure and is private for the app). Activities or services also only read or write data through a content provider.

Broadcast receivers Broadcast receivers respond to system-wide messages (for instance about a low battery, or an announcement that the screen has been turned-off).

2.2 Communication

Activities, services and broadcast receivers are activated through an *intent*. An intent is an asynchronously handled message carrying a characterization of the action to perform (for instance 'start', or 'view', or 'send'), and it may carry an URI or data to act upon. Intents may be anonymous, in which case the operating system searches for an activity or service or broadcast receiver in an app that is able to perform the action, or it may be direct, which means it is directed to a specific activity, service or broadcast receiver. A result of an intent is delivered in the form of a callback.

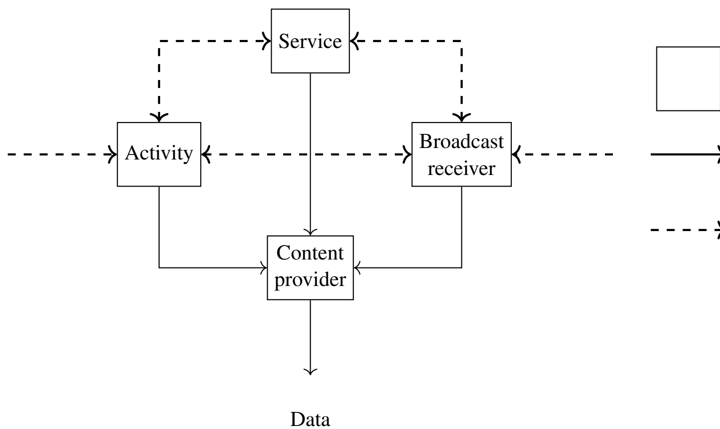

Fig. 1. The components of an Android app

Figure 1 shows these four types of components. Content providers are the only components that should fetch or save data. Activities, services and broadcast receivers may communicate with content providers through 'ordinary' associations (depicted as solid arrows). Activities, services and broadcast receivers may communicate with each other through intents (depicted as dashed arrows). The system may contact broadcast receivers (for notices) and activities (for instance to start an application, or to ask it to show a video), also through intents. Intents may be both directed or anonymous (we have no way to show that in the figure). Intents may be used within one app or between apps. Activities and services may thus send intents to the system or to other activities and services within the app; broadcast receivers, activities and services may receive intents from the system.

2.3 Life cycle

Activities and services go through a *life cycle*, and each of the transitions of one stage in the life cycle to another stage is associated with an *event*. Programming an activity or a service thus means in the first place that one specifies what should be done when each event takes place.

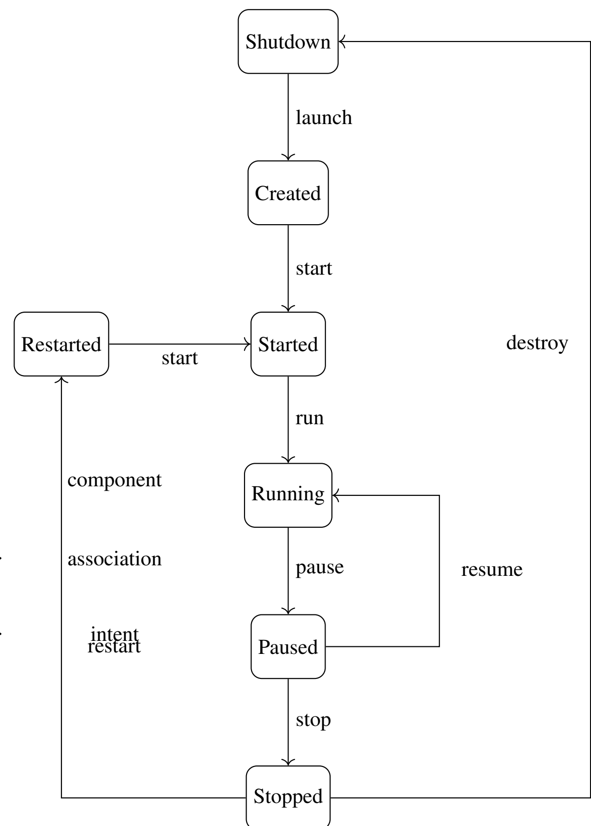

Fig. 2. The lifecycle of an activity

Figure 2 shows the states within the lifecycle of an activity. For each state transition, the activity receives an event for which an event handler may be defined, specifying what should be executed when the event takes place event.

[1] Android Developer Guides, Design and Documentation.https://developer.android.com/

2.4 Threads

Each app runs in a single process and by default all components run in a single thread. However, additional threads may be created, and one can reserve a separate thread for each component of an app. Because the responsiveness of mobile apps are important, separate threads are often needed to avoid a 'freezing' user interface. Services run, by default, in the main thread. Often, one would create new threads within a service. The same applies for content providers that store data through the internet.

3. MODELING ANDROID APPS

Modeling an application before building one, serves several purposes. One may use models to communicate with stakeholders (UML use case diagrams are an example), to have a visual image of the solution at a higher abstraction level than the code (UML class diagrams are an example), to be able to generate code (UML class diagrams are again an example), or, for instance, to be able to analyze the solution with respect to specific properties (Finite State Machines are an example). A diagram technique may be used for one or more of these purposes. Here, we do not discern for which purposes each diagram technique could be used.

Android Apps are object oriented programs written in the programming language Java and as such can be designed using the standard UML notations as class diagrams and sequence diagrams. But mobile applications are often significantly different from standard object oriented programs. For example, mobile applications are event-driven, mobile applications as embedded software should use limited device resources efficiently, and the development of mobile applications demands additional worries about the short time-to-market. These issues need special attention and the use of specialized design techniques [Kraemer 2011; Parada and Brisolara 2012; Stringfellow and Mule 2013].

This means that a course about designing and implementing Android Apps requires among other things proficiency in the concepts of object orientation, event-driven programming, and the Android architecture [Riley 2012]. Here, we focus not on the concepts but on modeling techniques. First, we describe the results of interviews we had with app developers about their modeling activities (or absence of them), then we discuss how to model the anatomy of Androids apps, and finally, we discuss modeling other aspects of Android apps.

3.1 In practice

For evaluation purposes we contacted a small number of former students to get information about their experience in developing mobile applications. Although it was by no means intended as an extensive and/or formal survey, a number of interesting patterns emerged. For most, they indicated that the user interface (look and feel, to the level of details) is the most important aspect of an app. By using user stories and later on storyboards, they quickly get an idea of what an app should be like. Some use paper prototyping at a regular basis to get the first results. They continue with agile development of the mobile app, and using rapid prototyping methods and A/B testing combined with regular contact with their customers ensures customer satisfaction. Some indicated that they start with the part of the application that has the highest risk, and continue with reassessing the risk associated with the remaining parts after each step in the development process. In this regard, business decisions are leading in the process.

When specifically asked about their application design, they were at a loss. Their first reaction was to claim that other aspects were more important, i.e. the user interface, the business process, agile development process, and customer satisfaction. As they talked more, keywords as declarative methods, facades, mediators, services, frameworks (for instance, AngularJS or Famo.us) arose, and associated design patterns, the unsuitability of model-view-controller, and the usefulness of the model-view-presenter design pattern. They indicated that although they design an app in their mind, they did not have the proper design techniques to design an application on paper, as they lacked the necessary graphical and semantic representations of important aspects of their application design.

3.2 Modeling the anatomy

3.2.1 *Activities.* Activities are associated with screens. Use cases can therefore be detailed into a flow of screens, with user input acting as a trigger for a transition to another screen (and therefore: activity).

There are several aspects of activities that one would like to be able to model: the functionality, the contents of the screen itself (with a focus on the functionality), the flow of activities and the specification of what to do at each life cycle event. For the functionality, one may use UML class diagrams and UML sequence diagrams, because activities are (subclasses of) Java classes.

Activity Diagrams
The obvious diagram technique to model the flow of activities seems to be the UML activity diagram. UML activity diagrams are meant to model procedural computations, work flows, and system level processes [2]. In an activity diagram, activities can send signals, can wait for signals, and the execution of one activity may lead to the execution of another activity. Activities may be nested.

Activity diagrams may be transformed into Petri Nets for analyzing purposes [Störrle and Hausmann 2004].

Interaction Flow Modeling Language
An activity is associated with a screen. The flow of activities in an Android app can therefore also be modeled using the (relatively new) Interaction Flow Modeling Language (IFML) [3]. This diagram technique has been developed for the design of (mainly) the client-side of web applications. The most important constructs of IFML are the following:

View container An element of the interface that comprises elements displaying content and supporting interaction and/or other view containers, for instance a screen.

View Component An element of the interface that displays content or accepts input, for instance a button.

Event An occurrence that affects the state of the application, for instance a user pressing a button.

Action A piece of business logic triggered by an event; either server-side or client-side.

Navigation Flow An input-output dependency. The source of the link has some output that is associated with the input of the target of the link, for instance the link from a row in a list of artists to a View Component showing information about that artist.

Data Flow Data passing between View Components or Actions as consequence of a previous user interaction, for instance the transfer of information of a shopping cart to a payment action.

[2]OMG Unified Modeling Language (OMG UML),Superstructure, version 2.2, http://www.omg.org/spec/UML/2.2/Superstructure/PDF/
[3]http://www.ifml.org

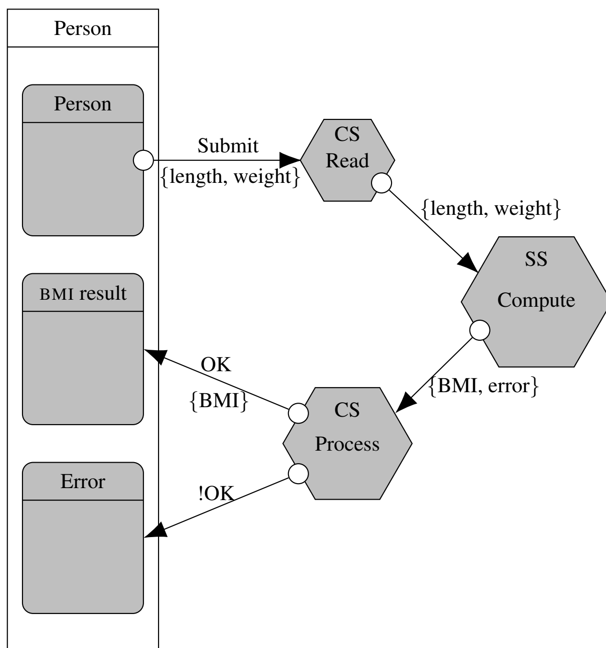

Fig. 3. An IFML diagram

Figure 3 shows an example of IFML used in a design of a web application. Person, BMI result and Error are three view components within the view Person. An event (the user submitting a form with his or her weight and length) triggers the activity Read, which takes place at the client-side (CS). The event that this activity is ready triggers an activity Compute on the server. The event that this activity has been processed triggers an activity Process on the client, while the BMI index and an error code are sent along. At the client, the Process activity either produces an OK event, which means that the BMI will be displayed in the BMI view component, or an error event, which means that the error view component will show an error. An event triggers the replacement. Events may be named, as we have done here for the Submit, OK en !OK events.

As is shown, IFML can be used to describe the flow between screens, and the associated activities. A screen may be modeled by a view container or a view component (depending on the level of detail of the model), while the associated activities may be modeled using actions.

There is a UML profile for IFML, which means that it can be adopted by general UML tools; at this moment there is only one dedicated IFML tool available.[4]

3.2.2 *Services.* The functionality of services may be modeled in the same way as the functionality of activities, using UML class diagrams and sequence diagrams. It is not possible, however, to indicate that they run in the background. Often, one would like to be able to model when services perform a certain action (for instance at certain events), and how they use threads to run in the background. Modeling events and threads is discussed below.

3.2.3 *Content providers.* For the functionality of content providers, the same applies as to services. One could model the type of storage using UML classes.

[4]http://www.webratio.com/portal/content/en/ifml-standard

3.2.4 *Broadcast receivers.* To model the broadcast receiver, one would like to be able to model which kind of notifications may be received, and which activity or service will handle the notification. As we have seen, IFML could be used for that purpose.

For broadcast receivers, the same applies as to services: one needs to model events and threads.

3.2.5 *Communication.* As can be seen in Figure 1 (where we used 'fantasy' arrows with custom semantics), it is not straightforward how to model communication through intents. One would like to specify whether an intent is directed or anonymous, and what action and which data are associated with an intent. Also, one would like to be able to specify for each activity and service what actions they can perform (for anonymous intents that are associated with actions).

The only research we are aware of, on this subject, uses custom-made building blocks in UML Activity diagrams [Kraemer 2011]. The most obvious choice seems to be an association in a UML class diagram with a classifier specifying that it is an intent, and specifying whether the intent is directed or anonymous. One could also use an association class, to specify the data that are sent with the intent. The arrows in an IFML diagram are very similar to directed intents, as they may carry data. The only restriction is that the arrows in an IFML diagram are bound to events.

At the moment of writing, there is no standard way to model intents. The same applies to message passing in general: it is unclear how to model anonymous message passing, for example.

3.2.6 *Life cycle.* As can be seen in Figure 2, it is easy to model the lifecycle of an activity using a UML state diagram. What is desirable, however, is a means to model the desired actions at each event connected with the life cycle. This might be done by specifying the state the activity has to reach after such an event, but that is difficult. For instance, in a mobile app, there is no guarantee that an event handler is executed when an app stops. One would like to be able to specify what one would like to *try* to achieve at a certain event.

The life cycle of Android apps and the fact that an app may receive broadcast messages are specific cases of the fact that Android apps are event-driven. There are two aspects of events that one would like to model:

—A specification of the state transition that is triggered by an event: the state before the event has taken place, and the state after execution of what is triggered by the event. This is a declarative specification of event handling, focusing on what should be achieved.
—A specification of which actions to take at a certain event. This is an imperative specification of event handling, focusing on how one achieves what is specified above.

With respect to the first issue, the main difficulty is that the state may have been altered between two related events, for instance between the event that an Ajax-call has been made and the event that the response is received. For these issues, Finite State Machines (a graphical notation with a corresponding algebraic notation that lends itself for analyzing [Magee and Kramer 2006]) are a usable modeling technique (see for instance [Marchetto et al. 2008; Altayeb and Damevski 2013]. Instead of Finite State Machines, one may use UML State-charts, which can be translated into Finite State Machines for analysis purposes [Drusinsky 2011]. Another possibility is the use of Petri Nets [Benveniste et al. 2003]. One has to bear in mind that there is no guarantee, on the Android platform, that the next state is reached after a certain event.

With respect to the second issue, one could use activity diagrams, and – again – IFML. With IFML, it is possible to specify which action is triggered by which event, and one may specify which data are transmitted as well. Handling broadcast messages for instance, can be modeled in detail using IFML, by specifying the data of a message, and the action or service to trigger. The same applies to life cycle events and events triggered by sensors of the device (either detecting user input or a change in the surroundings).

3.2.7 *Threads.* Because responsiveness is important, one would like to model which functionality is processed in a separate thread. Of course, possible problems with threads and state should be prevented by modeling threads and state, and analyzing those models.

Threads are necessary in event-based systems that must be responsive. The main thread of an Android application is the one that processes UI events, and this thread should not perform heavy computations or long blocking operations in response to user events [Yang et al. 2013]. Threads are notoriously difficult to 'get right', because they are inherently non-deterministic [Liu et al. 2011]. To prevent state-related problems with threads, one may use Finite State Machines, but this can easily lead to a state explosion.

Threads are not only hard to use; it is also difficult to teach concepts around threads, like concurrency and synchronization, in such a way that students really understand these concepts. There have been explicit attempts to focus on these aspects. Li et al., for instance, describe a course on concurrency in which they teach how to use state and sequence diagrams to model concurrent systems. They also show the well-defined transformation from state diagrams to threads-based implementations of monitor constructs and condition variables, and a corresponding transformation to a message-passing implementation [Li and Kraemer 2013].

Because the hard part of using threads lies in the different states that the program may be in, the modeling techniques for threads are essentially the same as for events.

3.3 Modeling other aspects

Some aspects of Android apps might be underexposed with our focus on the anatomy. Here, we discuss how one may model some of these aspects. We left out, for instance, permissions (which are declared in the manifest file) and security.

3.3.1 *User interaction.* The user interface of mobile apps is very important. It must be intuitive and clear, on devices that may differ in the properties of the screen and in sensors for user input. With respect to user interaction there is the question of the visual design, which falls beside the scope of this article, but there is also the question which kind of inputs each screen offers, and how those possibilities connect to the possible flow of activities: the question of the functionality of the user interface.

Use Case diagrams
UML use case diagrams are a means for specifying required usages of a system [OMG 2009]. As such, they may be used to specify which features a user might expect from an app, but use case diagrams cannot be used to model more detailed user interaction.

Sequence diagrams
UML sequence diagrams may model what happens after a user has taken a certain action. As such, they are geared to the specification of a specific interaction of objects, to fulfill a certain use case, but as use case diagrams, they cannot be used to model more detailed user interaction.

Interaction Flow Modeling Language
IFML cannot be used to model the 'artistic' aspect of the user interface, but it can be used to specify which elements each screen contains (where an element is a view component in IFML terms: an element that displays content or accepts input). Therefore, IFML can be used as the interface between designers and developers: it specifies the functional elements of each screen, which can be used by designers to design each screen.

3.3.2 *Distribution of Functionality.* The functionality of a mobile app may be divided between the client and the server. Some functionality may be implemented on both sides, with a different purpose (for instance, in the case of form validation). The question of how to divide the functionality between client and server is, in essence, an architectural issue. The usual modeling technique to describe these decisions is a UML deployment diagram [OMG 2009].

In the case of mobile apps, IFML can also be used to model the division of functionality between client and server. Each action is, by default, placed on the server. An action can be provided with [Client] or [CS] (client-side) to show that it is performed on the client.

Apps may be seen as distributed processes: in general, they have a client process and a server process, and in some cases, there may be a peer-to-peer aspect. This aspect of distributed processes may also be seen in terms of events: a message from the server, from a peer or from another app can be modeled as an event.

3.3.3 *User interface design patterns.* User interface design patterns play an important role in the design of mobile apps. These design patterns often concern the visual aspects of the graphical user interface. An example is to always display the Cancel button to the left and the OK button to the right [Nudelman 2013]. Other patterns concern the functionality of the user interface (such as asking whether the user has mistyped a search term, instead of assuming a case of mistyping and showing results for what the user probably meant). It would be useful if a modeling technique would allow one to add these patterns as ready-made building blocks for a graphical user interface.

Most patterns for Android are related to the visual aspects of the user interface. There is no support in any diagram technique for these specific design patterns. One can imagine, however, that it is possible to supply building blocks in IFML for some of these patterns. When the screen of an activity should, for instance, contain an OK button and a Cancel button, an IFML building block could refer to the Cancel/OK pattern [Nudelman 2013] that can be used by the designer.

4. HOW TO TEACH

Mobile apps are becoming and will become a subject in many curricula. The Curriculum guidelines for undergraduate programs in Computer Science 2013 for instance, has a new (elective) knowledge area 'Platform-based development' for mobile applications [Curricula 2013], and the word 'mobile' occurs in various other knowledge areas.

Modeling Android apps combines modeling techniques from different knowledge areas. The Curriculum guidelines for graduate degree programs in Software Engineering [GSwE 2009] mentions Finite State Machines and Petri Nets in the knowledge area of Formal methods, while UML class and sequence diagrams fall under the knowledge area of object oriented design. The same applies to the curriculum guidelines for undergraduate degree programs [Curricula 2013]. When one would have to place IFML in one of the knowledge areas, designing user interaction would be the most appropriate. The design of an Android app thus combines modeling

techniques from different knowledge areas. This is in line with our expectation, because mobile apps show concepts from different knowledge areas [Gordon 2013].

This means that there are two strategies to integrate modeling mobile apps in a curriculum. On the one hand, modeling mobile apps could be the subject of one or more courses, in which the concepts and the modeling techniques needed to model apps could be explained. On the other hand, the concepts could be taught in, for instance, a course on object oriented programming, while the necessary modeling techniques could be taught in courses on human computer interaction, object oriented design and distributed processes. Mobile apps would be used in examples in such courses.

4.1 Mobile apps as a subject

Learning how to model an Android app (or any mobile app) can be seen as a complex task. Learning complex tasks should be based on real-life authentic tasks [Merrill 2002]. This is because in complex learning, students should integrate knowledge, skills and attitudes. Such an integration does not arise automatically. A focus on learning tasks based on real-life authentic tasks is needed to help learners to integrate knowledge, skills and attitudes [Merriënboer and Kirschner 2001]. Students should be provided with support and guidance while solving such a task. That guidance should tell students how to recognize an acceptable solution and should provide guidance to the solution process: procedural information is required when one offers authentic tasks to enable complex learning [Kirschner et al. 2006]. This is also in line with guideline 14 'The curriculum should have a significant real-world basis' and 18 'Important efficiencies and synergies can be achieved by designing curricula so that several types of knowledge are learned at the same time' of the Curriculum guidelines for undergraduate degree programs in Software Engineering [Curricula 2009].

This forms an argument for teaching the concepts of mobile apps and different modeling techniques within the context of mobile apps, instead of within different knowledge areas. Support for the task of modeling a mobile app consist of knowledge about the concepts and knowledge about the modeling techniques. Guidance consists of rules to follow: where does one begin, how does one use the different techniques, in which sequence? This kind of guidance should be developed: there is no generally accepted procedure or 'how to' for modeling apps.

4.2 Mobile apps as examples

There are difficulties with the approach of apps as a subject. The Curriculum guidelines for graduate degree programs in Software Engineering, for instance, tells us: 'The principles underlying Software Engineering change relatively slowly, but the technology through which Software Engineering is practiced keeps changing at breakneck speed. Educational institutions must adopt explicit strategies for responding to changing technology without being caught in the trap of simply training the latest technology. A key to this is organizing the curriculum around enduring principles and planning to change the example technologies regularly.' [GSwE 2009]. This makes sense, because the knowledge areas will stay the same during time (while their content will evolve slowly), while application areas or domains will evolve much quicker.

Also, it would require a complete overhaul of the curriculum to assign concepts and modeling techniques that now belong to specific knowledge areas, to courses on mobile development, in the form of real-world problems with the necessary knowledge and procedural guidance. It is much easier to introduce mobile apps in the form of examples in different courses.

5. RELATED WORK

Research on mobile systems, pertinent to this subject, has been carried out in three areas: research on design methods (often with associated tools) for mobile apps, research on the concepts within mobile apps, and research on how to teach engineering mobile apps.

An example of research on design methods for mobile apps is the work of Parada and de Brisolara on a model-driven approach for the development of Android applications [Parada and Brisolara 2012]. Here, class diagrams and sequence diagrams of standard UML are proposed to model Android applications. Heitkötter and Majchrzak propose a domain-specific language to model a mobile application. [Heitkötter and Majchrzak 2013]. Ko et al. offer an approach in which standard UML is extended using stereotypes, tagged values and constraint meta-classes to model Android applications [Ko et al. 2012]. Kraemer et al. have a different approach. Here, the focus is on the responsive nature of mobile apps; the proposal is to design them using UML Activity diagrams, augmented with State Machines, for which they supply building blocks representing different Android concepts [Kraemer 2011].

What these approaches have in common is that they offer a specific method to design mobile applications, often by using UML. Our focus is different: we try to discern the different aspects of mobile apps and explore which modeling techniques might be useful for those concepts, with the purpose of providing a set of modeling techniques that might prepare students for the design of mobile apps; not by prescribing one method, but by providing students with different possibilities.

Gordon discusses the concepts that are relevant for mobile apps: user interface design and usability, device cooperation, hardware issues, data handling, application interaction and programming issues [Gordon 2013]. His focus is on the knowledge that students need, in different knowledge areas, to be able to create mobile apps, while our focus is on the design techniques they should be taught. Our study complements Gordons research.

Altayeb and Damevski argue that one should teach a model-first approach in developing mobile apps [Altayeb and Damevski 2013]. They use the Prolemy II environment [Davis II et al. 1999], which offers modeling techniques for Communicating sequential processes (CSP), continuous-time modeling, discrete-event systems, discrete-time, process networks, Petri Nets, synchronous dataflow, synchronous/reactive, and graphics and 3D animations. This is a choice for the first of our strategies to introduce mobile apps in the curriculum: using them as a subject, bundled with the necessary knowledge and techniques. Riley describes how he uses Android programming to teach Java and advanced programming skills [Riley 2012]. Stringfellow and Mule describe the use of an Android project in a course on Software engineering [Stringfellow and Mule 2013]. These researchers have in common that they describe how they use Android or mobile applications in general to teach certain areas of computer science. Our focus is on those modeling techniques that should be taught to students for the domain of mobile applications.

6. CONCLUSION AND DISCUSSION

In this article, we addressed two questions: the question of which modeling techniques we should teach our students with respect to the design of mobile applications, and the question of how to integrate modeling mobile applications in the curriculum.

With respect to the first question, we described the anatomy of Android apps, and made an inventarization of modeling techniques for each of the elements of these apps. We did the same for other

aspects of Android apps. We showed that the complex nature of mobile applications and the complex concepts that play a role in them, demand skills in and knowledge of a variety of modeling techniques for the design of mobile applications.

Practitioners report that they do not have the proper design techniques, and as a result mobile apps are not explicitly designed in practice. In particular when mobile apps become subject in a curriculum, it seems worthwhile to offer students relevant modeling techniques.

When comparing the modeling techniques that are suitable to model different aspects of Android apps with what is taught in Computer Science and Software Engineering curricula, the first observation is the fact that the Interaction Flow Modeling Language is a 'fit' for several aspects of Android apps, while it is not covered in most curricula (we could not find any curriculum that covered this modeling technique). This means that, when one would like to teach students how to design mobile apps, IFML should belong to the standard set of modeling techniques that are taught to students in Computer Science and Software Engineering. It would be worthwhile to adopt IFML in such curricula.

Other aspects of mobile apps may be modeled using modeling techniques that are, in general, taught within, for instance, courses on object oriented design and on distributed systems or on formal methods. The IFML could be taught in courses on, for instance, user-interface design. While most modeling techniques already have a place in most curricula, the fact that these modeling techniques are spread over such different courses, brings us to the second question.

We have described that modeling Android apps can be seen as a complex task, and should therefore, preferably, be taught as a complex task: using authentic tasks, with support on the concepts and the modeling techniques, and with guidance on how to proceed while designing an app.

Such an approach is one of two strategies to teach students modeling techniques for the context of mobile apps:

—Teach the different modeling techniques in different courses, along the lines of the above-mentioned knowledge areas: state-related modeling techniques in a course on formal methods, UML in a course on object-oriented design, and IFML in a course on user interaction design.

—Focus on designing and building mobile apps from the start, introducing the different modeling techniques and concepts, and gradually make it more complicated.

The first approach is in line with the Curriculum guidelines for graduate degree programs in Software Engineering [GSwE 2009]. All techniques could be integrated in a lab project around an app.

The second approach is in line with what is needed for complex learning, and is in line with the Curriculum guidelines for undergraduate degree programs in Software Engineering [Curricula 2009]. Adhering to this second approach would mean that the curriculum, in most cases, would have to be overhauled completely. Often, that is impossible.

In either approach, IFML should be introduced, and guidance on how to design mobile apps should be developed.

ACKNOWLEDGMENTS

We would like to thank Harold Pootjes for his valuable knowledge of IFML. We would also like to thank Mark Marijnissen, Andres Lamont, Bram den Teuling, and Jop van Heesch, for giving us insight in how they design mobile applications.

REFERENCES

Badreldin Altayeb and Kostadin Damevski. 2013. Utilizing and Enhancing Software Modeling Environments to Teach Mobile Application Design. *Journal of Computing Sciences in Colleges* 28, 6 (June 2013), 57–64.

Albert Benveniste, Eric Fabre, Stefan Haar, and Claude Jard. 2003. Diagnosis of asynchronous discrete-event systems: a net unfolding approach. *Automatic Control, IEEE Transactions on* 48, 5 (2003), 714–727.

The Joint Taskforce On Computing Curricula. 2009. Software Engineering 2004, Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. (2009).

The Joint Taskforce On Computing Curricula. 2013. Computer Science Curricula 2013. (2013).

J Davis II, Mudit Goel, Chirstopher Hylands, Bart Kienhuis, Edward A Lee, Jie Liu, Xiaojun Liu, Lukito Muliadi, Steve Neuendorffer, John Reekie, and others. 1999. *Overview of the Ptolemy project*. Technical Report. ERL Technical Report UCB/ERL.

Doron Drusinsky. 2011. *Modeling and verification using UML statecharts: a working guide to reactive system design, Runtime Monitoring and Execution-based Model Checking*. Newnes, Elsevier, Oxford, UK.

Eric Evans. 2004. *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, Boston, United States.

Aaron J Gordon. 2013. Concepts for mobile programming. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. ACM, ACM, New York, NY, USA, 58–63.

GSwE 2009. Graduate Software Engineering 2009 (GSwE2009) Curriculum Guidelines for Graduate Degree Programs in Software Engineering. (2009).

Henning Heitkötter and TimA. and Majchrzak. 2013. Cross-Platform Development of Business Apps with MD2. In *Design Science at the Intersection of Physical and Virtual Design, Proceedings of the 8th International Conference on Design Science Research in Information Systems and Technologies (DESRIST) (Lecture Notes in Computer Science)*, Jan vom Brocke, Riitta Hekkala, Sudha Ram, and Matti Rossi (Eds.), Vol. 7939. Springer, Berlin Heidelberg, Germany, 405–411. DOI:http://dx.doi.org/10.1007/978-3-642-38827-9

Paul A Kirschner, John Sweller, and Richard E Clark. 2006. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist* 41, 2 (2006), 75–86.

Minhyuk Ko, Yong-Jin Seo, Bup-Ki Min, Seunghak Kuk, and Hyeon Soo Kim. 2012. Extending UML Meta-model for Android Application. In *Proceedings of the 2012 IEEE/ACIS 11th International Conference on Computer and Information Science (ICIS '12)*. IEEE Computer Society, Washington, DC, USA, 669–674.

A. Kraemer, Frank. 2011. Engineering Android Applications Based on UML Activities. In *Model Driven Engineering Languages and Systems, Proceedings of the 14th Conference on Model Driven Engineering Languages and Systems (MODELS) (Lecture Notes in Computer Science)*, Jon Whittle, Tony Clark, and Thomas Khne (Eds.), Vol. 6981. Springer, Berlin Heidelberg, Germany, 183–197.

Craig Larman. 2012. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, third edition* (third ed.). Pearson Education, Upper Saddle River, United States.

Zhen Li and Eileen Kraemer. 2013. Programming with Concurrency: Threads, Actors, and Coroutines. In *Proceedings of the 2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*. IEEE Computer Society, IEE, Washington, DC, USA, 1304–1311.

Tongping Liu, Charlie Curtsinger, and Emery D Berger. 2011. Dthreads: efficient deterministic multithreading. In *Proceedings of the Twenty-Third*

ACM Symposium on Operating Systems Principles. ACM, ACM Press, New York, NY, USA, 327–336.

Jeff Magee and Jeff Kramer. 2006. *State models and java programs* (second ed.). wiley, Chichester, England.

Alessandro Marchetto, Paolo Tonella, and Filippo Ricca. 2008. State-based testing of Ajax web applications. In *Proceedings of the 2008 1st International Conference on Software Testing, Verification, and Validation*. IEEE Computer Society, IEEE Computer Society, Washington, DC, USA, 121–130.

J.J.G. Van Merriënboer and P.A. Kirschner. 2001. Three worlds of instructional design: State of the art and future directions. *Instructional Science* 29, 4-5 (2001), 429–441.

M. David Merrill. 2002. First principles of instruction. *Educational technology research and development* 50, 3 (2002), 43–59.

Greg Nudelman. 2013. *Android Design Patterns: Interaction Design Solutions for Developers*. John Wiley & Sons, Indianapolis, United States.

OMG 2009. OMG Unified Modeling Language (OMG UML),Superstructure, version 2.2. http://www.omg.org/spec/UML/2.2/Superstructure/PDF/. (2009).

Abilio G. Parada and Lisane B. de Brisolara. 2012. A model driven approach for android applications development. In *Proceedings of the Conference on Computing System Engineering, Brazilian Symposium (SBESC)*. IEEE Computer Society, IEEE Computer Society, Washington, DC, USA, 192–197.

Derek Riley. 2012. Using mobile phone programming to teach Java and advanced programming to computer scientists. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education (SIGCSE '12)*. ACM, New York, NY, USA, 541–546.

Harald Störrle and JH Hausmann. 2004. semantics of uml 2.0 activities. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE Computer Society, Washington, DC, USA, 235–242.

Catherine Stringfellow and Divya Mule. 2013. Smartphone Applications As Software Engineering Projects. *Journal of Computing Sciences in Colleges* 28, 4 (April 2013), 27–34.

Shengqian Yang, Dacong Yan, and Atanas Rountev. 2013. Testing for poor responsiveness in Android applications. In *Proceedings of the 1st International Workshop on the Engineering of Mobile-Enabled Systems (MOBS)*. IEEE Computer Society, IEEE Computer Society, Washington, DC, USA, 1–6.

Received August 2014; accepted