# Model-Driven Development of Mobile Applications Allowing Role-Driven Variants*

Steffen Vaupel[1], Gabriele Taentzer[1], Jan Peer Harries[1], Raphael Stroh[1], René Gerlach[2], Michael Guckert[2]

[1] Philipps-Universität Marburg, Germany
{svaupel,taentzer,harries,strohraphael}@informatik.uni-marburg.de
[2] KITE - Kompetenzzentrum für Informationstechnologie, Technische Hochschule Mittelhessen, Germany
{rene.gerlach,michael.guckert}@mnd.thm.de

**Abstract.** Rapidly increasing numbers of applications and users make the development of mobile applications to one of the most promising fields in software engineering. Due to short time-to-market, differing platforms and fast emerging technologies, mobile application development faces typical challenges where model-driven development can help. We present a modeling language and an infrastructure for the model-driven development (MDD) of Android apps supporting the specification of different app variants according to user roles. For example, providing users may continuously configure and modify custom content with one app variant whereas end users are supposed to use provided content in their variant. Our approach allows a flexible app development on different abstraction levels: compact modeling of standard app elements, detailed modeling of individual elements, and separate provider models for specific custom needs. We demonstrate our MDD-approach at two apps: a phone book manager and a conference guide being configured by conference organizers for participants.

**Keywords:** model-driven development · mobile application · Android

## 1 Introduction

An infrastructure for model-driven development has a high potential for accelerating the development of software applications. While just modeling the application-specific data structures, processes and layouts, runnable software systems can be generated. Hence, MDD does not concentrate on technical details but lifts software development to a higher abstraction level. Moreover, the amount of standardization in code as well as in user interfaces is increased. A high quality MDD infrastructure can considerably reduce the time to market in consequence.

* This work was partially funded by LOEWE HA project no. 355/12-45 (State Offensive for the Development of Scientific and Economic Excellence).

Mobile application development faces several specific challenges that come on top of commonplace software production problems. Popular platforms differ widely in hardware and software architectural aspects and typically show short life and innovation cycles with considerable changes. Moreover, the market does not allow a strategy that restricts app supply to a single platform. Therefore multi-platform app development is a very time and cost-intensive necessity. It demands that apps have to be built more or less from scratch for each and every noteworthy target platform. Available solutions try to circumvent this problem by using web-based approaches, often struggling with restricted access to the technical equipment of the phone and making less efficient use of the device compared to native apps. Furthermore, web-based solutions require an app to stay on-line more or less permanently which may cause considerable costs and restricted usability.

Although there are already some approaches to model-driven development of mobile apps, our contribution differs considerably in design and purpose of the language. It allows a very flexible app design along the credo: "Model as abstract as possible and as concrete as needed." Data, behavior and user interfaces can be modeled on adequate abstraction levels meaning that behavior and UI design are modeled in more detail only if standard solutions are not used. Separating the model into an app model and one or several provider models, we achieve the possibility of a two stage generation and deployment process. While the app model defines the basic data structures, behavior and layout, these basic elements may be used in provider models to define specific custom needs. Hence, a provider model is an instance of the app model which in turn is an instance of the meta-model defining the overall modeling language. This approach suits very well to the kind of apps we consider here: While app models are developed by software developers, provider models are usually constructed by customers generally not being software experts. A typical example for such an app is a museum guide. Here, the app model contains information about objects, categories, events, and tours in museums in general. It specifies possible behavior like searching for museum objects, reading detailed information about them, and following tours. General page styles are also provided by that model. Customers may add one or more provider models containing information about objects being currently presented and additional categories to group objects semantically. Specific functionality also be added such as reading upcoming events of the next four weeks, reading details about a top object, and special page styles like one for the next exhibition. Changing a provider model does not lead to deploying the app anew. It only requires to make the modified instance model available. It integrates into the app model providing a refreshed application with up-to-date data and adapted functionality. Generated apps can work off-line without major restrictions.

The paper is structured as follows: In the next section, the kind of apps considered is presented. In particular, we explain the kind of mobile apps we consider. In Section 3, we present our language design and discuss it along typical design guidelines. Section 4 presents the developed MDD-infrastructure con-

sisting of several model editors and a code generator for the Android operating system. Section 5 reports on a case example. Finally, Sections 6 and 7 discuss related work and conclude this paper.

## 2 The mobile applications domain

Mobile apps are developed for very diverse purposes ranging from mere entertainment to serious business applications. We are heading towards a kind of business app where basic generic building blocks are provided for a selected domain. These building blocks can be used and refined by domain experts to customize them according to their specific needs. The fully customized app is then ready to be used by end users. Let's consider concrete scenarios as they occur in our collaboration with advenco, the industry partner of our project: *key2guide* is a multimedia guide that can be configured without programming. Its typical application lies in the context of tourism where visitors are guided through places of interest, e.g. a museum, an exhibition, a town or a region. Objects of interest (e.g. paintings, crafts and sculptures presented in a museum) are listed and explained by enriched information. Furthermore objects may be categorized and ordered in additional structures, i.e. tours that guide visitors through an exhibition. As the reader might expect, such an app is pretty data-oriented. This data usually changes frequently over time. In consequence, a typical requirement is to offer a possibility that domain experts (e.g. museum administrators or tourism managers) can refresh data regularly. Moreover, moving around in a region might lead to restricted Internet connections. Hence, web apps would not be preferred solutions. In contrast, apps shall typically run off-line but can download new provider information from time to time.

A second product by advenco, called *key2operate*, allows to define manual business processes with mobile device support to be integrated into a holistic production process. E.g. in order to inspect machines of a production plant, the worker gets a list of inspection requests that has to be executed sequentially. Such an execution might include the collection of critical data (e.g. pressure or temperature). Machines can be identified by scanning bar codes or reading RFID chips. Control values might be entered manually by the worker. Moreover start and end times of the execution may be taken. After finishing an inspection request, the app shall display the next request to be executed and direct the worker to the corresponding machine in line. Again, an app is required that may be configured by users being production managers here defining their intended business processes. As production processes have become very flexible nowadays, manual processes with mobile device support also have to be continuously adaptable. *key2operate* allows such process adaptations without newly deploying it. However, process definitions are pretty simple since they support simple data structures only. Both apps work with a web-based backend content management system to maintain configurations that are available for end users.

To summarize: We are heading towards model-driven development of mobile business apps that support the configuration of user-specific variants. In this

scenario, there are typically several kinds of users, e.g. *providers* who provide custom content, and *end users* consuming a configured app with all provided information. Of course, the groups of providers and end users may be structured more elaborately such that different roles are defined. For example, a tourist guide for a town may cover sights in the town as well as several museums and exhibitions. The guiding information is typically given by several providers with different roles. Tourism managers of the town are allowed to edit information about sights in the category town only, while e.g. administrators of the history museum may edit all the information about objects in their museum. Role-specific app variants shall be developed.

Throughout our project work, the mobile apps described above (*key2guide* and *key2operate*) were used as reference applications for the development of an MDD infrastructure. In the beginning, we analyzed and optimized these apps in order to approximate a best practice solution in prototypical re-implementations. Thereafter, we used them to test the developed infrastructure by modeling them and comparing the generated apps with the original ones. Due to space limitations, we have chosen a smaller example to be used as demonstration object throughout this paper. This example shows a number of important features of our approach:

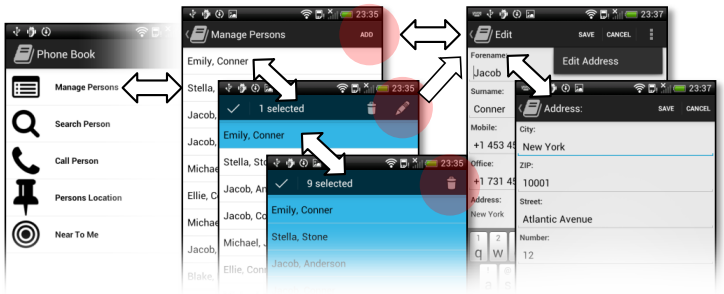
**Example** (A simple phone book app)**.** *One of the core apps for smart phones are phone books where contacts can be managed. In the following, we show a simple phone book app for adding, editing, and searching contact information about persons. Moreover, phone numbers are connected to the phone app such that selecting a phone number starts dialing it. Figure 1 shows selected screen shots of the phone app, already generated by our infrastructure. Little arrows indicate the order of views shown. In the following section, we discuss selected parts of the underlying model.*

## 3 Language design

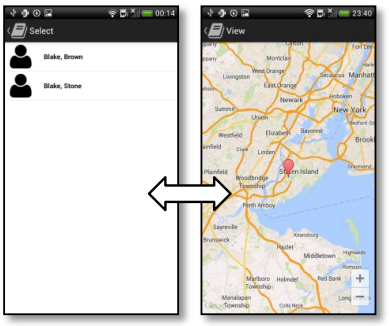
The core of an infrastructure for model-driven development is the modeling language. In the following, we first present the main design decisions that guided us to our modeling language for mobile applications. Thereafter, we present the defining meta-model including all main well-formedness rules restricting allowed model structures. To illustrate the language, we show selected parts of a simple phone book app model. Finally, the presented modeling language is discussed along design guidelines for domain-specific languages.

### 3.1 Design decisions

Due to our domain analysis, we want to support the generation of mobile apps that can be flexibly configured by providing users. This main requirement is reflected in our modeling approach by distinguishing two kinds of models: *app models* specifying all potential facilities of apps, and *provider models* defining




(a) Main Menu with *Manage Persons* Process (CRUD functionality)




(b) *Persons Location* Process


(c) *Call Person* Process

**Fig. 1.** Screen shots of phone book app

the actual apps. In Figure 2, this general modeling approach is illustrated. While app models are used to generate Android projects (1) being deployed afterwards (2), provider models are interpreted by generated Android apps (3), i.e., can be used without redeploying an app. Instance models can be carried out in two ways: usually this will be done at runtime, because the instance model does not exist at build time, alternatively it can be done at build time, by adding the instance model to the resources of the generated android projects.

The general approach to the modeling language is component-based: An app model consists of a *data model* defining the underlying class structure, a *GUI model* containing the definition of pages and style settings for the graphical user interface, and a *process model* which defines the behavior facilities of an app in form of processes and tasks. Data and GUI models are independent of each other, but the process model depends on them. A provider model contains an *object model* defining an object structure as instance of the class structure in the data model, a *style model* defining explicit styles and pages for individual graphical user interfaces, and a *process instance model* selecting interesting processes and providing them with actual arguments to specify the actual behavior of the intended app. Similarly to the app model, object and style models are independent of each other but used by the process instance model.

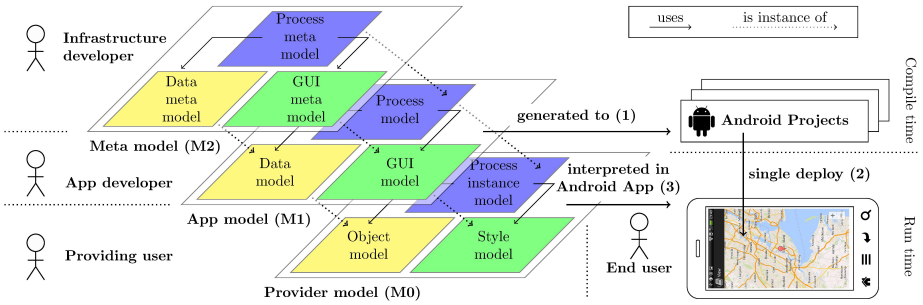



**Fig. 2.** Modeling approach

For the design of the modeling language, we follow the credo: *"As abstract as possible and as concrete as needed."* This means that standard design and behavior of an app can be modeled pretty abstractly. The more individual the design and behavior of the intended app shall be, the more details have to be given in the app model. Especially, all special styles, pages and processes that may be used in the intended app, have to be defined in the app model. Since the provider model shall be defined by app experts, they are already completely domain-specific and follow the pre-defined app model. Provider models support the development of software product lines in the sense that a set of common features are shared and some role-based variability is supported. Differences of considered apps are modeled separately by different provider models.

As far as possible, we reuse existing modeling languages which applies to the definition of data structures. Data modeling has become mature and is well supported by the Eclipse Modeling Framework (EMF)[22]. Hence, it is also used here to define the data model of an app. Specific information to the code generator (which is little up to now) is given by annotations.

The GUI model specifies views along their purposes as e.g. viewing and editing an object, searching objects from a list and showing them, doing a login and choosing a use case from a set. A GUI model is usually not intended to specify the inherent hierarchical structure of UI components as done in rich layout editors like the Interface Builder [20], Android Common XML Editor [12] and Android Studio [25]. However, the model can be gradually refined to obtain more specificity in the generated app. Style settings are specified independently of views and follow the same design idea, i.e. the more default look-and-feel is used, the more abstract the model can be.

Activities and services are modeled similarly along their purposes, i.e. different kinds of processes are available covering usual purposes such as CRUD functionality (create an object, read all objects, update or edit an object, delete an object) including searching, choosing processes as well as invoking GUI components, operations and processes. More specific purposes may be covered by the well-known concept of libraries, i.e. a basic language is extended by language components for different purposes as done for LabView [10].

To support the security and permission concepts of mobile platforms, the process model includes platform-independent permission levels. The permission concept is fine granular (i.e. on the level of single tasks), nevertheless some platforms like Android support only coarse granular permissions (i.e. on the level of applications). Another security-related feature is the user-specific instantiation of processes. Potentially, features of an application can be disabled by a restricted process instance model.

### 3.2 Language Definition

After having presented the main design decisions for our modeling language, we focus on its meta-model now. It is defined on the basis of EMF and consists of three separated Ecore models bundled in one resource set. While the data model is defined by the original Ecore model, two new Ecore models have been defined to model behavior and user interfaces of mobile apps.

Given a data model with Ecore, it is equipped with domain-specific semantics. Data models are not only used to generate the underlying object access but influence also the presentation of data at the user interface. For example, sub-objects lead to a tabbed presentation of objects, attribute names are shown as labels (if not overwritten) and attribute types define the appropriate kind of edit element being text fields, check boxes, spinners, etc. Furthermore, data models determine the behavior of pre-defined CRUD processes in the obvious way. Attribute names are not always well-suited to be viewed in the final app. For example, an attribute name has to be string without blanks and other separators while labels in app view may consist of several words, e.g. “Mobile number”. In such a case, an attribute may be annotated by the intended label.

The meta-model for user interface models is shown in Figure 3. Different views in user interfaces of mobile apps are modeled by different kinds of pages (View Page, Edit Page,...) each having a pre-defined structure of UI components and following a purpose. Only custom pages allow an individual structure of UI components (not further detailed here). The indicated ones are considered basic and may be accomplished with special-purpose ones in the future. The look-and-feel of a user interface is specified in style settings.

Figure 4 shows the meta-model for behavior models of mobile apps. This meta-model is influenced by the language design of BPMN [6] and (WS)-BPEL [5]. The main ingredients of a behavior model are processes which may be defined in a compositional way. Especially the composition of existing processes promises a scalable effort for process modeling. Each process has a name and a number of variables that may also function as parameters. A parameter is modeled as a variable with a global scope, contrary to locally scoped variables. The body of a process defines the actual behavior consisting of a set of tasks ordered by typical control structures and potentially equipped with permissions. There is a number of pre-defined tasks covering basic CRUD functionality on objects, control structures, the invocation of an external operation or an already defined process as well as the view of a page. While task CrudGui covers the whole CRUD functionality with corresponding views, Create, Read and Delete just cover single

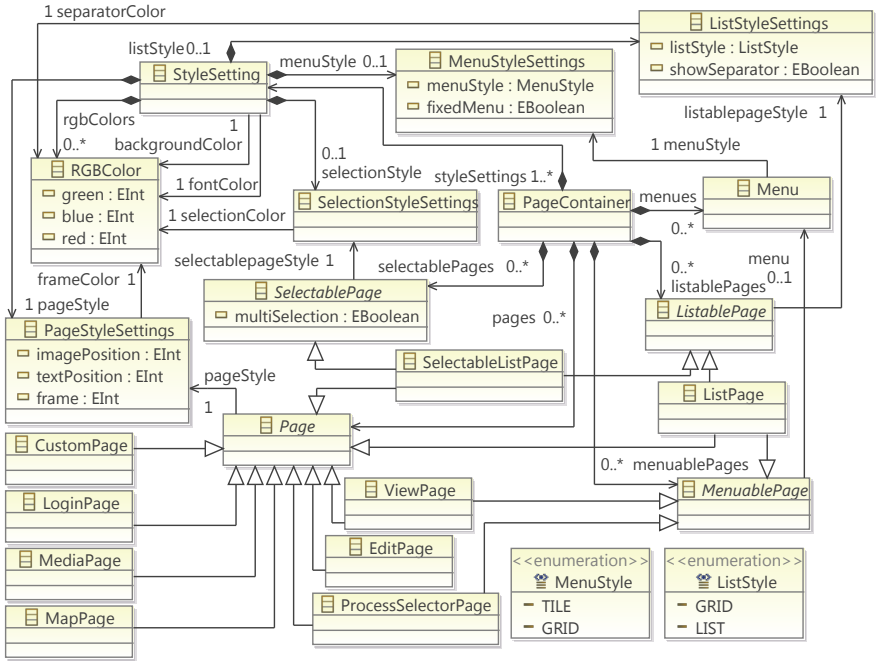



**Fig. 3.** Ecore model for defining graphical user interfaces of mobile apps

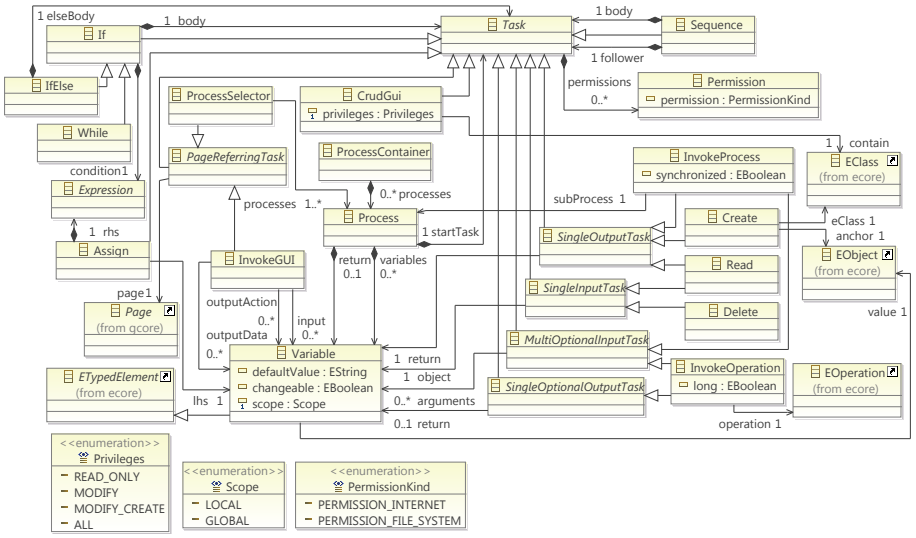



**Fig. 4.** Ecore model for defining mobile app behavior

internal CRUD functionalities. When invoking a process, the kind of invocation - synchronous or asynchronous - has to be specified.

Since all three meta-model parts are Ecore models, each model element can be annotated to cover additional generator-relevant information or just comments.

To get consistent app models, we need a number of well-formedness rules in addition. Especially the consistency between model components has to be taken into account. The main ones are listed below formulated in natural language. The complete list of rules formalized as OCL constraints can be found at [4].

1. There is exactly one process with name *Main*. This process is the first one to be executed.
2. There is at least one task of type *ProcessSelector* in the *Main* process.
3. A *Process* being registered in a *ProcessSelector*, contains - potentially transitively - at least one task of type *InvokeGUI* or *CrudGui*.
4. Considering task *InvokeGUI*, number, ordering and types of input and output data as well as output actions has to be consistent with the type of page invoked. E.g. a *MapPage* gets two Double values as output data, a *LoginPage* gets two strings as input to show the user name and password and a Boolean value as output data representing the result of a login trial.

**Example** (App model of simple phone book app)**.** *In the following, we present an instance model of the presented meta-model being the* app model *of the simple phone book app introduced in Section 2. We concentrate on selected model parts; the whole app model is presented at [4].*

*The simple data model is an Ecore model depicted in Figure 5. The structuring of contact data in a* Person *and* Address *seems to have advantages, since not too much information will be presented in one view.* PhoneBook *is just a container for* Person*s and not intended to be viewed.*

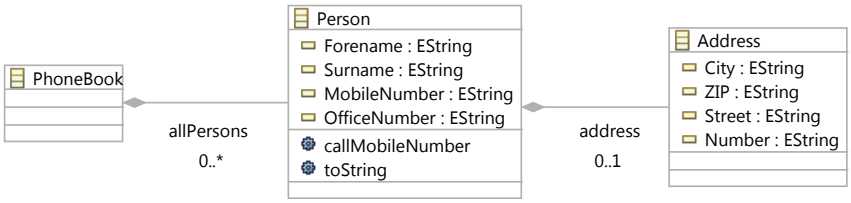



**Fig. 5.** Data model of simple phone book app

*Next, the user interface of our phone book app is modeled. This part of the app model is pretty simple, it just contains a default style setting, a default menu, and four pages, namely a* ProcessSelectorPage*, an* EditPage*, a* ViewPage*and a* SelectableListPage *for* Person *objects and a* MapPage *for* Address *objects. Note that we just add these pages to the model but do not specify their structures (see Figure 6).*

*The behavior of the phone book app is modeled by a process selector as main process that contains processes for all use cases provided. Figures 7(a) and 7(b)*

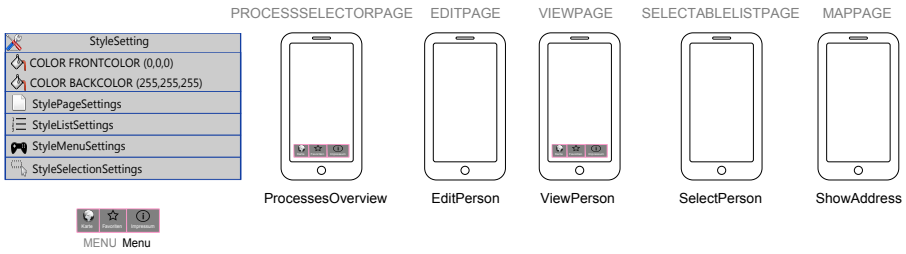



**Fig. 6.** User interface model of simple phone book app

*show processes* Main *being a process selector and* CRUDPerson *covering the whole CRUD functionality for contacts. Figure 7(c) shows the definition of a search process where first a search pattern is created that may be edited in an EditPage, then it is passed to a ReadProcess resulting in a list of persons being viewed in a SelectableListPage. If a person is selected from that list, its details are shown in a ViewPage. Figure 7(d) shows how to connect to the phone app to call a person. After searching for a person, operation* callMobileNumber *is invoked on the selected Person object. Just a few lines of code are needed to start the corresponding Android activity. I.e. the operation is implemented manually. At [4], process* NearToMe *is shown defining situation-dependent behavior in the sense that all persons of my phone book with an address near to my current position are displayed.*

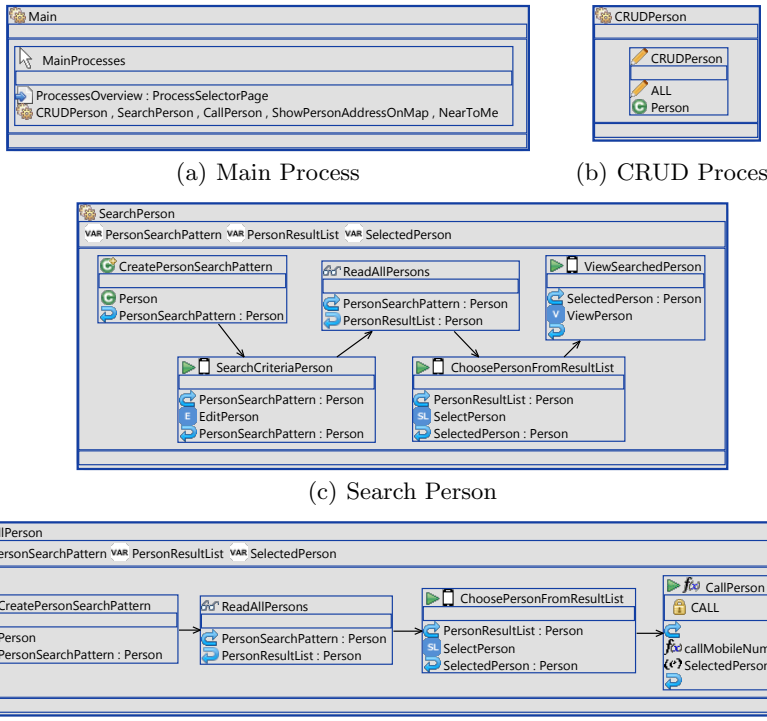
*An initial* provider model *just contains an empty phone book as object model and the main process as process instance model. The object model changes whenever the list of contacts is modified by the user.*

### 3.3 Discussion

After having presented the main features of our modeling language for mobile applications, we now discuss it along the design guidelines for domain-specific languages stated in [17]. The main purpose of our language is code generation. It shall be used mainly by software developers, perhaps together with domain experts and content providing users. The language is designed to be platform-independent, i.e. independent of Android or other mobile platforms.

A decision whether to use a textual or graphical concrete syntax does not have to be taken since we design the language with EMF and therefore, have the possibility to add a textual syntax with e.g. Xtext [9] or a graphical one with e.g. the Graphical Modeling Framework (GMF) [13, 21]. Currently, a graphical editor is provided as presented in the next section. The development of a textual one is less work and shall be added in the near future. We decided to reuse EMF for data modeling since it is very mature. Since we define our language with EMF, the Ecore meta-model can also be reused, together with its type system.

Next, we discuss the choice of language elements. Since all generated mobile apps shall share the same architecture design (being detailed in the next sec-




(a) Main Process (b) CRUD Process

(c) Search Person

(d) Call Person

**Fig. 7.** Process model of simple phone book app

tion), the modeling language does not need to reflect the architecture. However, data structures, behavior and user interface design are covered. Since we want to raise the abstraction level of the modeling language as high as possible, we have discussed each specific feature of mobile apps carefully to decide if it can be set automatically by the generator or if the modeler should care about it. For example, asynchronous execution of an operation is decided indirectly if the operation is classified as long-lasting but can also be set directly. Permissions are completely in the hand of the modeler since they depend on the operations executed. The authors of [17] emphasize the simplicity of a language to be useful. Our language follows this guideline by avoiding unnecessary elements and conceptual redundancy, having a very limited number of elements in the core language and avoiding elements that lead to inefficient code.

The concrete syntax has to be well chosen: For data modeling, we adopt the usual notion of class diagrams since it has proven to be very useful. Process models adopt the activity modeling style to define control structures on tasks since well-structured activity diagrams map usual control structures very well. Notations for pages and tasks use typical forms and icons to increase their descriptiveness and make them easily distinguishable. Models are organized in

three separate sub-models wrt. different system aspects, i.e. data model, process model and GUI model. Moreover, data structures can be organized in packages and processes can be structured hierarchically. However, processes and pages cannot be packaged yet. Not many usage conventions have been fixed up to now (except of some naming conventions) but will be considered in the future.

There is especially one part where the abstract and the concrete syntax of our language diverge, the definition of control structures for task execution. While the concrete syntax follows the notion of activity diagrams, the abstract syntax contains binary or ternary operations such as if clauses and while loops. This allows an easier handling of operations for code generation, however, they are unhandy during the modeling process. There are no places where the chosen layout has any affect on the translation to abstract syntax. Our language provides the usual modularity and interface concepts known from other languages: Packages and interface classes in data models as well as processes and process invocations in behavior models.

## 4 MDD-infrastructure for mobile applications

Infrastructures for model-driven software development mainly consist of editors and code generators. In the following, we present an MDD-infrastructure for mobile applications as a prototypical implementation of the presented modeling language, together with a multi-view graphical editor and a code generator to Android. Another code generator to iOS will come soon. While the language itself is based on EMF, the graphical editor is based on GMF [13]. Both code generators are written in Xtend [9]. The editor and the code generator are designed as separate Eclipse plug-ins. They use the common implementation of the abstract language syntax including model validation, captured again in plug-ins.

*Graphical editor for app models.* The graphical editor for app models is designed as a graphical editor consisting of three different views for data modeling, process modeling and GUI modeling. The existing Ecore diagram editor has been integrated for data modeling. Figures 6 and 7 show screen shots of depicted processes and pages. As expected, changes in one view are immediately propagated to the other ones accordingly.

While the concrete syntax of control structures for task execution follows the notion of activity diagrams, the abstract syntax contains binary or ternary operations such as if clauses and while loops instead. This diversion between the abstract and the concrete syntax of our language cannot be covered directly by mapping concrete model elements to abstract ones. Therefore, a slight extension of the modeling language has been defined and is handled by the editor, i.e. concrete models are mapped to extended abstract models that are translated to non-extended ones by a simple model transformation. By application of the well-known generation gap pattern [23, p.85-101], the standard presentation of GMF-based editors has been adapted to special needs such as special labels and icons.

*Code generation to Android.* Having edited an app model, it has to be validated before code generation since the code generator is designed for correct models only. The code generator produces two projects: an Android project containing all the modeled activities, and an Android library project. Mobile apps shall be generated that can be flexibly configured by content providing users, of course without redeploying these apps. To realize this requirement, an Android library project is generated being based on EMF. This library project is able to interpret configurations written by providers. It is used by the main Android project. (See Figure 8.) The main Android project follows the usual model-view-controller architecture of Android apps. Packages **model** and **crud** form the data access layer with the usual CRUD functionality, while package **gui** contains controllers in form of activities, fragments, adapters, and services. Additionally, view components are generated as app resources.

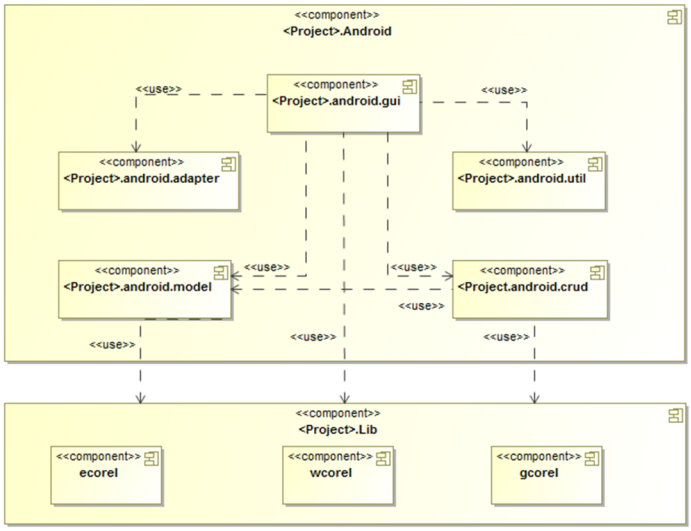



**Fig. 8.** Architecture of generated apps

All these projects are usually immediately compiled and then, ready to start. By default, the SD card of the mobile device contains an initial provider model consisting of an empty object model, i.e., without any data, and an initial process instance model containing the main process with all those processes assigned to the main process by the app model. This provider model can be extended during run time. After regeneration, it might become partly invalid, dependent on the kind of app model changes. If, e.g., the process model has changed but the data model has not, the object model is still readable, but the process instance model is not. It is up to future work, to support automated migration of provider models.

## 5 Case study

Our major case study is a guide app for conferences being configured to guide participants through conferences like Models 2014. Depending on the user's role, i.e. provider or consumer, two different provider models are used leading to two different app variants: one for conference organizers with the full range of CRUD processes and one for participants with read and search processes only. Participants use the app as a conference guide with the look-and-feel of a native app. Besides searching for information, they may select sessions and add them to a list of favorites. These sessions may be transferred to the selected Android calendar so that reminders can be set. At [4], the interested reader can find additional information about how the guide app is modeled and how data can be entered. It shows a typical application of our two stage app development by showing how to generate an app and how to specialize it to variants for different custom needs. For a conference, quite some data has to be provided which is usually tedious using a mobile device. Using an Android HDMI stick, the app can be presented on an external screen and input can be given via an external keyboard which leads to a very convenient way of editing provider models directly by an app, suitable for providing larger amounts of data. The models of this guide app as well as of the phone book app (used as running example) are pretty small (less than 100 model elements) while the generated code is comprehensive (thousands of code lines). Hence, one can see that the abstraction level of development is raised considerably.

## 6 Related work

The model-driven development of mobile applications is an innovative subject which has not been tackled much in the literature. Nevertheless, there are already some approaches which we compare to ours in the following.

$MD^2$ [14] is an approach to cross-platform model-driven development [7] of mobile applications. As in our approach, purely native apps for Android and iOS shall be generated. However, the domain of data-driven business apps, differs from ours: While $MD^2$-generated apps are based on a kind of app model only, our approach offers provider models in addition. Moreover, the underlying modeling languages differ in various aspects: The view specification by $MD^2$ is structure-oriented and pretty detailed, i.e. views are specified on an abstraction layer similar to UI editors. In contrast, the gcore language of our approach is purpose-oriented and thus, lifted to a higher abstraction level. $MD^2$-controller specifications show some similarities and some differences to our process specification. Similarly to our approach, action types for CRUD operations are provided. But it is not clear how additional operations (different from CRUD functionality) can be invoked, as e.g. starting a phone call by selecting a phone number. The generated Android apps follow the MVC-architecture pattern as well. While the data model is translated to plain (old) Java objects (POJOs) in $MD^2$ with serialization facilities for server communication as well as a JEE application to be run on a server, our approach also supports off-line execution.

Two further MDD approaches focusing on data-centric apps are *applause* [8, 2] and *ModAgile* [3]. Both support cross-platform development for mainly Android and iOS. In contrast to our approach, behavior is nearly not modeled and user interfaces are modeled rather fine-grained.

Another kind of development tools for Android apps are event-driven approaches such as App Inventor [1] providing a kind of graphical programming language based on building blocks and Arctis [18] being based on activity diagrams. Both approaches focus on rather fine-grained behavior and/or UI specification and largely neglect the modeling of data structures.

Besides the generation of native apps, there are several approaches to the model-driven development of mobile Web apps being originated in the generation of Web applications. Although Web apps show platform independence by running in a Web environment, they have to face some limitations wrt. device-specific features, due to the use of HTML5 [19, 24]. There are several approaches to MDD of Web apps, such as *mobl* [16, 15] and a WebML-based solution by WebRatio [11]. Since we are heading towards apps being most of the time off-line as demanded by the domain considered, Web apps are not well-suited.

Our approach supports the model-driven development of native apps by high-level modeling of data structures, behavior and user interfaces. In addition, the role-based configuration of app variants is supported.

## 7 Conclusion

Model-driven development of mobile apps is a promising approach to face fast emerging technology development for several mobile platforms as well as short time-to-market with support for several if not all existing platforms. In this paper, a modeling language for mobile applications is presented that allows to model mobile apps as abstract as possible and as concrete as needed. Different user roles are not combined in one app but lead to several app variants that may be configured after code generation, i.e. by content providing users, for end users. The considered domain are business apps being data or event-driven such as tourist and conference guides as well as manual sub-processes in production processes. A selection of example apps being developed with our MDD-tool environment, can be found at [4]. Future work shall cover further platforms, a code generator to iOS is currently under development, and language extensions towards flexible sensor handling and augmented reality. Moreover, generated apps shall be evaluated wrt. software quality criteria, especially usability, data management, energy efficiency and security.

## References


1. App Inventor. `http://appinventor.mit.edu`
2. Applause. `https://github.com/applause/applause`
3. ModAgile. `http://www.modagile-mobile.de`

4. PIMAR: Model-driven development of mobile apps. http://www.uni-marburg.de/fb12/swt/forschung/software/pimar/
5. Web Services Business Process Execution Language (WS-BPEL) Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (2007)
6. Business Process Model And Notation (BPMN) Version 2.0. http://www.omg.org/spec/BPMN/2.0 (Januar 2011)
7. Allen, S., Graupera, V., Lundrigan, L.: Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution. Apresspod Series, Apress (2010), http://books.google.de/books?id=JKpKrwtoWNAC
8. Behrens, H.: MDSD for the iPhone: developing a domain-specific language and IDE tooling to produce real world applications for mobile devices. In: Cook, W.R., Clarke, S., Rinard, M.C. (eds.) SPLASH/OOPSLA Companion. pp. 123–128. ACM (2010)
9. Bettini, L.: Implementing Domain-Specific Languages with Xtext and Xtend. Packt Publishing Ltd. (2013)
10. Bishop, R.: Learning with LabVIEW. Pearson Education (2011)
11. Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks 33(1-6), 137–157 (2000)
12. Goadrich, M.H., Rogers, M.P.: Smart smartphone development: iOS versus Android. In: Proceedings of the 42nd ACM technical symposium on Computer science education. pp. 607–612. SIGCSE '11, ACM, New York, NY, USA (2011)
13. Gronback, R.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Eclipse Series, Pearson Education (2009), http://books.google.de/books?id=8CrCXVZXLjcC
14. Heitkötter, H., Majchrzak, T.A., Kuchen, H.: Cross-Platform Model-Driven Development of Mobile Applications with $md^2$. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC '13, Coimbra, Portugal, March 18-22, 2013. pp. 526–533. ACM (2013)
15. Hemel, Z., Visser, E.: Declaratively programming the mobile web with Mobl. In: Lopes, C.V., Fisher, K. (eds.) OOPSLA. pp. 695–712. ACM (2011)
16. Hemel, Z., Visser, E.: Mobl: the new language of the mobile web. In: Lopes, C.V., Fisher, K. (eds.) OOPSLA Companion. pp. 23–24. ACM (2011)
17. Karsai, G., Krahn, H., Pinkernell, C., Rumpe, B., Schneider, M., Völkel, S.: Design Guidelines for Domain Specific Languages. In: Rossi, M., Sprinkle, J., Gray, J., Tolvanen, J.P. (eds.) Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling (DSM'09). pp. 7–13 (2009)
18. Kraemer, F.A.: Engineering Android Applications Based on UML Activities. In: Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6981, pp. 183–197. Springer (2011)
19. Oehlman, D., Blanc, S.: Pro Android Web Apps: Develop for Android using HTML5, CSS3 & JavaScript. Apresspod Series, Apress (2011), http://books.google.de/books?id=pZlF7lQY5SQC
20. Piper, I.: Learn Xcode Tools for Mac OS X and iPhone Development. IT Pro, Apress (2010)
21. Rubel, D., Wren, J., Clayberg, E.: The Eclipse Graphical Editing Framework (GEF). Eclipse (Addison-Wesley), Addison-Wesley (2011), http://books.google.de/books?id=GiKTAR9M-L4C

22. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley, Boston, MA, 2 edn. (2009)
23. Vlissides, J.: Pattern hatching: design patterns applied. The software patterns series, Addison-Wesley (1998), http://books.google.de/books?id=4qRQAAAAMAAJ
24. Williams, G.: Learn HTML5 and JavaScript for Android. ITPro collection, Apress (2012), http://books.google.de/books?id=PRlytmflmhoC
25. Zapata, B.: Android Studio Application Development. Packt Publishing (2013)