Model-Driven Development of Mobile Applications: Towards Context-Aware Apps of High Quality *

Gabriele Taentzer, Steffen Vaupel

Philipps-Universität Marburg, Germany
{taentzer,svaupel}@informatik.uni-marburg.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 (MDD) can help.
An infrastructure for MDD has a high potential for accelerating the development of software applications. While just modeling the application-specific data structures, processes and user interfaces, runnable apps can be generated for multiple platforms such as Android and iOS. Hence, MDD can lift software development to a higher abstraction level leaving all technical details to the generators. Moreover, the domain-specific model of a mobile application can be the starting point for validation and verification based on formal models.
In this paper, we consider the state-of-the-art in MDD of mobile applications and consider future research directions such as the MDD support for higher mobility in changing contexts, a combination of sensors, and augmented reality. Moreover, we discuss quality assurance of mobile applications.

1 Introduction

The use of mobile applications has become an indispensable part of daily life. This has already and will lead to rapidly increasing numbers of applications and users that make the development of mobile applications to one of the most promising fields in software engineering. Mobile application development faces several specific *challenges* that come on top of commonplace software production problems. Popular platforms differ widely in hardware and software characteristics and typically show short life and innovation cycles with considerable changes. The market often requires that apps must be available for several platforms which makes a very time and cost-intensive multiple platform development a necessity. Available solutions try to circumvent this problem by using

* This work was partially funded by the German Research Foundation, project "Distributed Model-Driven Software Development".

web-based approaches, often struggling with restricted access to the technical equipment (e.g. sensors) of the mobile device 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 usability restrictions. Model-driven development (MDD) can help to improve this situation by a faster and easier development process of native apps and easier adaptability to new features of underlying platforms. Mobile apps are modeled in a modeling language focusing on main system aspects being enough to automatically generate platform-specific code. Hence, MDD allows to develop applications on a higher abstraction level, neglecting technical details. Furthermore, code quality may improve since code can be generated according to standards. In addition, the validation of app models w.r.t. system requirements may be facilitated by deducing formal models on a similar abstraction level.

The heart and soul of model-driven development is the *domain-specific modeling language* (DSML). Our developed DSML covers three aspects of mobile apps: its data entities and relations, its behavior including data management, access of sensors, use of other apps, etc., and its user interface. The design of our modeling language follows the credo: *"Model as abstract as possible and as concrete as needed."* This means the following: Standard solutions are modeled very abstractly while more specific solutions are modeled in more details. E.g. data management by the usual CRUD functionality (using create, read, update, and delete operations) may be modeled by one predefined model element type while application-specific behavior is specified on the level of usual control structures.

To efficiently work with the DSML, we provide an *Eclipse-based tool environment* consisting of a graphical editor with three different views for data, behavior and user interface models as well as two code generators to Android [13, 22, 6] and iOS [15, 5]. Before developing these two code generators, we studied the design principles of mobile apps in Android and iOS and found a lot of commonalities. The idea behind this work is to develop apps for different platforms as similar as adequate. It is possible to use the same overall architecture independent of the chosen platform. We demonstrate the potentials and limits of our MDD approach at a *small example*.

Considering our approach to MDD of mobile apps, the following issues are still open: (1) Due to spatial movement, mobile apps offer new possibilities to interact with the environment: They support an increasing variety of sensors such as cameras, global positioning system (GPS), compass, etc. which can be advantageously used to position a mobile device and to inform the user about its position. These capabilities are especially interesting for augmented reality (AR) and navigation in space. (2) Mobile applications claim to operate reliably during spatial movement, however, developers have to deal with the effects of changing environmental contexts. One of the most important contexts is the connectivity of mobile devices. Since mobile applications are increasingly used as front-ends of transaction systems, they have to be designed for being able to deal with intentional or accidental loss of connection. In order to support higher mobility - in the sense that operations may execute across the boundaries of

changing network states - problems and requirements for context-aware architectures of mobile applications are considered. We discuss how these issues may be supported by model-driven development as well.

A faster and easier development of mobile apps is nothing if their quality cannot be assured. We discuss what software quality means for mobile apps and what kinds of quality assurance techniques may be promising.

2 Modeling Mobile Applications

The core of an MDD infrastructure is the domain-specific modeling language. It is used to model the specific aspects of applications in a so-called *app model*. This model is platform-independent and the input to available code generators, here, Android and iOS. The generator results are platform-specific software projects containing runnable apps. For an overview see Figure 1.

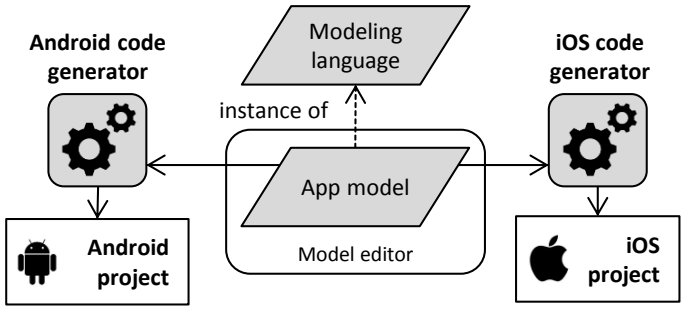


Fig. 1. Model-driven development of mobile apps

An app model consists of three sub-models: the *data model*, the *process model* to describe the app behavior, and the *graphical user interface (GUI) model*. In the following, we focus on the main language features. A detailed language definition can be found in [28].

The *data model* contains the typical elements of object-oriented data modeling like classes, attributes, aggregations, associations, etc. It is not only used to generate the underlying data access objects (DAOs) but also determines the design of the user interface that is concerned with data input and output.

The *GUI model* defines the graphical user interface of an app. It contains pages, style settings, and menus. The page type indicates the purpose of the page (e.g. to select a processor or to edit and view data). Thus, app modelers describe the required user interface just by selecting a page type (e.g., EditPage or ViewPage) and the code generator deduces the detailed structure of the interface (e.g. labels, edit and text fields, check boxes, etc.) by consulting the data model.

The *process model* defines the behavior of an app. This model part is structured in processes, including various types of sequentially performed tasks. A number of tasks are pre-defined such as tasks to create, read, update and delete data objects. The process model may refer to the data types of the data model and invoke pages of the GUI model; hence, it uses the other two model parts.

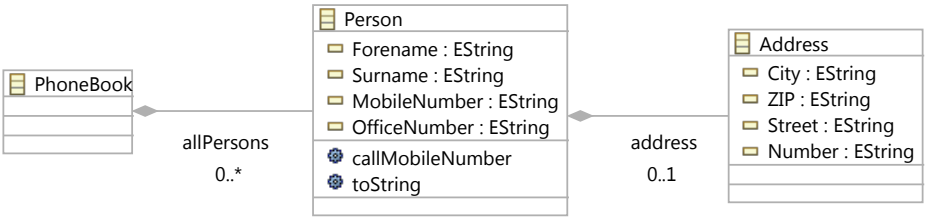


Fig. 2. Data model of simple phone book app

In the following, we consider the app model of a simple phone book app. This app shall support the management of personal contacts as well as the search for phone numbers and the set up of a phone call. A simple *data model* is depicted in Figure 2. Contact data is structured in classes Person and Address. The class PhoneBook is just a container for Persons and not intended to be viewed.

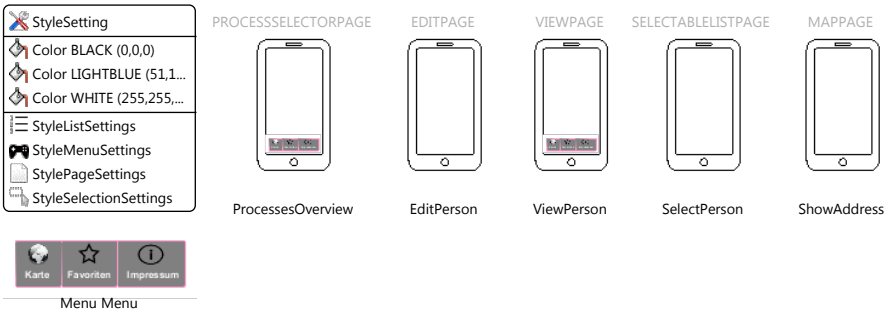
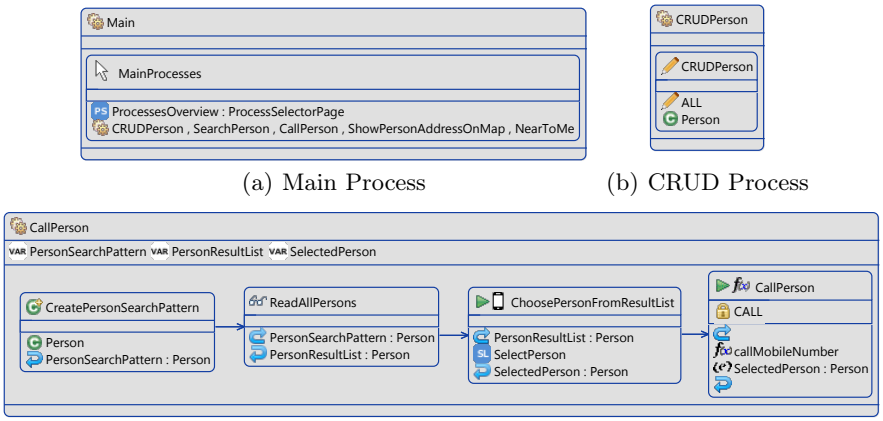


Fig. 3. User interface model of simple phone book app

The modeled user interface of our phone book app is shown in Figure 3. This part of the app model is pretty simple, it just contains a style setting, a menu, and five pages, namely a ProcessSelectorPage, an EditPage, a ViewPage, a SelectableListPage for Person objects, and a MapPage for Address objects. Note that we just add these pages to the model and use them to specify behavior but do not specify their layout.



(a) Main Process (b) CRUD Process

(c) Call Person Process

Fig. 4. Process model of simple phone book app

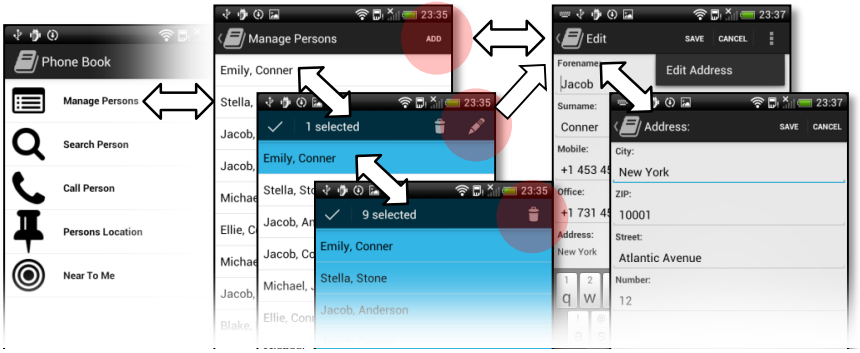
The behavior of the phone book app is modeled by a process selector as main process that contains processes for all use cases provided. Figures 4(a) and 4(b) show processes Main being a process selector and CRUDPerson covering the whole CRUD functionality for contacts. Figure 4(c) shows how to connect to a 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 (not shown) are needed to start the corresponding Android activity or iOS service. I.e. the operation is implemented manually.

Figure 5 shows selected screen shots of the phone book app, already generated by our Android generator. Little arrows indicate the order of views shown. The first sub-figure (Figure 5(a)) shows the main menu containing a standard CRUD process (*Manage Person*) to create, edit, and delete persons. The standard behavior and user interface for this task have been generated from a simple CRUD process (see Figure 4(b)) which the app modeler has created before. Moreover, the app considers the user location context to find contacts near the current location of the user (Figure 5(b)). Phone numbers are connected with an installed phone app in such a manner that whenever the user has selected a phone number, it automatically starts dialing (Figure 5(c)).

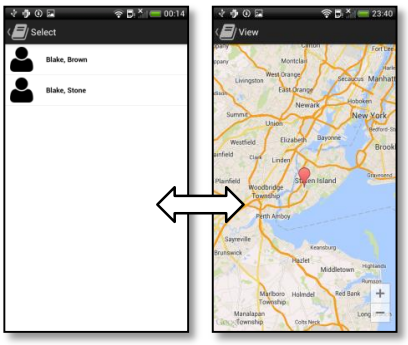
3 Tool Support

The modeling language is defined on the basis of the *Eclipse Modeling Framework* (EMF) [26]; therefore, model editors may be textual (e.g. Xtext [12]) or graphical (e.g. Graphical Modeling Framework (GMF) [16, 25]). A GMF-based graphical editor is currently provided.

Two code generators are available, one for Android [13, 22, 6] and one for iOS [15, 5]. Both code generators can process the same platform-independent app



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



(b) *Persons Location* Process

(c) *Call Person* Process

Fig. 5. Screen shots of phone book app

model and generate apps running directly on Android and iOS devices. They are written with Xtend [12] in a template-based style.

All the generated mobile apps have the following overall, platform-independent architecture: The architecture of generated mobile apps reflects the separation of data, process, and GUI aspects in app models leading to a model-view-controller (MVC) architecture. Since we generate data-oriented mobile applications, the architecture of each generated mobile app has a data layer. This layer contains the modeled data entities (e.g., persons and addresses) and provides functionality to serialize and deserialize these objects. The data layer forms the model of the application. The controller layer implements the behavior specified by the process model, i.e., it holds the application logic. It is the intermediary layer between the model and the view. The controller invokes the interactive user dialogs and processes events returning from user dialogs. The view layer provides all dialogs. This layer is most platform-specific because it uses the graphical components provided by the envisioned platform. These dialogs do not contain any calculations and navigation logics (except of input validation), and return all events to

the controller. Finally, the architecture of a generated mobile app implements a transaction concept. A process invoked by the initial process selector dialog opens a transaction based on the data layer. If the user returns to this initial process selector dialog by confirming all steps, this transaction is committed. Otherwise, the user can cancel the transaction at any passing dialog, and the changes made are consequently lost.

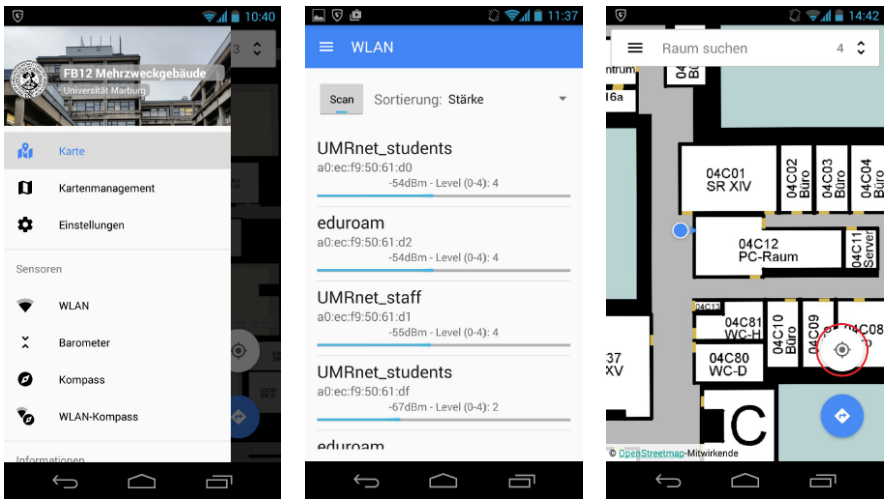
4 Modeling Contextual Aspects

Mobile applications change contexts as no other type of application does. A context can be any information that can be used to characterize the situation of a person, place, or object that is considered relevant for the interaction between a user and an application. An app is context-aware if it uses the context to provide relevant information and/or services to the user [8]. Besides more traditional contexts like user contexts (e.g. roles concept) and platform environments (e.g. display size) [28] being already supported by our approach, we will focus on the use of context information for navigation and travel now. Apart from this, we point out the dangers of changing contexts.

4.1 Using context information for navigation and travel

Most of the apps for navigation and travel use the GPS to localize a mobile device. For example, a public transport app can dynamically list the nearest bus stop from which a bus will leave. Such a context-aware app uses the GPS position and the time of the request for calculating the next reachable connection and departure point. Indoor navigation is more challenging, because GPS is usually not available inside a building. Hence, we developed a prototype app (cf. screen shots in Figure 6) that combines different sensors (Barometer, Compass, WLAN Access Points, Camera & QR-Codes) to get a more accurate positioning in buildings. Additionally, the indoor navigation app provides different methods of sensor processing (trilateration, distance comparison, field intensity, weighted average) showing advantages/disadvantages in accordance with the building-specific installation of access points.

Another application of combined sensor information is related to augmented reality (AR): AR uses the built-in camera to recognize objects and to augment the live view with different types of virtual objects. This is usually very processor-intensive and mobile devices often have limited processor capacity. Hence, the recognition and augmentation of many objects being close together is not feasible since the device cannot compare all trained patterns with the current view captured from the camera in an acceptable time. In order to circumvent this problem, the number of patterns can be reduced through an accurate positioning of the device. For example, an AR application is used to augment exhibits in a museum. The accurate positioning of the device inside the building (floor and room) can reduce the amount of data that must be processed (location-based filtering). This approach was tested in [10].



(a) Main Menu (b) Sensor data (c) Map view and position

Fig. 6. Screen shots of indoor navigation app

This way of mobility-related interaction with the environment is not yet fully capable in the DSML. It is intended to use the realized prototypes for extending the DSML as well as the code generators in an agile way [27].

Modeling context-aware apps Although there are no pre-defined modeling elements at the moment to model context-awareness, it is possible to model a context-aware app with the standard modeling elements. Our DSML already provides full access to the device sensors. Focusing on data-driven apps, we can demonstrate the location-based filtering of data that is already covered by our MDD approach. The phone book app contains a Process Near to me. This process uses the location context of the device (user) and filters the addresses of contacts accordingly. Only contacts within a given radius appears on the list. Thus, the list of contacts depends on the location of the device.

4.2 Dangers of changing contexts

Mobility and changes of contexts caused thereby can also be a danger. The changing context, especially the connection context, usually makes mobile applications with standard architectures non-operable in off-line situations. A look at real-world applications gives evidence that application developers hardly realize connection-aware mobile applications. None of the available apps for on-line shopping supports transactions (such as ordering of products) in a disconnected mode although a sudden stop in the middle of an ordering process may prevent potential customers from eventually ordering selected products. To construct mobile applications that are on-line and off-line capable, the applications have

to have a context-aware architecture supporting additional architectural components like replication, synchronization and local transaction managers.

Providing a context-aware app architecture Processes that use connecting operations are, in general discarded in an off-line context. These may be used anyway if the conflict-free reintegration of locally modified data can be guaranteed. Certain mobile transaction models can coordinate isolated changes of local copies. Two of them are especially interesting: The Keypool transaction model [1, 2] uses the independence of the individual instances of a class. The basic idea of the Keypool method is to split the entire dataset into subsets that are distributed among the participating mobile clients. Operations do not have to be restricted. The Escrow transaction model [24, 21] is suitable for accessing and modifying aggregate data. The basic idea of the Escrow method is to restrict the set of transactions and/or the domains of their arguments for off-line transactions. Restrictions have to be set carefully to guarantee that the participating transaction is conflict-free on the one hand and to optimize the goodput (i.e., the number of successfully executed operations) on the other.

Usually, developers have to acquire a lot of knowledge about mobile transaction processing before they are able to design on-line and off-line capable applications. Our basic approach is the provision of code generators that can create mobile applications with such a generic context-aware app architecture [29]. Modelers add only a few parameters to the app model (e.g. which processes are conflicting, which mobile transaction model should be used, etc.) and the code generators create a mobile app with corresponding replication, off-line transaction management, and (conflict-free) synchronization.

5 Quality Assurance of Mobile Applications

As for any kind of software system, the quality notion for mobile apps covers various quality aspects such as correct functionality, reliability, security, usability, maintainability, just mention a few. Since correctness and reliability are usually the most important ones, we concentrate on those in the following.

In model-driven software development, software quality is extremely dependent of the platform-independent model since the code generator takes it directly as input. Hence, the generated code is not considered directly but the input model is analyzed instead for discovering potential problems. Moreover, the actual app behavior can be analyzed during runtime as possible for apps being classically developed.

5.1 Quality of app models

To analyze the quality of app models, syntax checks such as smells or translations to formal models are possible approaches. The second is especially interesting to verify semantical properties. Moreover, an app model can be used to generate

skeleton code for test cases, not necessarily for the generated application code but for separately implemented, i.e., non-generated code.

Smells represent modeling knowledge being condensed from developer experiences; they point to model parts that are suspicious w.r.t. quality. This means that modelers should have a look at smelling model parts and should decide which have to be improved. Model smells may produce code smells, i.e. generated code parts do not have high quality. A model smell in an app model may point to, e.g., processes with too many tasks leading to loosely structured services.

The translation of an app model to a formal model of choice can help to find potential reliability problems. Interesting questions are for example: Which processes are potentially in conflict? Which objects are potentially accessed concurrently? Can the modeled processes run into deadlocks? To get a meaningful formal model, app models may be preprocessed replacing standard processes such as the CRUD process by more detailed processes showing process behavior on a homogeneous abstraction level.

5.2 Quality assurance at run time

App behavior cannot only be analyzed at development time but also at run time. There are two main approaches: To test individual operations implemented by hand, skeleton code for test cases may be generated. Another way to analyze the run time behavior is to log relevant information and to process it further.

App developers do not have to test all the code of the generated app. Since a code generator is reused again and again, it should result in a mature tool that ensures reliable code generation. If the intended app shows individual behavior, however, this behavior is not generated but has to be implemented manually and integrated into the generated code. These manually written code parts are encapsulated in operations which should be tested for their correctness. Based on their signatures and their embedding into generated code, it is possible to generate skeleton code for testing them in unit tests. The skeleton code sets up a test suite and has to be completed by semantical information about operations under test.

To find out how generated behavior interacts with individually specified behavior, it may be worthwhile to analyze the run time behavior of an app. This is also useful to get realistic information about, e.g., execution time, energy efficiency, and net usage of the app. As a basis, we could log relevant information in a log file. This may comprise information about active services, called operations with actual arguments, and much more. Condensing the log information in a log model would lead to a basis for further analyses. As proposed for the app model, the log model could be translated to a formal model of choice to use suitable analysis techniques. Interesting questions to answer are for example: How many transactions finished successfully? What are the execution times for performed transactions?

6 Related Work

The model-driven development of mobile applications is an innovative field 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 [17] is an approach to cross-platform model-driven development [9] of mobile applications. As in our approach, purely native apps for Android and iOS are generated. That approach focuses on the domain of data-driven business apps. The underlying modeling languages of that approach and ours 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 GUI language of our approach is purpose-oriented and thus, lifted to a higher abstraction level. The generated mobile 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 Java Enterprise Edition (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* [11, 4] and *ModAgile* [7]. 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 is the event-driven approach App Inventor [3] providing a kind of graphical programming language based on building blocks. Building blocks are simple components with graphical UI and input/output facilities. The development is mainly concerned with the app behavior. The language does not support app development on a higher abstraction level. E.g. pure CRUD functionality has to be designed step-by-step with every detail while our approach can model CRUD functionality directly for a data model. A further event-driven approach is Arctis [20] being based on activity diagrams. Like App Inventor, Arctis also focuses on rather fine-grained behavior and/or UI specification and largely neglects 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 [23, 30]. There are several approaches to MDD of Web apps, such as *mobl* [19, 18] and a WebML-based solution by WebRatio [14]. Since we are heading towards off-line capable apps being off-line, Web apps are not well-suited. Apps generated with our infrastructure can work on-line and off-line.

To summarize, our approach supports the model-driven development of native apps by high-level modeling of data structures, behavior and user interfaces while supporting the role-based configuration of app variants and in thus differs considerably from approaches aiming at comparable goals.

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 noteworthy 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. Two code generators are available to produce Android and iOS apps. The considered domain are data-oriented apps that may be enriched by specific behavior elements such as interaction with sensors and other apps as well as entertainment elements as, e.g, little games. Example apps are, e.g., tourist and conference guides as well as an app for home automation (SmartPlug).

Future work will be concerned with language extensions to cover mobility-related aspects such as mobile transaction models allowing reasonable off-line usage of apps, flexible sensor handling and, based on various sensors, advanced augmented reality. Moreover, generated apps shall show high software quality; criteria of interest are correctness, usability, energy efficiency, and security. Translating the app model or log information that has been collected at run time, to a formal model of choice may help to provide powerful analysis techniques.

References

1. Adaptive Server ® Anywhere 9.0.2 – Adaptive Server Anywhere SQL User's Guide (October 2004), http://infocenter.sybase.com/archive/topic/com.sybase.help.adaptive_server_anywhere_9.0.2/pdf/asa902/dbugen9.pdf
2. Adaptive Server ® Anywhere 9.0.2 – SQL Remote ™ User's Guide (October 2004), http://infocenter.sybase.com/archive/topic/com.sybase.help.adaptive_server_anywhere_9.0.2/pdf/asa902/dbsren9.pdf
3. App Inventor. http://appinventor.mit.edu (2015)
4. Applause. https://github.com/applause/applause (2015)
5. Apple Developer Connection. https://developer.apple.com/devcenter/ios/ (2015)
6. Google Android Developer Portal. http://developer.android.com/index.html (2015)
7. ModAgile. http://www.modagile-mobile.de (2015)
8. Abowd, G.D., Dey, A.K., Brown, P.J., Davies, N., Smith, M., Steggles, P.: Towards a better understanding of context and context-awareness. In: Handheld and ubiquitous computing. pp. 304–307. Springer (1999)
9. Allen, S., Graupera, V., Lundrigan, L.: Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution. Apress (2010)
10. Arlt, H.C., Malerczyk, C.: Transformation based object classification - Ein Verfahren zur automatischen Initialisierung von Augmented Reality-Anwendungen. Friedberger Hochschulschriften, Technische Hochschule Mittelhessen University of Applied Sciences, Band 38 (2015)
11. 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)

12. Bettini, L.: Implementing Domain-Specific Languages with Xtext and Xtend. Packt Publishing Ltd. (2013)
13. Burnette, E.: Hello, Android: Introducing Google's Mobile Development Platform. Pragmatic Bookshelf (2010)
14. 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)
15. Conway, J., Hillegass, A.: iOS Programming: The Big Nerd Ranch Guide, Third Edition. Pearson Education (2012)
16. Gronback, R.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Pearson Education (2009)
17. 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)
18. Hemel, Z., Visser, E.: Declaratively programming the mobile web with Mobl. In: Lopes, C.V., Fisher, K. (eds.) OOPSLA. pp. 695–712. ACM (2011)
19. 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)
20. 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)
21. Laux, F., Lessner, T.: Escrow serializability and reconciliation in mobile computing using semantic properties. International Journal On Advances in Telecommunications 2(2), 72–87 (2009)
22. Meier, R.: Professional Android 4 Application Development. Wiley (2012)
23. Oehlman, D., Blanc, S.: Pro Android Web Apps: Develop for Android using HTML5, CSS3 & JavaScript. Apress (2011)
24. O'Neil, P.E.: The escrow transactional method. ACM Transactions on Database Systems (TODS) 11(4), 405–430 (December 1986)
25. Rubel, D., Wren, J., Clayberg, E.: The Eclipse Graphical Editing Framework (GEF). Addison-Wesley (2011)
26. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley, Boston, MA, 2 edn. (2009)
27. Vaupel, S., Strüber, D., Rieger, F., Taentzer, G.: Agile bottom-up development of domain-specific ides for model-driven development. In: Proceedings of FlexMDE 2015: Workshop on Flexible Model-Driven Engineering, pp. 12 – 21, Vol. 1470. CEUR-WS.org (2015)
28. Vaupel, S., Taentzer, G., Harries, J.P., Stroh, R., Gerlach, R., Guckert, M.: Model-driven development of mobile applications allowing role-driven variants. In: Model-Driven Engineering Languages and Systems, pp. 1 – 17, LNCS 8767. Springer (2014)
29. Vaupel, S., Wlochowitz, D., Taentzer, G.: A generic architecture supporting context-aware data and transaction management for mobile applications. In: Proceedings of the 3rd ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft 2016, Austin, TX, USA, May 16-17, 2016. IEEE (2016)
30. Williams, G.: Learn HTML5 and JavaScript for Android. Apress (2012)