A Framework for Model-Driven Development of Mobile Applications with Context Support

Steffen Vaupel

PHILIPPS-UNIVERSITÄT MARBURG

DISSERTATION

A Framework for Model-Driven Development of Mobile Applications with Context Support

vorgelegt von

Steffen VAUPEL, M.Sc.
geboren am 12. Oktober 1984
in Marburg a. d. Lahn.

Vom Fachbereich Mathematik und Informatik
der Philipps-Universität Marburg
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften

Dr. rer. nat.

genehmigte Dissertation.

Gutachter:

Prof. Dr. Gabriele Taentzer,	Philipps-Universität Marburg
Prof. Dr. Michael Goedicke,	Universität Duisburg-Essen

Prüfungskommission:

Prof. Dr. Thorsten Thormählen,	Vorsitzender
Prof. Dr. Gabriele Taentzer,	
Prof. Dr. Michael Goedicke,	
Prof. Dr. Bernhard Seeger,	
Prof. Dr. Christoph Bockisch	

als Dissertation eingereicht am:	26.10.2017
Tag der mündlichen Prüfung:	20.12.2017

erschienen: Philipps-Universität Marburg, Deutschland, 2018
Hochschulkennziffer 1180

A Framework for Model-Driven Development of Mobile Applications with Context Support

Dissertation, Philipps-Universität Marburg (1180), 2018.

Steffen Vaupel

Curriculum vitae
2011, Master of Science (Informatik), Philipps-Universität Marburg.

Philipps-Universität Marburg, Deutschland
Fachbereich Mathematik und Informatik (FB12)
Arbeitsgruppe Softwaretechnik
Hans-Meerwein-Straße 6
35032 Marburg a. d. Lahn

swt@informatik.uni-marburg.de
https://www.uni-marburg.de/fb12/arbeitsgruppen/swt

Cover Design/Umschlaggestaltung: S. Vaupel
Typesetting/Satz: S. Vaupel
Rendering date/Erstellungsdatum: 2018-07-13
Production/Druck: sedruck.de
Edition (Print run)/Ausgabe (Auflage): 1. (5)

1-330 Pages with 151 Figures (including 110 in color), 60 Tables
1-330 Seiten mit 151 Abbildungen (davon 110 in Farbe), 60 Tabellen

Printed on acid-free paper/Gedruckt auf säurefreiem Papier

Printed in Germany/Gedruckt in Deutschland

The electronic version of this document complies with the standard PDF/A-1b:2005.

Die elektronische Fassung dieses Dokuments erfüllt den Standard PDF/A-1b:2005.

Para mi amada
esposa Lisandra

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

ACID	Atomicity, Consistency, Isolation, Durability
ANTLR	ANother Tool for Language Recognition
API	Application Programming Interface
APK	Android Application Package
AR	Augmented Reality
ARM	Advanced RISC Machine
AS	Abstract Syntax
ATM	Automated Teller Machine
BIS	Business Information System
BPEL	Business Process Execution Language
BPMN	Business Process Model and Notation
BSD	Berkeley Software Distribution
CAA	Context-Aware Application
CIM	Computation-Independent Model
CMS	Content Management System
COM	Context Ontology Model
CPIM	Context Platform-Independent model
CRUD	Create, Read, Update, Delete
CSCW	Computer-supported Cooperative Work
CSS(3)	Cascading Style Sheets (Level 3)
DAO	Data Access Object
DB	Database
DBMS	Database Management System
dp	Device-Independent Point
DSML	Domain-specific Modeling Language
DVM	Dalvik Virtual Machine
EBNF	Extended Backus-Naur Form
EMF	Eclipse Modeling Framework
EMOF	Essential Meta Object Facility
FODA	Feature Oriented Domain Analysis
GCS	Graphical Concrete Syntax
GEF	Graphical Editing Framework
GI	German Association of Computer Science
GMP	Graphical Modeling Framework
GPL	General Purpose Language
GPML	General Purpose Modeling Language
GPS	Global Positioning System
GUI	Graphical User Interface
HCI	Human Computer Interface
HTML(5)	Hypertext Markup Language (Version 5)
HUTN	Human-Usable Textual Notation
IDE	Integrated Development Environment
J2ME	Java Platform 2 Micro Edition
JEE	Java Platform Enterprise Edition
JPA	Java Persistence API
JSE	Java Platform Standard Edition
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
M2C	Model-to-Code (transformation)

M2M	Model-to-Model (transformation)
M2T	Model-to-Text (transformation)
MDA	Model-Driven Architecture
MDBMS	Mobile Database Management System
MDD	Model-Driven Development
MDIA	Model-Driven Integration Architecture
MMW	My Mobile Web
MOF	Meta Object Facility
MPIS	Manufacturing and Production Information System
MTM	Mobile Transaction Model
MVC	Model-Controller-View
NFC	Near Field Communication
NIB	NeXT Interface Builder
OCL	Object Constraint Language
OLTP	Online Transaction Processing
OMG	Object Management Group
OpenGL/ES	Graphics Library for Embedded Systems
ORM	Object-Relational Mapping
OWL	Web Ontology Language
PervML	Pervasive Modeling Language
PIM	Platform-Independent Model
PIMAR	Platform-Independent Mobile Augmented Reality
POJO	Plain Old Java Object
PSM	Platform-specific Model
RGB	Red, Green, and Blue
RISC	Reduced Instruction Set Computer
RTE	Round-Trip-Engineering
SCXML	State Chart XML
SAP	Systeme, Anwendungen und Produkte in der Datenverarbeitung (Systems, Applications & Products in Data Processing)
SDK	Software Development Kit
SE	Software Engineering
SGL	Skia Graphics Library
SQL	Structured Query Language
SSL	Secure Sockets Layer
SVG	Scalable Vector Graphics
SysML	Systems Modeling Language
TM	Transaction manager
UAT	User Acceptance Tests
UML	Unified Modeling Language
VPN	Virtual Private Network
VR	Virtual Reality
W3C	World Wide Web Consortium
WAI	Web Accessibility Initiative
WCAG	Web Content Accessibility Guidelines
WebML	Web Modeling Language
XMI	XML Metadata Interchange
XML	EXtensible Markup Language
XnU	X is not UNIX

Abstract

Model-driven development (MDD) of software systems has been a serious trend in different application domains over the last 15 years. While technologies, platforms, and architectural paradigms have changed several times since model-driven development processes were first introduced, their applicability and usefulness are discussed every time a new technological trend appears. Looking at the rapid market penetration of smartphones, software engineers are curious about how model-driven development technologies can deal with this novel and emergent domain of software engineering (SE).

Indeed, software engineering of mobile applications provides many challenges that model-driven development can address. Model-driven development uses a platform independent model as a crucial artifact. Such a model usually follows a domain-specific modeling language and separates the business concerns from the technical concerns. These platform-independent models can be reused for generating native program code for several mobile software platforms. However, a major drawback of model-driven development is that infrastructure developers must provide a fairly sophisticated model-driven development infrastructure before mobile application developers can create mobile applications in a model-driven way.

Hence, the first part of this thesis deals with designing a model-driven development infrastructure for mobile applications. We will follow a rigorous design process comprising a domain analysis, the design of a domain-specific modeling language, and the development of the corresponding model editors. To ensure that the code generators produce high-quality application code and the resulting mobile applications follow a proper architectural design, we will analyze several representative reference applications beforehand. Thus, the reader will get an insight into both the features of mobile applications and the steps that are required to design and implement a model-driven development infrastructure.

As a result of the domain analysis and the analysis of the reference applications, we identified context-awareness as a further important feature of mobile applications. Current software engineering tools do not sufficiently support designing and implementing of context-aware mobile applications. Although these tools (e.g., middleware approaches) support the definition and the collection of contextual information, the adaptation of the mobile application must often be implemented by hand at a low abstraction level by the mobile application developers.

Thus, the second part of this thesis demonstrates how context-aware mobile applications can be designed more easily by using a model-driven development approach. Techniques such as model transformation and model interpretation are used to adapt mobile applications to different contexts at design time or runtime. Moreover, model analysis and model-based simulation help mobile application developers to evaluate a designed mobile application (i.e., app model) prior to its generation and deployment with respected to certain contexts.

We demonstrate the usefulness and applicability of the model-driven development infrastructure we developed by seven case examples. These showcases demonstrate the designing of mobile applications in different domains. We demonstrate the scalability of our model-driven development infrastructure with several performance tests, focusing on the generation time of mobile applications, as well as their runtime performance. Moreover, the usability was successfully evaluated

during several hands-on training sessions by real mobile application developers with different skill levels.

Kurzfassung

Die modellgetriebene Entwicklung von Softwaresystemen (MDD) ist in den letzten 15 Jahren in verschiedenen Domänen zu einem ernstzunehmenden Trend geworden. Da sich Technologien, Plattformen und Architekturparadigmen seit der Einführung des modellgetriebenen Ansatzes oftmals verändert haben, wird dessen Anwendbarkeit und Nützlichkeit erneut für jeden neuen technologischen Trend diskutiert. In Hinblick auf die zügige Marktdurchdringung von Smartphones fragen Softwareentwickler, wie modellgetriebene Entwicklungstechniken in diesem neuen und aufstrebenden Bereich der Softwareentwicklung (SE) eingesetzt werden können.

Der modellgetriebene Ansatz deckt tatsächlich viele Herausforderungen ab, welche die Softwareentwicklung von mobilen Anwendungen mit sich bringt. Elementarer Bestandteil der modellgetriebenen Entwicklung ist ein plattformunabhängiges Modell. Solch ein Modell folgt gewöhnlich einer domänenspezifischen Modellierungssprache und trennt die fachlichen Belange der Anwendung von den technischen Belangen. Das plattformunabhängige Modell wird bei der Generierung von Programmcode für diverse mobile Plattformen wiederverwendet. Ein großer Nachteil der modellgetriebenen Entwicklung ist allerdings die Notwendigkeit, eine relativ ausgereifte modellgetriebene Infrastruktur durch Infrastrukturentwickler bereitzustellen, bevor Entwickler von mobilen Anwendungen diese modellgetrieben erstellen können.

Daher widmet sich der erste Teil dieser Arbeit dem Entwurf einer modellgetriebenen Infrastruktur für mobile Anwendungen. Der konsequente Entwurfsprozess beinhaltet eine Domänenanalyse, den Entwurf einer domänenspezifischen Modellierungssprache sowie entsprechender Modelleditoren. Zur Sicherstellung der Codequalität und einer angemessen Architektur der durch die Codegeneratoren erzeugten mobilen Anwendungen werden zuvor einige repräsentative Referenzanwendungen analysiert. Dadurch erhalten die Leser sowohl Einblick darin, welche Schritte zum Entwurf und der Realisierung einer modellgetriebenen Infrastruktur notwendig sind, als auch in die Merkmale und Anforderungen von mobilen Anwendungen.

Die Domänenanalyse und die Analyse der Referenzanwendung zeigen, dass die Erkennung von Kontexten ein weiteres wichtiges Merkmal von mobilen Anwendungen ist. Gegenwärtige Softwareentwicklungswerkzeuge unterstützen den Entwurf und die Realisierung von kontextabhängigem Verhalten nur unzureichend. Obwohl diese Werkzeuge (z.B. Middlewarebibliotheken) die Definition und Sammlung kontextrelevanter Informationen unterstützen, muss die Anpassung der mobilen Anwendung üblicherweise manuell und auf einer niedrigen Abstraktionsschicht programmiert werden.

Daher zeigt der zweite Teil dieser Arbeit, wie kontextabhängige mobile Anwendungen mit dem modellgetriebenen Ansatz einfacher erstellt werden können. Es werden Techniken wie Modeltransformation und Modelinterpretation benutzt, um mobile Anwendungen zur Entwurfszeit oder Laufzeit anzupassen. Darüber hinaus hilft eine Modellanalyse und eine modellbasierte Simulation, eine erstellte mobile Anwendung (d.h., ein Anwendungsmodell) vor der Generierung oder der Bereitstellung zu beurteilen.

Die Nützlichkeit und Anwendbarkeit der entworfenen modellgetriebenen Entwicklungsinfrastruktur wird in sieben Beispielszenarien demonstriert, welche

den Entwurf von mobilen Anwendungen in verschiedenen Domänen zeigen. Die Skalierbarkeit wird durch Lasttests demonstriert, welche auf die Generierungs- und Ausführungszeit der generierten Anwendungen abzielen. Darüber hinaus ist die Bedienbarkeit innerhalb verschiedener praktischer Schulungen mit authentischen Anwendern verschiedener Kenntnisstufen erfolgreich erprobt worden.

Acknowledgements

This work would have been impossible without the aid of many people, who contributed to it in many ways.

First of all, I am very thankful to my supervisor, Prof. Dr. Gabriele Taentzer, who helped me structure and organize my sometimes curious ideas. Her knowledge and experience helped a lot in improving my working methods and to get to this point in my academic career. Furthermore, I thank Prof. Dr. Michael Goedicke for agreeing to be the second reviewer of this thesis. His suggestions helped me improve its details.

Many thanks also to the members of the PIMAR research project team and the colleagues and students who implemented parts of the model-driven development infrastructure and/or carried out other valuable preliminary work. I would like to thank (in no special order) Tillmann Heigel, Christian Konrad, Michael Bernhardt, Tobias Zülch, Konstantin Jäger, Manuel Wolf, Alexander Eifler (Domain Analysis, Reverse Engineering, and Forward Engineering), Jan Peer Harries, Samuel Becker (Android code generator), Kevin Kaiser, Felix Pape, Patrick Hofmann, Cui Xiaolin (Forward engineering and iOS code generator), Raphael Stroh, Parisa Moosavinezhad (Model editors), Jonas Kelbert (Model quality), Manuel Fleischer (Model-based simulation), Damian Wlochowitz, Stefan Schulz, Anneke Walter (Android code generator/case examples), Katharina Muth, Maria Wilfing, and Bilal Belafkir (Reverse Engineering, Test, and Assessment). Without their work, the system would not have been operational, and I thank them for their engagement. Prof. Dr. Michael Guckert and René Gerlach helped me with the coordination of the many students who were involved in this project.

The work carried out in this thesis is partly funded by the LOEWE - State Offensive for the Development of Scientific and Economic Excellence (funding line 3: promoting SME collaborative projects) and by the German Research Council (Deutsche Forschungsgemeinschaft – DFG). I would like to thank both organizations for their financial support.

During the practical assessment of the model-driven development infrastructure within the word trainer case study, Susanne Krauß and Kirill Dorofeev provided many helpful suggestions and helped me improve the usability of the model-driven development infrastructure.

I want to thank our research partners in Norway, Prof. Yngve Lamo, Prof. Lars Michael Kristensen, Dr. Volker Stolz, and Dr. Florian Mantz, for their hospitality during our visits at the *Høgskolen i Bergen*, Norway in 2014 and 2015.

Several discussions with Dennis Priefer, Wolf Rost, and Dieudonne Timma Meyatchie of the iCampus/JooMDD team (Technische Hochschule Mittelhessen – THM) helped me to position my work regarding the general design principles of model-driven development infrastructures.

My colleagues in the software engineering group, Dr. Daniel Strüber, Kristopher Born, Stefan John, Nebras Nassar, and Felix Rieger, participated in the evolution of this work, and I thank them for their encouragement and their critique. Additionally, I would like to thank the anonymous reviewers of the published work related to this thesis. Many of their useful suggestions were adopted in this thesis.

Felix Rieger read an earlier version of the thesis, and supported me in improving my English. But, of course, he is not responsible for all the errors and weak constructions I introduced afterward.

Finally, I would like to thank my family and friends, who showed amazing patience and understanding while I was writing this thesis.

Marburg, 2018-07-13

Steffen Vaupel

Chapter 1

Introduction

The market penetration of smartphones is one of the fastest in technological history. In the U.S. market, the maturation (time from 10% to 40% market penetration) of smartphones was two times faster than other technologies such as the introduction of personal Internet access. In 2011, the U.S. market for smartphones could be considered as matured [27]. This phenomenon of rapid smartphone adoption could be observed globally. The Ericsson mobility report of 2016 [Eri16] estimates a world market penetration of 5.1 billion devices and expects an increase to 6.1 billion devices in 2022. Top adopters in 2016 were consumers in South Korea (88%) and Australia (77%) [Pou16]. **1.1**

Moreover, the software market volume for these kinds of devices and hardware is even much higher. In 2013, an average smartphone user has installed 26 mobile applications [49]. Thus, we calculate an overall market penetration of over 112 billion [Eri13] installed mobile applications in 2013. Besides, the research company *StatCounter* [48] reports that in October 2016, for the first time in history, the majority (51.3%) of website requests were done through mobile devices. This indicates that a lot of web-based applications are used by mobile clients (e.g., web browsers). **1.2**

A special feature of smartphones is their ability to detect contexts (e.g., location), which also drives the popularity of mobile applications [Tan08, Chap. 8.2]. While most smartphone users always carry their devices with them (or wear them), mobile applications could detect contexts of different kinds and act in a situation-dependent way. Such context-aware mobile applications may provide information tailored to the user and situation context, which is generally known as *smart* behavior. Novel device features (e.g., sensors) support the sensing of contexts. Context-awareness allows a lot of new mobile applications in industrial, health-care, and social application scenarios. Unfortunately, technical context changes (e.g., loss of connection) may affect the reliably of mobile applications in a negative way. **1.3**

To sum up, mobile devices and mobile applications nowadays are just as important as traditional personal computer systems and standalone applications. In the near future, they will be the primary personal and professional computer systems. Context-awareness gives rise to new business cases. Lack of context-awareness can cause negative effects during mobile application operation. Hence, context-awareness of mobile applications is an important and relevant area for academic research. **1.4**

The rapid evolution of mobile computing creates a heterogeneous landscape of software and hardware platforms, versions [Gro+14], and architectural designs. Considering the market shares of the different software and hardware platforms over the years, no software platform or vendor has such a majority that the remaining platforms can be ignored. The consensus is that multiple platforms and devices need to be considered further while developing a mobile application. **1.5**

After selecting a platform (e.g., Android), the heterogeneity continues. For example, to reach at least 90% of Android users currently requires creating a mobile application supporting versions of the Android platform spanning from version 4.2 (API 17) to version 7.1 (API 25) [11]. In addition, individual device features are **1.6** very heterogeneous. Sensor availability and screen sizes may further reduce the potential user group. For example, the previously mentioned share of 90% Android users will be reduced to approximately 11% when requiring a large or extra-large screen for the developed mobile application. Additionally, current software engineering approaches do not provide the automatic adaptation or translation of a platform-specific implementation (e.g., Java used by Android) to another platform (e.g., Objective-C/Swift used by iOS). Hence, software engineering technology and development processes of mobile applications are not yet able to keep pace with the sudden rise of mobile platforms and the related heterogeneity.

1.7 Thus, developing mobile applications requires a development process that is different in many ways from the development process of traditional client-server applications or rich-client applications running on a single workstation. This development process, especially the labor-intensive implementation tasks, must be repeated for multiple software platforms (e.g., Android, iOS) to cover different user groups acceptably. For example, approximately 60% of the mobile applications were developed across different platforms [Joo+13]. Additionally, the test efforts will also increase for any additionally supported platform, version, or device (if tested at all). To mitigate or avoid these multiple native implementations, cross-platform solutions are often used.

1.8 Many existing frameworks and tools for mobile application development focus on cross-platform development as opposed to native application development. A native application has a single codebase that is written for one specific target platform and device type while using a software development kit (SDK). The codebase is translated by the appropriate target compiler and the resulting binaries run only on devices having the target platform. Requiring additional device-specific sensors limits the distribution of the application to devices of the same type. Developers must create and maintain multiple codebases to support diverse software platforms or device types. Thus, an obvious approach is to reuse a single codebase as a specification of the software system for different platforms [Hei+12]. This approach is known as *cross-platform development*. Heitkötter et al. distinguish three approaches for implementing cross-platform development: (i) runtime-based approaches, (ii) cross-compiling approaches, and (iii) model-driven development approaches (cf. Allen et al. [All+10]).

1.9 First, *runtime-based approaches* use a runtime environment to execute platform-independent program code or translated programs. For example, the Java Virtual Machine (JVM) is a well-known runtime environment executing compiled programs, while web browsers are good examples of program interpreters, e.g., JavaScript/HTML (Hypertext Markup Language.)[1]

1.10 *Cross-compiling approaches* use a cross compiler in order to translate platform-independent program code into a platform-specific or native code. Usually, they do not translate Java (favored in Android) into Objective-C (favored in iOS) or vice versa (except [Pud10]), but rather use source languages allowing program specifications in a more declarative way. This approach requires cross compilers for each of the desired software platforms. Additionally, there is often no or little abstraction between the source language and the native target language.

1.11 The last approach is the *model-driven development approach*. The model-driven development approach translates domain-specific models with high abstraction into platform-specific native program code. This approach also requires a model compiler (M2C), i.e. code generator, for each of the desired software platforms.

1.12 Web-based applications are a very popular example for *runtime-based* cross-platform development. Web-based and native applications follow opposing architectural approaches: web-based applications follow the traditional client-server paradigm and are multi-platform capable (e.g., Android, iOS). They provide good transactional

[1] This usually requires back-end systems e.g., application servers with rich functionality.

support based on the underlying database management systems, but they generally lack offline capability and cannot fully access the devices' sensors/hardware.

Although web-based mobile applications are multi-platform capable, their development approach could not be considered as platform-independent. The issue with web-based mobile applications is that functionality (if any) is located mostly on the server-side. Thus, mobile web-based applications could be considered as mobile services rather than autonomous mobile applications. In contrast, native applications can work autonomously and exploit device features (e.g., sensors, memory) very well. They usually lack platform portability due to their nativeness and platform- and device-tailored implementation. In case that native applications act as interoperable mobile clients of a multi-user system, conflict-free operation cannot be guaranteed. Different system contexts (e.g., loss of connection) cause problems in conflict detection while mobile clients are offline. **1.13**

Apparently, it is a problem to combine the advantages of the architectural designs and get rid of the disadvantages. While initiating a mobile application development project, teams often have to make an architectural and software platform choice (e.g., heading either towards a native and standalone mobile application architecture in Android/iOS or a platform-independent web-based mobile application) [Puv+16] [Bre+14], which thereafter dominates or limits the development process. Hence, software vendors and mobile application developers are increasingly asking for a development process and tools that help them to tackle the problem of the software platform and device heterogeneity and unleash the domain-specific opportunities of mobile computing (e.g., context-awareness) at the same time. Their goal is to roll out native mobile applications of high quality on the one hand, but avoid the development and test effort for all desired software platforms on the other hand. The mobile applications to be developed should additionally support different contexts, which is a specific requirement of the mobile computing domain. **1.14**

This leads to several general questions that are addressed in this thesis: **1.15**

- What are the features of mobile applications and the different platforms?
- What is supported by current (platform-independent) development approaches for mobile applications? What is missing?
- How do a development approach and tooling that covers missing parts look like?
- How can mobile applications support different contexts and how can a development approach support this aim?
- Which architectures of mobile applications are appropriate with respect to the domain-specific requirements (e.g., context-awareness)?

In the following, we provide the motivation for this work. Then we point out the challenges of mobile application development. Subsequently, the goals, the approach, and the contributions of this work are presented. This contains references to already published work. Finally, the remainder of this thesis is outlined. **1.16**

1.1 Motivation

As sketched in the preceding section, the main problem faced while developing mobile applications is the heterogeneity of platforms. Hence, platform-independent development approaches for mobile application development are generally useful to tackle this problem. **1.17**

This thesis builds on the model-driven development approach. "Model-driven development is simply the notion that we can construct a model of a system that we can then transform into the real thing." (Mellor et al. [Mel+03]) In terms of **1.18** mobile applications development, a model describes several aspects of a mobile application in an abstract manner, whereas the transformation creates a concrete implementation. This approach has many advantages (cf. den Haan [21] and Selic [Sel03]) compared to a traditional development process and fits well into the domain of mobile application development:

1.19 The model-driven development approach uses a (domain-specific) modeling language, expressing software systems on a higher abstraction level compared to the codebase; the development might be faster and thus more cost-effective. A modeling element might be transformed into several lines of code or other artifacts. In the same time span, modeling allows expressing more functionality than writing code traditionally. Moreover, in case that the model-driven development toolset contains multiple code generators for different platforms, the leverage effect will be even stronger.

1.20 Another advantage of the model-driven development approach is its focus on business aspects as opposed to technical aspects. A model-driven development process can potentially be applied by domain experts rather than only by technical experts. This separation of concerns and skills empowers domain experts while technical experts focus on the needed toolset for model-driven development.

1.21 An increased quality of the resulting mobile application could be ensured by the model-driven development approach. The quality of traditionally developed mobile applications depends on the overall and architectural design, the skills, and discipline of the mobile application developers, and the used test and review process. Model-driven development frameworks provide automatic model validation and model quality analysis. Generated native program code follows established design patterns and has a high quality by design.

1.22 The model-driven development approach also resonates well with the high market pressure in the mobile sector. Technical changes between platform versions (platform evolution) can be concealed by the code generators. Migration of existing mobile applications, i.e., models, to a higher version of the software platform is well supported. In particular, beginners with less experience in mobile application development have a lot of trouble to find adequate architectural designs which meet their requirements. In the worst-case scenario, the selected architectural design is contradictory to the requirement specification. The model-driven development approach relieves the mobile application developers at this point and provides an adequate architecture. Besides, the model-driven development approach enables changes in an agile way because the application model can be improved and refined during the development process.

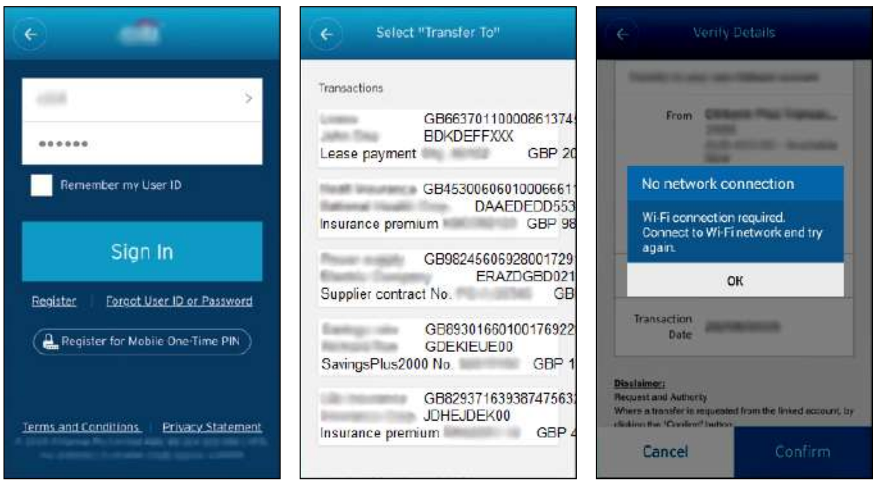
1.23 Finally, the model-driven development approach supports many use cases related to software engineering. For example, complex and error-prone program analysis tasks (e.g., concurrency problems such as deadlocks) on the level of code could be lifted to the model level [Sho+08]. Ideally, domain-specific model-driven development tools support mobile application developers while inspecting a problem of interest. With this support a manual review of the whole codebase to extract the relevant information will no longer be necessary.

1.24 To sum up, the model-driven development approach seems to be a promising approach in the domain of mobile application engineering, not only driven by the argument of multi-platform support. Although the model-driven development approach meets many software engineering requirements for mobile applications, a closer look at the different model-driven development approaches will reveal that there are further challenges.

1.2 Challenges

Several frameworks and tools that support model-driven development of mobile applications, such as Applause [Beh10] [05], MD² [Hei+13], JUSE4Android [SA14b], [SA14a], [Sil+14], Modagile [08], and Mobl [HV11b] [HV11a], have been presented in the literature. Their features will be discussed in detail later. With respect to these solutions, one might ask: why do we need an additional framework for the model-driven development of mobile applications? 1.25

The answer is related to the genesis of mobile application development. The genesis of mobile application started with web-based applications that are executed on mobile devices. Web application developers usually deal neither with heterogeneous sets of platforms and devices nor with system contexts such as network interruptions. They assume a standardized platform (e.g., a web browser) without high variability of hardware or sensors and a stable network connection to the server. Accordingly, the mobile clients are minimal in terms of architectural design, while the servers are full-featured. Current model-driven development approaches for mobile applications adopt this concept: most frameworks and tools generate mobile applications that follow a client-server architecture. Without certain extensions, this architecture is generally insufficient to deal with changing system or device contexts. A look at the landscape of existing web-based mobile applications shows that the loss of network connection (system context) very often stops mobile applications unexpectedly. Additionally, the web-based mobile applications do not always provide an appropriate graphical user interface for the particular mobile device used (device context). 1.26



(A) Login page

(B) Insufficient scaling (transfer order templates)

(C) Error message (transfer order)

FIGURE 1.1: Web-based mobile banking application

Example (Web-based banking software). Since online banking is the most popular way to make bank transactions, an increasing number of banking customers use mobile devices to access online banking services. Figure 1.1 shows a banking transaction order and the typical problems that may occur while using a web-based mobile application. A customer starts a banking session by entering the credentials at an initial login page (Figure 1.1a). S/he may have a list of transfer order templates from which one can be selected to make a money transfer. The first issue of this web-based mobile application is the inappropriate presentation of this data on the chosen device. Due to different device types, the web browser scales the graphical user interface to the devices's display size or density. As shown in Figure 1.1b, the text could thus be truncated. Besides, text may exceed the space inside the scaled 1.27

widget elements. Hence, different device contexts are not fully supported by the shown web-based mobile application. Second, if the customer starts a transaction and meanwhile the network connection to the server is interrupted, the transaction will fail (Figure 1.1c). Even short interruptions will stop the transaction permanently because customers are usually required to log in again after every interruption of the network connection. A restart of the transaction is often not possible due to security reasons. Thus, web-based mobile applications support system contexts such as the connection context insufficiently. □

1.28 Dehlinger and Dixon [DD11] identified context-awareness as one of the major research directions for mobile applications software engineering. Consequently, a challenge of today's research in model-driven development for mobile applications lies in dealing with contexts. A context is the interrelation of a mobile application to other objects/subjects. These interrelated objects/subjects can be the *platform* on which the mobile application is deployed, the *device* on which the platform is operated, the mobile *user* which uses the mobile application, the different *system* environments (e.g., available/interrupted network service), and many others. Figure 1.2 shows the two main model-driven development approaches (Web-based approach and native approach) and the level of context-awareness they can reach within different contextual dimensions.

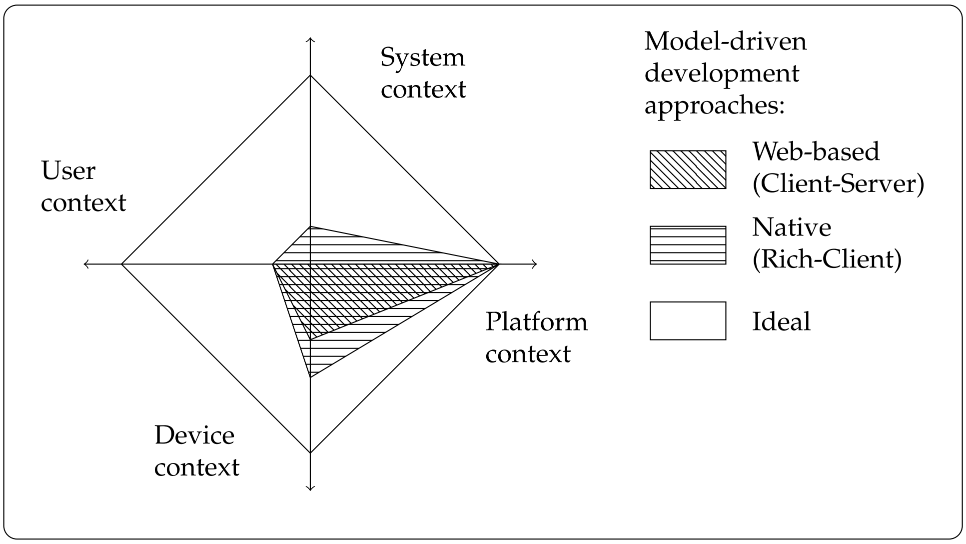


FIGURE 1.2: Model-driven development approaches and their contextual coverage

1.29 *Web-based* approaches are generally suitable for covering platform contexts. They also partly provide access to the features of a device. In turn, web-based approaches support system contexts such as the connection context poorly.

1.30 *Native* implementations cover device and system contexts in a better way. It should be noted that native approaches completely cover web-based approaches in terms of the contextual evaluation, but the effort to implement a native approach is much higher.

1.31 Web-based model-driven development approaches usually require one code generator, whereas a native model-driven development approach needs a code generator for every desired native platform. Hence, web-based model-driven development approaches also seem justified regarding development costs.

1.32 The advantages and disadvantages of native and web-based target languages are still subject of discussion [CL11].

1.33 Book et al. [Boo+05] discuss how the degree of mobility (which causes different system contexts) affects the architectural requirements. In turn, only an appropriate architectural design of a mobile application enables context-awareness, particularly with respect to different system contexts. Several model-driven development frameworks do in fact support the generation of native program code; however, they follow a rich-client architecture (Applause [Beh10] [05]) or a client-server architecture (Modagile [08]). Thus, they are either only offline-capable or only online-capable.

To sum up, we face the challenge of context-awareness and propose a model-driven development approach to create mobile applications that can deal with different contextual dimensions. **1.34**

1.3 Goals of This Work

A fundamental goal of this work is to show that model-driven development can deal with the novel and emergent domain of mobile application development. Domain experts should be able to develop mobile applications quickly and easily with a model-driven development approach. Moreover, since the model-driven development approach is usually understood and implemented as a one-level approach, i.e., using only a design model to generate a static implementation, this work will demonstrate that a two-level approach comprising a design model and a runtime model is more flexible in terms of different contextual dimensions. Our generated native mobile applications contain both a static part and a part for interpreting runtime models. Hence, these mobile applications are no longer only static but can also be configured at runtime using runtime information. This hybrid architecture of the generated mobile applications supports different contexts, as demonstrated by four selected contextual dimensions, i.e., *platform*, *device*, *user*, and *system context*. The generated mobile applications are context-aware because the two-level model-driven development approach can handle these contexts very well. **1.35**

Moreover, an additional goal of this work is to provide architectural flexibility of the generated mobile applications depending on the type of mobile application. Using existing model-driven development frameworks for mobile applications, developers cannot change the architecture of the mobile application. The architecture is prescribed by the code generators of the used frameworks. Hence, the generated mobile application may not always meet the requirements (e.g., not online- and offline-capable, not interoperable with a back end system). Thus, an additional goal of this work is to provide different architectural designs, i.e., variants, for supporting different connection contexts, which are in turn part of the *system context*. These architectural variants enable different operation modes (only offline-capable, only online-capable, and online- and offline-capable) of the mobile applications. Additionally, a model-based simulation system (cf. [Bal+04]) supports mobile application modelers at design time to evaluate their design models by estimating the transactional throughput (number of successful transactions) for a particular architecture. **1.36**

1.4 Approach

Regarding Figure 1.2, we address the first three dimensions – *platform*, *device*, and *user context* – using a two-level model-driven development approach. This approach uses design-time models for code generation and runtime models to adapt already generated mobile applications at runtime. The approach is based on the generation of native mobile applications, as web-based solutions are generally unable to work offline and limited in hardware access. **1.37**

Supporting different *system contexts* (e.g., connection context) requires additional architectural features (cf. Book et al. [Boo+05]) as part of the generated mobile **1.38** applications. However, mobile application modelers should not be involved in specifying the detailed architecture of the resulting mobile applications because we cannot assume that they are technical experts. Thus, mobile application modelers just declare the kind of the architecture (e.g., online- and offline-capable, only offline-capable, only online-capable), and the code generators configure the architectural components depending on the given requirements. The general architectural design is based on a generic extended client-server architecture supporting context-aware data and transaction management for mobile applications.

1.39 This overall approach has three major advantages. First, the model-driven development approach covers the *platform context* very well since different code generators can be used to generate platform-specific implementations from the platform-independent design models. Second, the flexibility of design models (e.g., through model transformation) allows the generation of *device*- and *user context*-specific versions of the mobile application. Moreover, runtime models enable some of this flexibility even during runtime. Finally, the approach makes domain experts capable of creating *system context*-aware mobile applications without requiring a special expertise in architectural design and the technical requirements. These advantages are evaluated by several case examples.

1.4.1 Design Models and Runtime Models

1.40 In order to support *platform*, *device*, and *user* contexts better, we propose a two-level modeling approach as a first fundamental difference from the existing one-level modeling approaches.

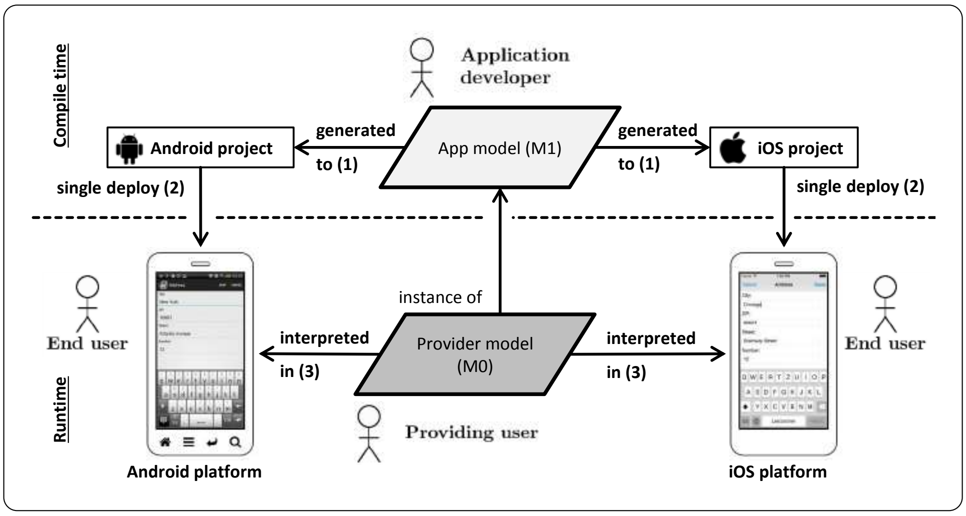


FIGURE 1.3: Two-level modeling approach

1.41 Figure 1.3 shows this novel approach in detail. A mobile *application developer* (not necessarily a technical expert) creates an *app model*. The *app model* conforms to a domain-specific modeling language (DSML) for mobile applications, which is developed during the course of this work. The mobile *application developer* can generate mobile application projects including source code and further artifacts (e.g., platform-specific graphical user interface descriptions, external libraries) for different platforms (Step 1). This already covers the *platform* context. After compiling these projects, a mobile application can be deployed to the mobile device of the respective platform (Step 2).

1.42 Using a traditional model-driven development approach, the deployed applications are usually not reconfigurable and must be remodeled, generated, and deployed again in order to realize the modification. They could not provide many configurations for different user groups or platform types at the same time. Hence, our approach uses a second modeling level. A *providing user* may create one or more *provider models*. A configuration given through this *provider model* is interpreted at runtime by the deployed mobile applications (Step 3). Hence, redeployment is not necessary. The *provider model* may modify the appearance, behavior, or data used in the mobile applications. This architecture implements a hybrid approach because runtime models are used as well as static application code [Ben+11]. The resulting applications provide runtime adaptability [Flo+06] but are not solely interpreted. This provides *user* and *device* contexts in an appropriate way, as shown later in detail by several case examples.

1.4.2 Extended Client-Server Architecture

As a second fundamental difference, we abandon the strict client-server and rich-client paradigm used by many existing model-driven development approaches. Based on the work of Satyamaran [Sat96] (cf. also Fuchß [Fuc09]), we follow an architectural paradigm called *blurred roles*. *Blurred roles* denote a relaxation of the commonly used client-server architecture. Parts of the server are duplicated on the client-side. **1.43**

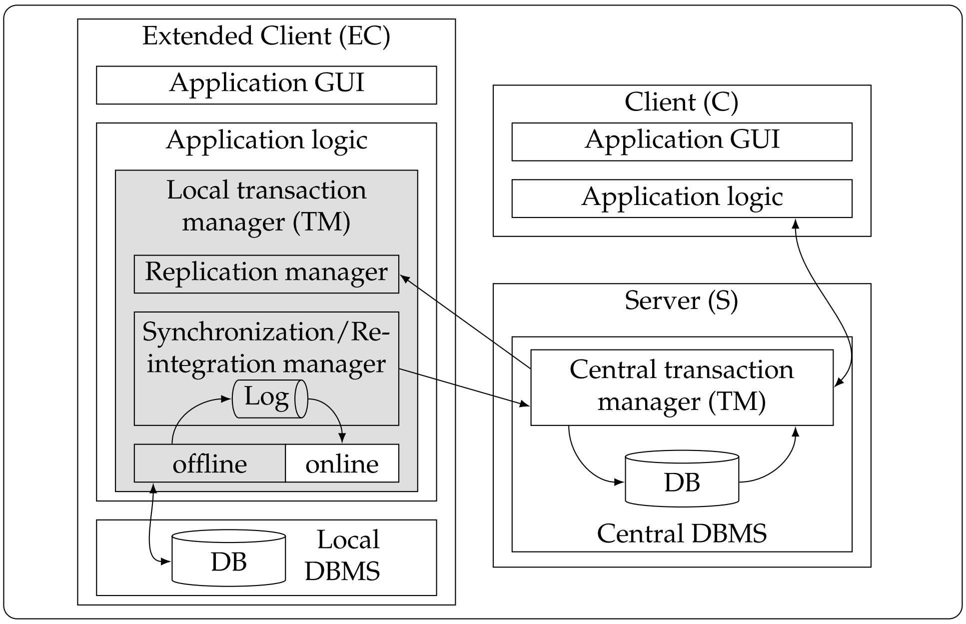


FIGURE 1.4: Extended client-server architecture vs. client-server architecture

A mobile application that follows the traditional client-server architecture, as is shown on the right-hand side of Figure 1.4, cannot operate when the connection of the client C and the server S is interrupted: required services, such as the database (DB), are no longer available. In turn, an extended client EC (left-hand side of Figure 1.4) can provide replicated data and transaction logging as part of a local transaction manager (TM). These facilities can be used while the extended client (EC) is disconnected from the server S. Moreover, the architecture of the extended client exploits a particular *mobile transaction model* (affects the implementation of the gray colored areas in the figure). Such *mobile transaction models* are required to avoid and resolve conflicts in multi-user environments. As we see later, the generated architecture of the mobile application plays an important role in increasing the transactional throughput of mobile applications. **1.44**

1.5 Contributions

1.45 The main contribution of this thesis is a framework for the model-driven development of mobile applications. Based on the particular design of this framework comprising more than state-of-the-art concepts, further contributions were made. These contributions deal with different contextual dimensions. The resulting main contributions are clustered in two parts: i) the model-driven development infrastructure and ii) the context support of mobile applications using this infrastructure. Moreover, the evaluation of each contribution constitutes an additional contribution of this work.

1.5.1 Model-Driven Development Infrastructure (Contribution 1)

1.46 The first main contribution is the model-driven development infrastructure itself. The development of the model-driven development infrastructure starts with a **requirements** elicitation and definition. The requirements are separated into modeling language requirements, architectural requirements, and tool requirements. A **domain analysis** of the target domain was subsequently carried out. We created a feature model describing the domain, even if this work does not support all identified features. After becoming more acquainted with the domain the model-driven development infrastructure should be designed for, we develop a **domain-specific modeling language** and corresponding graphical **model editor**s. This includes the creation of an abstract syntax (AS) and a graphical concrete syntax (GCS). Since a model-driven development infrastructure consists of both a domain-specific modeling language and **code generators**, the facilities for code generation are also a major contribution. In order to find appropriate architectural patterns of the mobile applications to be generated, we examined selected mobile applications and re-engineered **reference applications** that later serve as blueprints for the code templates and the architectural variants. This work supports the creation of platform-specific **code generators** [Vau+14] [Vau+18b] for Android and iOS. Such code generators for different software platforms support a platform-independent development process of mobile applications. Hence, the model-driven development infrastructure supports the dimension of platform context. Finally, based on the domain-specific modeling language and the code generators, several **prototypes** of mobile applications could be created. These contributions were developed while following an **agile bottom-up development process of domain-specific IDEs for model-driven development** [Vau+15]. The main components (i.e., the domain-specific modeling language and the code generators) of our model-driven development infrastructure are evaluated using different evaluation methods (i.e., design guidelines regarding the domain-specific modeling language, user experience evaluation, etc.).

1.5.2 Context Support for Mobile Applications (Contribution 2)

1.47 The second main contribution builds on the model-driven development infrastructure and deals with the support of different contextual dimensions. As in the first contribution, a list of **requirements** is given beforehand. The **domain analysis** is cycled again to come up with a feature model that respects additionally the context-related features of mobile applications.

1.48 Based on the two-level modeling approach (cf. Section 1.4.1), we implemented **user-** and **device context** support for mobile applications.

1.49 The model-driven development process and the generated mobile applications provide **design time instantiation** and **runtime instantiation** of application features (i.e., certain use cases) and provide manifold versions of mobile applications

[Vau+14] [Vau+16b]. In case of runtime instantiation, a regeneration or redeployment is not required. Mobile applications may be configured flexibly after deployment to certain user contexts. Hence, the model-driven development infrastructure supports the user context dimension. Moreover, considering the re-configurability of mobile applications in general, it might be interesting to also find more suitable mobile application variants related to their usability and ergonomics. For example, Lindner et al. [Lin+14] provide an efficient evaluation of the usability of Android applications by exploiting cognitive models.

The support of device contexts is also enabled by the two-level modeling approach. **1.50** Particularly, a **design time adaptation** (model transformation) and **runtime adaptation** can adapt mobile application at design time or runtime to certain device contexts. Different device types can be supported by adapting the graphical user interface (GUI) automatically to different device contexts. This contribution supports also the device-specific adaptation of behavioral aspects of mobile applications. Hence, the model-driven development infrastructure supports the dimension of device context in many respects.

Finally, to support different system contexts, we contribute a design process for **1.51** the design of online- and offline-capable mobile applications. This design process is based on **model-based analysis**, **model-based simulation** , and **model-driven generation**. Moreover, the support of different system contexts requires modifying the code generators contributed in the first part of this thesis. In order to support online- and offline-capable mobile applications, the generated mobile application must follow the generic extended client-server architecture (cf. Section 1.4.2). A preliminary evaluation of the proposed design process is carried out. It shows that our design process is useful when it comes to evaluating app designs by their offline capability and that the mobile application developers can easily apply our design process.

Apart from these contributions, the generated mobile applications provide fea- **1.52** tures for **augmented reality** (AR), which is not discussed in this work [Guc+15] [Vau+18b].

1.6 Thesis-Related List of Publications

The following papers relate to the contributions of this thesis and have been pub- **1.53** lished in publications which provide a scientific peer-review process (in chronological order):

1. [Vau+14]: Steffen Vaupel, Gabriele Taentzer, Jan Peer Harries, Raphael Stroh, René Gerlach, and Michael Guckert. "Model-Driven Development of Mobile Applications Allowing Role-Driven Variants". In: *Model-Driven Engineering Languages and Systems - 17th International Conference, MODELS 2014, Valencia, Spain, September 28 - October 3, 2014. Proceedings*. Ed. by Jürgen Dingel, Wolfram Schulte, Isidro Ramos, Silvia Abrahão, and Emilio Insfrán. Vol. 8767. Lecture Notes in Computer Science. Springer, 2014, pp. 1–17
2. [Lin+14]: Stefan Lindner, Philippe Büttner, G Taenzer, Steffen Vaupel, and Nele Russwinkel. "Towards an efficient evaluation of the usability of android apps by cognitive models". In: *Proceedings 3. Interdisziplinärer Workshop Kognitive Systeme: Mensch, Teams, Systeme und Automaten, Magdeburg, 25.-27. März 2014*. Ed. by A. Wendemuth, M. Jipp, A. Kluge, and D. Söffker. Otto von Guericke Universität Magdeburg, 2014
3. [Guc+15]: Michael Guckert, Cornelius Malerczyk, René Gerlach, Gabriele Taentzer, Steffen Vaupel, and Michael Fatum. "Plattformunabhängige Entwicklung mobiler Anwendungen mit Augmented Reality-Funktionalität".

In: *Anwendungen und Konzepte in der Wirtschaftsinformatik – AKWI* 3 (2015), pp. 14–18

4. [Vau+15]: Steffen Vaupel, Daniel Strüber, Felix Rieger, and Gabriele Taentzer. "Agile Bottom-Up Development of Domain-Specific IDEs for Model-Driven Development". In: *Proceedings of the Workshop on Flexible Model Driven Engineering co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2015), Ottawa, Canada, September 29, 2015.* Ed. by Davide Di Ruscio, Juan de Lara, and Alfonso Pierantonio. Vol. 1470. CEUR Workshop Proceedings. CEUR-WS.org, 2015

5. [Vau+16b]: Steffen Vaupel, Gabriele Taentzer, René Gerlach, and Michael Guckert. "Model-driven development of platform-independent mobile applications supporting role-based app variability". In: *Software Engineering 2016, Fachtagung des GI-Fachbereichs Softwaretechnik, 23.-26. Februar 2016, Wien, Österreich*. Ed. by Jens Knoop and Uwe Zdun. Vol. 252. LNI. GI, 2016, pp. 99–100

6. [Ger+16]: René Gerlach, Michael Guckert, Cornelius Malerczyk, Hans Christian Arlt, Steffen Vaupel, Gabriele Taentzer, and Michael Fatum. "Modellgetriebene Entwicklung mobiler Anwendungen mit Augmented Reality Funktionalität". In: *Mobile Anwendungen in Unternehmen*. Ed. by Thomas Barton, Christian Müller, and Christian Seel. Springer, 2016, pp. 193–211

7. [Vau+16a]: Steffen Vaupel, Damian Wlochowitz, and Gabriele Taentzer. "A generic architecture supporting context-aware data and transaction management for mobile applications". In: *Proceedings of the International Conference on Mobile Software Engineering and Systems, MOBILESoft '16, Austin, Texas, USA, May 14-22, 2016*. ACM, 2016, pp. 111–122

8. [TV16]: Gabriele Taentzer and Steffen Vaupel. "Model-Driven Development of Mobile Applications: Towards Context-Aware Apps of High Quality". In: *Proceedings of the International Workshop on Petri Nets and Software Engineering 2016, including the International Workshop on Biological Processes & Petri Nets 2016 co-located with the 37th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2016 and the 16th International Conference on Application of Concurrency to System Design ACSD 2016, Toruń, Poland, June 20-21, 2016.* Ed. by Lawrence Cabac, Lars Michael Kristensen, and Heiko Rölke. Vol. 1591. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 17–29

9. [Kri+17]: Lars Kristensen, Gabriele Taentzer, and Steffen Vaupel. "Towards Verification of Connection-Aware Transactions Models for Mobile Applications". In: *Petri Nets and Software Engineering. International Workshop, PNSE'17, Zaragoza, Spain, June 25-26, 2017. Proceedings*. Ed. by Daniel Moldt, Lawrence Cabac, and Heiko Rölke. Vol. 1846. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 227–228

10. [Vau+18b]: Steffen Vaupel, Gabriele Taentzer, René Gerlach, and Michael Guckert. "Model-driven development of mobile applications for Android and iOS supporting role-based app variability". In: *Software and System Modeling* 17.1 (2018), pp. 35–63

11. [Vau+18a]: Steffen Vaupel, Gabriele Taentzer, and Michael Guckert. "Model-Driven Design of Connectivity-Aware Mobile Applications". In: *Mobile Apps Engineering: Architecture, Design, Development and Testing*. Ed. by Ghita Kouadri Mostefaoui, Mitul Shukla, and Faisal Tariq. (To appear). CRC Press, 2018

1.7 Outline

The remainder of this thesis is structured as follows: the first part deals with the development of the model-driven development infrastructure for mobile applications. **1.54**

- The basic concepts and terms of the model-driven development approach and the foundations of mobile applications and their engineering will be presented in Chapter 2.
- Chapter 3 describes the chosen software development process model used while developing the framework for the model-driven development of mobile applications.
- Together with our domain experts, we collect the requirements of the model-driven development infrastructure and present these in Chapter 4.
- The domain analysis in Chapter 5 characterizes the target domain of mobile applications. As a result, a feature model describes the variants of mobile applications occurring in this domain.
- In Chapter 6, we describe the core of the model-driven development infrastructure, which is the domain-specific modeling language. We sketch the design decisions and define the domain-specific modeling language by a declarative metamodel. This metamodel is divided into data-, process-, and GUI sub-models. Besides, we provide well-formedness rules and graphical concrete syntax leading to the graphical model editor for app models. We also consider model quality. Finally, the domain-specific modeling language is evaluated with respect the modeling language requirements, existing guidelines for domain-specific modeling languages, and the guidelines for user interface description languages.
- Chapter 7 starts with the qualitative analysis of reference applications. We consider a mobile application that represents an information system and another mobile application that realizes a transactional system. A representative reference application was created subsequent to this analysis. Based on this representative reference application, the architectural design and the code can be abstracted and transferred to code generators. Additionally, the construction of code generators deals with the project and platform-specific initialization for the used IDEs, the architecture of the generated mobile applications, and the mapping of the model elements to platform-specific types. Subsequently, the processing of runtime models is explained. Finally, these contributions are evaluated.
- In Chapter 8, we present related work on model-driven development frameworks for mobile applications.

The second part of the thesis deals with the context support and demonstrates how the already provided or added functionality supports context-awareness. **1.55**

- The basic concepts and terms of context-awareness and the open issues will be presented in Chapter 9.
- In Chapter 10, we present the requirements related to the contexts of mobile applications.
- In Chapter 11, we continue the domain analysis to add context-related features to the feature model. As a result, the extend feature model describes, in addition, the properties of context-aware mobile applications.
- In Chapter 12, we show how the generated mobile applications deal with user contexts through design and runtime instantiation of role-specific processes. The evaluation includes two case examples.

- In Chapter 13, we explain how design time adaptation (e.g., model transformation) and runtime adaptation support different types of mobile devices. The mechanisms are demonstrated by a case example.
- Chapter 14 presents the design process for the design of online- and offline-capable mobile applications. To this, the model-driven analysis, model-based simulation, and model-driven generation are presented. These contributions are evaluated by an analysis of different app models. In the second part of the evaluation, we will report on the re-engineering of an existing real-world mobile application while using the proposed design process.
- Chapter 15 discusses the related work according to existing model-driven development frameworks for context-aware mobile applications.

1.56 In Chapter 16, we summarize this thesis. We give an outlook on possible future research related to this work.

1.57 This thesis contains also different appendices with further material: Appendix A (Well-Formedness Rules), Appendix B (App Models), Appendix C (Tutorial), and Appendix D (Miscellaneous).

Part I

Model-Driven Development of Mobile Applications

Chapter 2

Model-Driven Development and SE of Mobile Applications – Foundations and Definitions

The following sections exemplify the process and the components used in model-driven development approach. Since we are heading towards the construction of a model-driven development infrastructure in this work, we give a short introduction how these components can be designed. Subsequently, we present the state-of-the-art software engineering processes that are used to develop mobile applications and point out the open issues. Finally, we present several mobile platforms and their key concepts. **2.1**

2.1 Model-Driven Development

Compared to the document-centric and technology-dominated traditional software process models (e.g., V-Model [Boe79], Waterfall model [Roy87]), the model-driven development process [Mel+03] differs considerably. Model-driven development uses software models at all stages of the development process and is, therefore, a model-centric approach. This approach supports *maintainability* of software systems in such a way that modifications are made through modifications in the software models (cf. Lientz [Lie83], Lientz and Swanson [LS81] [LS80], and Lientz et al. [Lie+78]). *Portability* can be ensured by different code generators which generate native program code for different software platforms. In general, model-driven development increases the *abstraction* of software development and leaves most technical details up to the code generators. **2.2**

As stated before, the model-driven development approach uses a software model as the primary artifact. The term *model* is ambiguous, as it is used widely in general sciences and may differ considerably across different disciplines. Stachowiak [Sta73] defines that a model has the following properties: it represents an original (*mapping*), it reduces details of the original (*reduction*), and it has a purpose, i.e., is used instead of the original (*pragmatic*). Thus, models are usually *descriptive* in traditional scientific disciplines [Sei03]. However, Seidewitz states that models can be used to *specify* non-existing software systems, which is relevant to the model-driven development process. Thus, software models can be used both *prescriptively* and *descriptively* in the software engineering discipline. **2.3**

In this work, we adopt the term *model* from Kleppe et al. [Kle+03, p. 16]. They state "A model is a description of (part of) a system written in a well-defined language. A well-defined language is a language with well-defined form (syntax) and meaning (semantics), which is suitable for automated interpretation by a computer." This definition gives rise to the question to how a language can be defined well. **2.4**

In general, there are two fundamentally different [WK05] approaches to defining **2.5** languages: the (i) grammarware and (ii) modelware approach.

2.6 The grammarware approach uses the Extended Backus-Naur Form (EBNF) as a metalanguage to define (programming) languages. *Graph grammars* are similar to Chomsky grammars which are used in formal language theory. The main idea of *graph grammars* is a rule-based transformation of graphs modeling and representing different kinds of systems (e.g., software systems) [Ehr+06]. Graph grammars will not be further pursued within this thesis.

2.7 In contrast, the modelware approach uses a meta-modeling approach in order to define *metamodels* describing again a set of models. We use the modelware approach in this work because a metamodel-based definition of languages provides several features, such as the creation of well-formed models, model transformation, model synthesis (code generation), and model comparison [Pai+12]. These features are essential for the model-driven development methodology.

2.8 *Metamodels* describe the permitted structure for models themselves. The terms *metamodel* and (domain-specific) *modeling language* will be used synonymously in this work because the domain-specific modeling language is defined by a metamodel. Moreover, we often will not differentiate between *modeling language*s and *programming languages*, but we are aware that *modeling language*s raise the level of abstraction and will, in general, not have the same expressiveness as *programming languages* (cf. Neubauer et al. [Neu+14]). The permitted structure of metamodels can be described again by metamodels, which is known as the *metaisation* [Str98] principle. Metamodels can be self-descriptive in order to avoid an endless *metaisation*. In this work, we follow a four-layered architecture (cf. Figure 2.1 left-hand side). The Object Management Group (OMG) standards Meta Object Facility (MOF) [34] and Unified Modeling Language (UML) [36]; [37] can be mapped (cf. Figure 2.1 right-hand side) to this four-layer architecture.

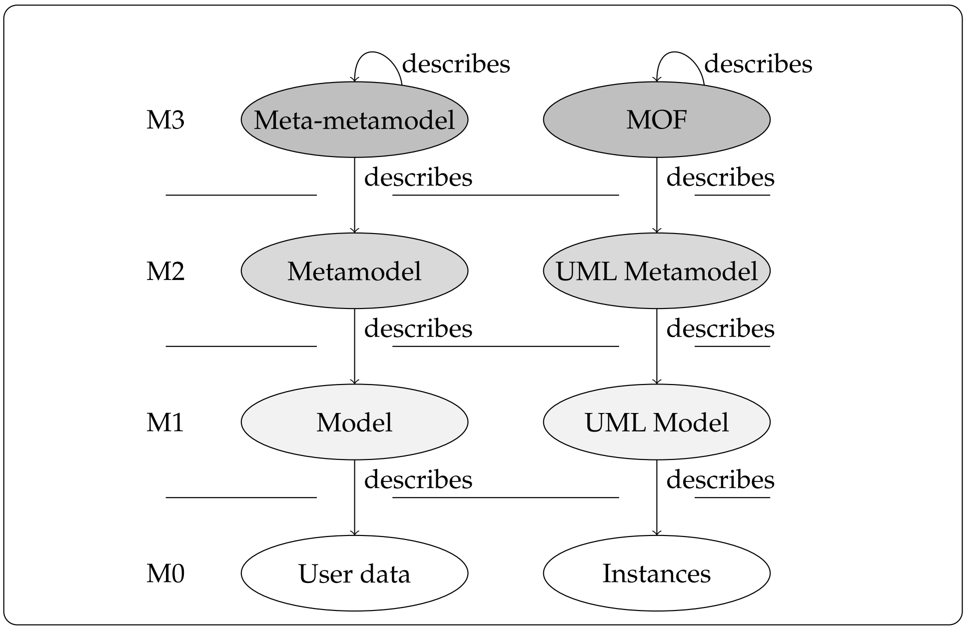


FIGURE 2.1: Four-layer metamodel architecture

2.9 The modelware approach covers the definition of an *abstract* syntax but disregards other aspects such as *concrete* syntax and *semantics* (cf. [Küh05]). A *concrete* syntax can be defined textually or visually [Fon07] by mapping the visual elements or textual tokens to the elements of the abstract syntax [Mul+08]. The concrete syntax may be derived automatically [Gar+06] from metamodels and kept synchronized [Rát+10] to the metamodel during changes. Since semantics specification is still a widely discussed topic [HR04] in the modeling community, we notice that the semantic of a model and model elements are often specified informally in practice (e.g., in UML).

Authors such as Ludewig [Lud03], Aktinson [AK03], and Kühne [Küh05] presented more sophisticated definitions of *model*s and *metamodel*s that contain a discussion of linguistic and ontological aspects. **2.10**

Having defined the crucial artifact – the model – of the model-driven development process, we can turn toward the usage of the model. According to the taxonomy of model transformations provided by Mens and Van Gorp [MG06], model transformation can be *horizontal*/*vertical* and *exogenous*/*endogenous*. A horizontal model transformation will not lower/increase the abstraction level of the source and the target model, but a vertical transformation will do so (cf. Stachowiak's *reduction aspect*). Exogenous transformations have different source and target languages. Source and target language of an endogenous transformation are identical. Model refactoring is an example of an endogenous transformation. The model-driven development process shown in Figure 2.2 exemplifies different instantiations of these kinds of model transformations. **2.11**

The process starts with a computation-independent model (CIM). The vocabulary of the CIM follows the respective application domain (e.g., business) and contains no technical or computational aspects. The first model-to-model (M2M) transformation transforms the CIM into a platform-independent model (PIM). This kind of transformation can be classified as *exogenous* and *horizontal* transformation. The resulting PIM represents the structure and behavior of the specified software system. Instances of PIM models are usually UML models or other kinds of models which are used typically in the software engineering domain (e.g., Business Process Model and Notation – BPMN, Web Services Business Process Execution Language – (WS)-BPEL, and Systems Modeling Language – SysML). **2.12**

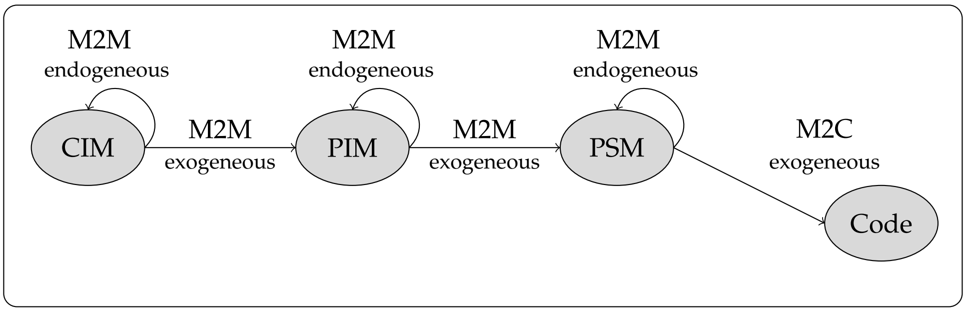


FIGURE 2.2: Common usage of software models in an MDD process

Using such models as a technical but platform-independent definition of the system, another M2M transformation creates the platform-specific model (PSM) as a result of an *exogenous* and *horizontal* transformation. A PSM expresses the structure and functionality provided by the PIM in a platform-specific modeling language. For example, a class model may define the data structure of a software system (as part of a PIM). The transformation from PIM to PSM may deliver a relational model, in case that a relational database is the target platform. While CIM and PIM are unique artifacts, a PSM may occur more than once, depending the number of platforms to be supported. **2.13**

The last transformational step transforms the PSM to native program code, which is called model-to-code transformation (M2C). This kind of transformation is an *exogenous* and *vertical* transformation. The generation of code is a *vertical* transformation because code generators usually add a lot of information and technical details and **2.14** lower the level of abstraction. *Endogenous* and *horizontal* model-to-model transformations can be performed on all models (CIM, PIM, and PSM) and represent a refactoring of the model.

2.15 Based on the automation of the different model transformations, we will clarify that model-driven development is usually tool-supported and (semi)-automated. Domain specialists use a textual or graphical model editor providing a familiar vocabulary. The model editors also provide functions for validating and optimizing (e.g., evaluation and refactoring) created models. Quality-assured and validated CIM models are translated into PIM models. Technical specialists may add computation information to the PIM models using, again, model editors. The level of automation can be differentiated by the terms *-based* and *-driven*. Streitferdt et al. [Str+08] state that model-based development uses a model once for only one desired platform, while model-driven development reuses the model artifact in an automated process for several platforms.

2.16 As Figure 2.2 shows, the process is forward-oriented, i.e., the CIM will be transformed in a unidirectional way to the native program code. Consequently, modifications of the system are in general only possible by changing the initial artifact, the CIM. However, some frameworks [AC06] provide bidirectional transformations and keep the artifacts consistent irrespective of which artifact was modified. This concept is called *round trip engineering* (RTE). Indeed, RTE is not feasible for the model-driven development process as a whole, since it seems hard to recognize and map high-level constructs (e.g., model elements) to constructs of a lower level (code statements).

2.17 The final result of the model-driven development process is native program code, as Figure 2.2 shows. Further artifacts, such as program documentation and test cases can also be generated. Compiled native program code is executable on (native) runtime environments. In contrast to this model-to-code synthesis, other approaches directly execute platform-specific models on runtime environments [VG14]. This approach needs a runtime environment that can execute the PSM. Thus, we call the traditional model-to-code synthesizer a *model compiler* and the runtime approach a *model interpreter*. Each of these approaches offer advantages and disadvantages, respectively, which will be discussed in the next section.

2.18 One difference of the model-driven development process to the model-driven architecture (MDA) initiative [33] of the OMG is that the model-driven development process is generally not limited to the OMG standards (cf. Favre [Fav04]). The model-driven development process may contain domain-specific modeling languages which are developed for domains where they should be applied.

2.19 Based on the presented MDD techniques, we select the modelware approach to define our domain-specific modeling language. Our metamodel architecture has three layers, namely Metamodels, Models, and Instances. Moreover, our model-driven development infrastructure supports PIMs and PSMs.

2.2 Design of Model-Driven Development Infrastructures

2.20 In the previous section, we describe the artifacts and components (i.e., modeling languages and model editor; code generators) which are used inside a model-driven development infrastructure during the model-driven development of a software system. However, in order to introduce a model-driven development approach to a new domain, a corresponding model-driven development infrastructure must be first created by infrastructure developers. Although there is no comprehensive state-of-the-art process or automation for the systematic development of domain-specific model-driven development infrastructures, best practices have been proposed by several authors (e.g., by Kelly and Tolvanen [KT08, Pt. IV], Völter et al. [Völ+13, Pt. II], Völter [Völ09]). We want to clarify that these best practices do not describe a general software development process on how model-driven development infrastructures could be created, but rather provide guidance on how individual components should be designed. Therefore, we contribute our agile bottom-up development process for domain-specific model-driven IDEs. Based on the following building blocks this process will be described in the next chapter.

2.2.1 Design of Modeling Languages and Model Editors

The hardest part during the development of a domain-specific model-driven development infrastructure may possibly be the identification of concepts, i.e., domain-specific language elements. Various authors, such as Deursen et al. [VD+00], Hudak [Hud96], Spinellis [Spi01], and Luoma et al. [TK05] carried out research pertaining to the domain-specific modeling language construction. They propose different methods for concept identification in new domains. **2.21**

Mernik et al. [Mer+05] describe a detailed approach to construct a domain-specific modeling language. This approach consists of the steps *decision*, *analysis*, *design*, and *implementation*. The *decision* motivates the development of the new domain-specific modeling language. The main reasons to develop a domain-specific modeling language are *notational improvement*, *task automation*, and *description of product lines*. Within the *analysis* step, the problem domain is identified and domain knowledge is gathered. Sources of knowledge are the technical documents, knowledge provided by domain experts (interviews), customer surveys and existing implementations, i.e., native program code in general-purpose languages (GPL). Mernik et al. point to two variation points within the *design* step. First, the designed domain-specific modeling language can exploit other modeling or programming languages. In this case, the domain-specific modeling language is embedded (piggyback) in another language. Otherwise, the domain-specific modeling language is developed and used in an isolated fashion (language invention). Second, the degree of formalism denotes whether the domain-specific language is described informally or with formal methods (e.g., grammars). Finally, the *implementation* steps describe how a domain-specific language can be implemented. Based on the assumption that models of a domain-specific modeling language are executable, Mernik et al. propose several implementation patterns for such languages. Three important implementation patterns are the *interpreter*, *compiler*, and *hybrid* approach, which are discussed in the next section in more detail. **2.22**

A domain-specific modeling language which is developed with the techniques mentioned before is not ready for use. The domain-specific modeling language needs a least one *concrete* syntax. Although the design of a textual or graphical concrete syntax is also a very creative part which affects the usability and acceptance of a modeling language in many respects, this task is supported well by tools. Many state-of-the-art frameworks support the semi-automated generation of textual and visual model editors. **2.23**

2.2.2 Design of Model Compilers

Technically, a code generator realizes a model-to-code (M2C) transformation. Since the input and output follow a well-defined language, the term *model compiler* is also often used in literature. Two alternative approaches are proposed in literature to create model compilers: **2.24**

2.2.2.1 Visitor-Based Approach

A *visitor-based* approach processes every model element and generates a stream of corresponding native program code. The framework Jamda (Java Model-Driven **2.25**

Architecture) [Boo03] [41] follows this approach. It provides the creation of metamodeling elements and corresponding Java classes and an application programming interface (API) for modifying these models. A visitor mechanism traverses the model and invokes code synthesis (CodeWriters) for the considered model element. The Jamda framework manipulates metamodel elements as long as possible by using cumulative transformations. A disadvantage of this framework is its focus on a top-down model compiler development. Having, as in our case, reference applications, it seems not feasible to decompose them into single transformation steps. Nevertheless, this approach will be applied partially to expand and decorate the input model with additional model elements (cf. Section 7.3.2) during a preprocessing step.

2.2.2.2 Template-Based Approach

2.26 Most model compiler frameworks (e.g., JET [67], FUUT-je, Codagen Architect, AndroMDA [24], ArcStyler, MetaEdit+ [TR03], and OptimalJ [Lon03]) follow a *template-based* approach. A template represents a unit of static code with meta-code *gaps*. These meta-code *gaps* are filled during the model compilation, using information from the app model as an instance of the domain-specific modeling language. Hence, template extraction is aimed at identifying which parts of a program are static, which parts are schematically recurring (e.g., declaration of attributes and corresponding getters and setters), and which parts depend on the modeled information. In contrast to the *visitor-based* approach, the *template-based* approach fits well to the bottom-up creation of a code generator.

2.27 Since the model-driven development infrastructure should generate runnable mobile applications, all related resources, i.e., layouts, mobile application project properties, and icons, must be generated as well, since the mentioned artifacts can also generated by a template-based approach. Hence, we deal not only with model-to-code transformation but also with a more general model-to-text (M2T) transformation. Actually, this would not affect the template-based approach. Both code generators to be developed are written in Xtend [Bet13] using the template-based approach.

2.28 **Example** (Native Program Code Template). Listing 2.1 shows a native program code template for the generation of POJOs (Plain Old Java Object). The main template function (lines 1-8) gets the modeling element *class* and generates the native program code of this class comprising the local field declarations and the accessor methods (getter and setter).

LISTING 2.1: Template for POJO generation

```
1  <<DEFINE Root FOR Class>>
2    public class <<name>> {
3      <<FOREACH attrs AS a>>
4        private <<a.type.name>> : <<a.name>>;
5      <<ENDFOREACH>>
6      <<EXPAND AccessorMethods FOREACH attribute>>
7    }
8  <<ENDDEFINE>>
9
10 <<DEFINE AccessorMethods FOR Attribute>>
11   public <<type.name>> get<<name.toFirstUpper>>(){
12     return this.<<name>>;
13   }
14   public void set<<name.toFirstUpper>>(<<type.name>> <<name>>)
         {
15     this.<<name>> = <<name>>;
16   }
17 <<ENDDEFINE>>
```

The first *static code* occurs in line 2 outside the meta-code tags («. . .»). This code will be generated without modification. The static code in line 4 may occur several times because it is part of the for-each-iteration. As seen in line 6, other template functions can be called again in order to generate native program code. This native program code appears at the location of template method invocation.

Using the class *Address* (cf. Figure 2.3) as input, the generated code is shown in Listing 2.2. From a developer's perspective, it is often not possible to distinguish between generated and manually written program code. We follow this attitude when generating human-readable code. Although models are the primary artifacts within the model-driven development approach, it is advisable to keep generated artifacts human-readable regarding labels and structure. **2.29**

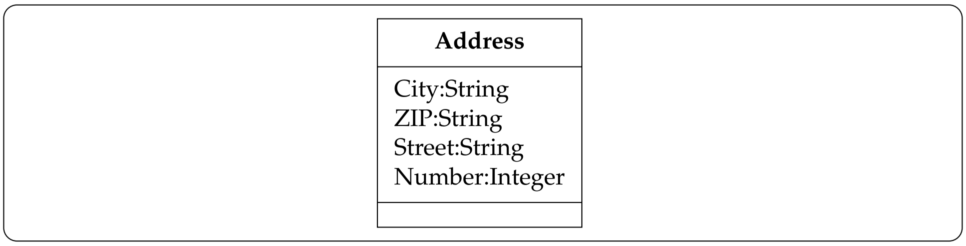


FIGURE 2.3: The class *Address*

LISTING 2.2: The compiled object *Address* (of type class)

```
public class Address
    private String : City;
    ...
    private Integer : Number;

    public String getCity(){
        return this.City;
    }
    public String setCity(String City){
        this.City = City;
    }
    ....
}
```

This encourages the manual extension of generated software prototypes (as required by our agile bottom-up development approach) as well as the maintainability of the code templates. □

A very common practice is the creation of code templates (cf. Listing 2.1) from existing code samples (cf. Listing 2.2) gained by reference applications. We will also follow this common practice of model compiler construction, but it should also be clarified how this approach is affected by model interpreter requirements. **2.30**

2.2.3 Model Compiler vs. Model Interpreter

As shown by the preceding example, the model compiler replaces the meta-code statements by the corresponding static information (e.g., class name, type names). The following two examples show the difference of a model compiling approach from a model interpreter approach: **2.31**

Example (Model compiler). Given the GUI model in Figure 2.4, the Page *ProcessesOverview* (ProcessSelectorPage) and its style settings can be translated into the native program code shown in Listing 2.3. **2.32**

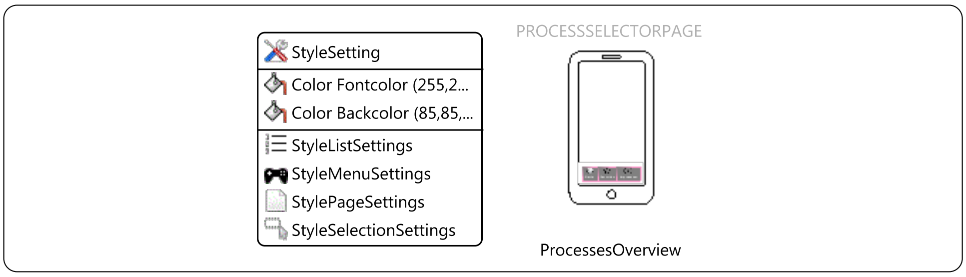


FIGURE 2.4: Excerpt of a GUI model (showing a *ProcessSelectorPage* and *StyleSettings*)

LISTING 2.3: The compiled object *ProcessesOverview* (excerpt)

```
1 ...
2 public class MainProcessesActivity extends Activity {
3     ...
4     private void createProcessList() {
5       ...
6       main_list.setFontColor(...(255,255,255));
7       ...
8     }
9 }
```

2.33 The corresponding code template evaluates the meta-code statement «`styleSetting.getFontColor().getRed()`» and generates the specified color value (=`255`) to the native program code in line 6. A program-code compiler processes the generated native program code, and the resulting mobile application shows the modeled behavior, i.e., the specified font color. □

2.34 Assume now the installation of the generated mobile application on different devices. For example, the mobile application is deployed to an eBook reader that only provides a monochrome color scheme. The mobile application may not be usable because of the unfavorable style scheme. A common practice when using the model-driven development approach is to modify the GUI model and to generate a new mobile application variant that fits better to the desired device type. By following this approach, to deal with the device heterogeneity, a countless set of mobile applications and app models will result. According to the proposed approach (cf. Section 1.4.1) to use *design models* and *runtime models*, a better solution seems to be the configuration of some values at runtime. This is what a model interpreter realizes.

2.35 **Example** (Model interpreter). Given the same setting as in the preceding example, a slightly changed code template produces the native program code shown in Listing 2.4. The code template generates static code that works similarly to the meta-code statement shown in the preceding example.

LISTING 2.4: The partly compiled object *ProcessesOverview* (excerpt)

```
...
public class MainProcessesActivity extends Activity {
    ...
    private void createProcessList() {
      ...
      main_list.setFontColor(...(
        RuntimeModel.getStyleSetting().getFontColor().getRed(),
            ...));
      ...
    }
}
```

The introduced statement in line 7 loads a runtime model at runtime (provided by the class `RuntimeModel`) and inserts the corresponding color value. Thus, the resulting mobile application can be configured at runtime by a runtime model. Modification of the design model, regeneration of the mobile application, and redeployment are not necessary. □

The question that may arise now is: why construct a model compiler instead of a model interpreter? Similar to the traditional discussion regarding interpreters and compilers, both have advantages and disadvantages. **2.36**

Model compiling has the following advantages (cf. den Haan et al.[1]) over model interpretation: **2.37**

Model compilers capture a lot of technical domain knowledge in the shape of generation rules. The generated mobile applications can be generated in such a way as to look like manually coded mobile applications. Model instances and generation rules remain hidden to the mobile end user. Thus, compiled model-based mobile applications do *not reveal too much knowledge* about the domain and are *easier to understand* compared to a model interpreter. **2.38**

From an infrastructure developer's perspective, a model compiler approach is *easier to start with* because reference applications can be used for code extraction and concept analysis. Besides, the model compiler approach is more *iterative*. A number of meta-code statements inside a code template could be increased or refined during the creation of the model-driven development infrastructure. Another great advantage consists in *debugging* support. The generated native program code can be easily debugged using conventional debugging approaches, while model interpreters are hard to debug. **2.39**

In contrast, model interpreters have the following advantages over model compilers: **2.40**

Model interpreters enable *changes at runtime*, which is of great importance regarding the issue of context-awareness. Regarding platform portability, model interpreters may abstract from concrete platforms and execute platform-independent models. Finally, a model interpreter usually needs to be deployed only once and can then be used for runtime modifications of the mobile application using the runtime model. **2.41**

Although the model interpreter approach sounds very flexible, not all platforms fully support such flexible instantiation. For example, the Android platform does not support the programmatic access to all graphical user interface elements due to a static resource management. Thus, the implementation of the model interpreter is sometimes limited by the runtime behavior of a software platform. **2.42**

Based on the presented design variants for the different components of a model-driven development infrastructure, we developed a domain-specific modeling language as a self-contained language that depends on no other language. We initially focus on a graphical concrete syntax for this language, even if a concrete textual syntax might be useful for certain groups of users (i.e., technically-skilled developers). The code generators follow a template-based approach since most meta-tools support this kind of approach. Moreover, a template-based approach allows a bottom-up construction of code generators. Finally, we followed both a model compiler approach and a model interpreter approach in this thesis, particularly to benefit from the advantages of each of the approaches. Later on, we will describe which information can be modeled and evaluated at runtime (cf. Section 7.3.4). **2.43**

2.3 Software Engineering of Mobile Applications

Next, we turn our attention to the software engineering process of mobile applica- **2.44**

[1] http://www.theenterprisearchitect.eu/blog/2010/06/28

tion. Empirical studies reveal that software engineering of mobile applications is dominated by agile methods, as 86% of survey participants confirm that the agile methods are appropriate for the development of mobile applications [Flo+14a]. Scrum was the most closely followed approach. Apart from these empirical findings, Abrahamsson shows that agile methods fit very well to the traits observed in mobile software development [Abr05]. Agile methods match well because of small development teams (41% have 5-15 members [Flo+14b]), short development cycle durations (57% projects have a project duration of 6-18 months [Flo+14b]), and high product release frequency (46% of the projects have a monthly release), whereas traditional software engineering processes need more lead time for up-front analysis and design.

2.45 Although general agile methods can be applied, the existing agile methods were refined in order to optimize them for the particular requirements of mobile application development. Proposed approaches are Mobile-D [Abr+04] from Abrahamsson et al., Hybrid Method Engineering (HME) [RR08] by Rahimian and Ramsin, the Mobile Application Software Agile Methodology (MASAM) [Jeo+08] by Jeong et al., and the Scrum and Lean Six Sigma (SLeSS) approach [Cun+11] by Cunha et al. In this context, the work from Rahimian and Ramsin is noteworthy because they map the main characteristics of agile software development to the traits observed in mobile software development. Hence, agile principles and mobile application development match well. Unfortunately, most of these development processes have not been evaluated or tested in practice.

2.46 The community still considers a number of open issues: in 2007, a GI[2] Dagstuhl Research Seminar on *Software Development Methodologies for Mobile Applications* defined three main challenges of mobile application software engineering. The main research dimensions identified by the participants are i) the *architecture* of the device, ii) the *data and context management*, and finally iii) the *user interface heterogeneity* [Kön09]. Wasserman [Was10] confirms the mentioned issues but also explains that *non-functional requirements*, such as battery lifetime or exceptional events such as the loss of connectivity, are yet to be covered well by current software engineering approaches. He also mentions *portability* as an open issue and sketches different research directions to solve the problem. Additionally, Dellinger and Dixon [DD11] propose the use of self-adaptive mechanisms to better support the dynamism in mobile applications at runtime. Muccini et al. [Muc+12] highlight the insufficient methods to *test* mobile applications. Besides these well-separated research directions, Alencar and Cowan [Ale12] point to several additional issues and emergent applications.

2.3.1 Model-Driven Development of Mobile Applications

2.47 As already shown in Figure 1.2, the existing model-driven development frameworks show only a low contextual coverage, even if the generated mobile applications follow a web-based or native implementation. The state-of-the-art of model-driven development of mobile applications has two major shortcomings: first, the traditional model compiler approach hardly allows any configuration of the mobile applications at runtime. Consequently, a mobile application cannot be configured properly to a specific device- or user context. Second, the architecture of the generated mobile applications is defined by the code generators. Hence, the architecture cannot be changed or modified according to the current requirements of the mobile applications.

[2] German Association of Computer Science

2.4 Mobile Software Platforms

This section presents the state-of-the-art concepts and technologies that are provided or used by the major mobile software platforms. We select two native platforms (Android and iOS) for this presentation since they are the most common mobile platforms. Additionally, we present a cross-platform technology, since the cross-platform approach is (due to its platform-independence) closely related to the model-driven development approach. We take a closer look at Apache Cordova [03] since it is a representative state-of-the-art framework for cross-platform development of mobile applications. Apache Cordova is available for many software platforms. **2.48**

2.4.1 Android – Concepts and Technology

The first commercial version of the Android operating system was released in 2008 by the Open Handset Alliance and Google. The operating system is open-source (Apache License 2.0/GNU GPL v2 for the Linux kernel), but may have proprietary third-party components. It targets smartphones, tablet computers, TVs, eBooks and different consumer electronics devices from a huge set of different vendors. It runs on 32/64 bit ARM (Advanced RISC Machines), x86, x86-64, MIPS and MIPS 64 (MIPS Technologies) hardware architectures. A graphical multi-touch user interface serves as the default user interface. **2.49**

The internal software stack of the Android operating system (see Figure 2.5) is based on a *Linux kernel*. The kernel provides process and memory management and connects the hardware of the device to the application-oriented upper layers. Several libraries inside the *Libraries* layer provide, for example, HTML rendering (WebKit), persistence (SQLite), C-language support (libc), security functionality (Secure Sockets Layer – SSL), media/graphics support (Open Graphics Library for Embedded Systems – OpenGL/ES; Skia Graphics Library – SGL), and other built-in functionality. The sub-layer *Android Runtime* is particularly relevant in the context of our work, as it used to execute Java-based programs. Its Dalvik Virtual Machine (DVM) is the counterpart to the Java Virtual Machine (JVM). The core libraries provide different Android-specific classes and Java core classes (e.g., `android.*`, `java.*`, and `javax.*`). However, this library does not contain all standard Java SDK classes. The Dalvik virtual machine is designed for limited resources (e.g., CPU, RAM avg. 10MB, max. 64 MB) and works based on registers (instead of using a stack). The *Application Framework* layer provides many higher-level services to Android applications. Finally, the *Applications* layer comprises the installed mobile applications. This includes standard applications (e.g., Home Screen, Browsers, and E-Mail reader) and user-installed applications. The latter ones are distributed over a mobile application store. Mobile applications which are to be distributed through this store must be signed by the publishing authority. **2.50**

From a mobile application developers' perspective, an Android application is comprised of Java executable files (in Dalvik format), XML (EXtensible Markup Language) files that describe resources and layouts, and different mobile application resources like videos, images, and audios. All artifacts are bundled to an .apk-File (Android Application Package – APK). An essential file of this package is the Manifest file, which contains meta-information about the developed application. The APK contains, amongst standard Java SDK classes, classes which are part of the Android SDK library. This library reflects the application framework layer shown in Figure 2.5. For example, *Activities* can be used to create a screen that can interact with the user. In turn, *Activities* can interact with each other using *Intents*. Functionality that has no user interaction through the graphical user interface can be provided by non-visible *Services*. As an example, a music player might be realized with a service. Since every mobile application runs in its own instance of **2.51**

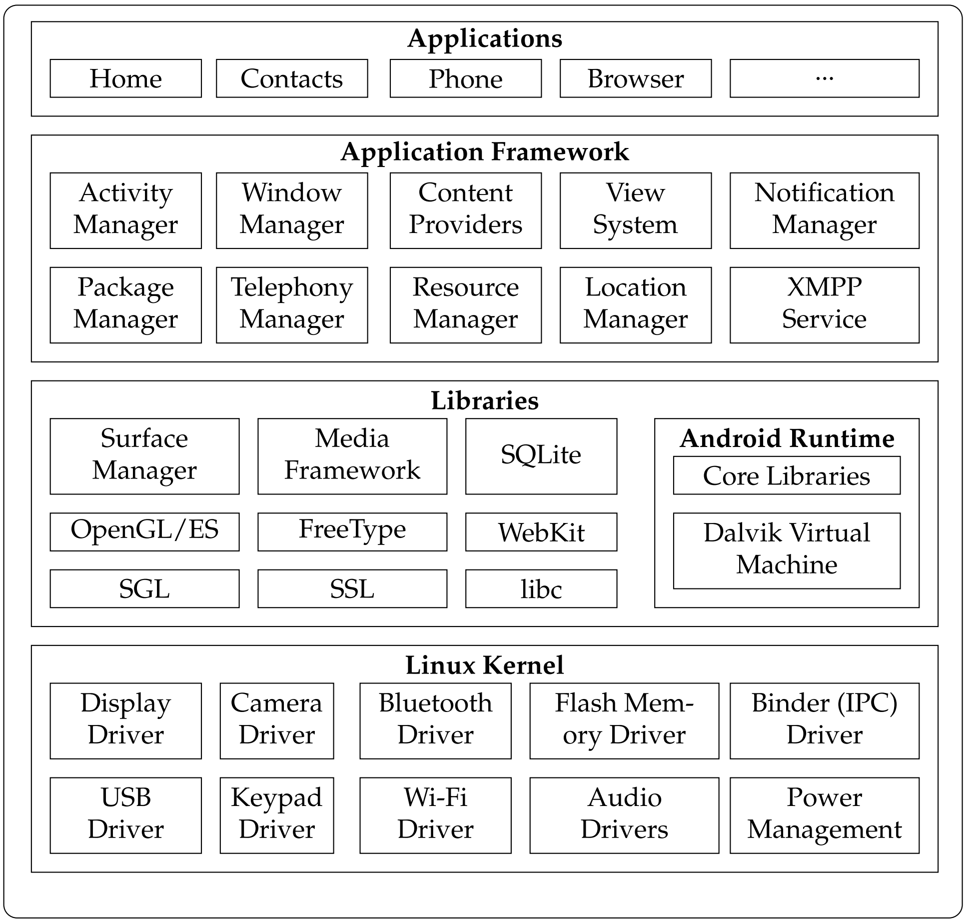


FIGURE 2.5: Android software stack

the virtual machine, *Content Providers* and *Broadcast Receivers* are used to establish a communication between mobile applications.

2.52 A model-driven development infrastructure designer must keep in mind that the Android platform neither provides full support in terms of the Java SE specification nor the Java JEE libraries. Any functionality which is taken from a third party library must be bundled to the developed mobile application. Moreover, the Dalvik virtual machine is also limited in many respects (e.g., the overall number of methods in a mobile application).

2.4.2 iOS – Concepts and Technology

2.53 The first version of iOS (formerly known as iPhone OS) was released in 2007 by Apple Inc. The operating system is closed-source. It exclusively targets devices from Apple, such as smartphones (iPhone), tablets (iPad), and multimedia devices (iPod Touch). It runs on different versions of ARM hardware architectures. The user interface is graphical and multi-touch capable.

2.54 The internal software stack of iOS (see Figure 2.6) is based on the XnU[3] kernel which is, in turn, an incarnation of the Mach kernel (Carnegie Mellon University). Moreover, the Darwin operating system and Berkeley Software Distribution (BSD) are also ancestors of iOS. The *Kernel and Device Drivers* layer provides additionally the device drivers. The *Core OS* layer provides frameworks for realizing process intensive calculations (Accelerate Framework), interaction with wireless devices

[3] X is not UNIX

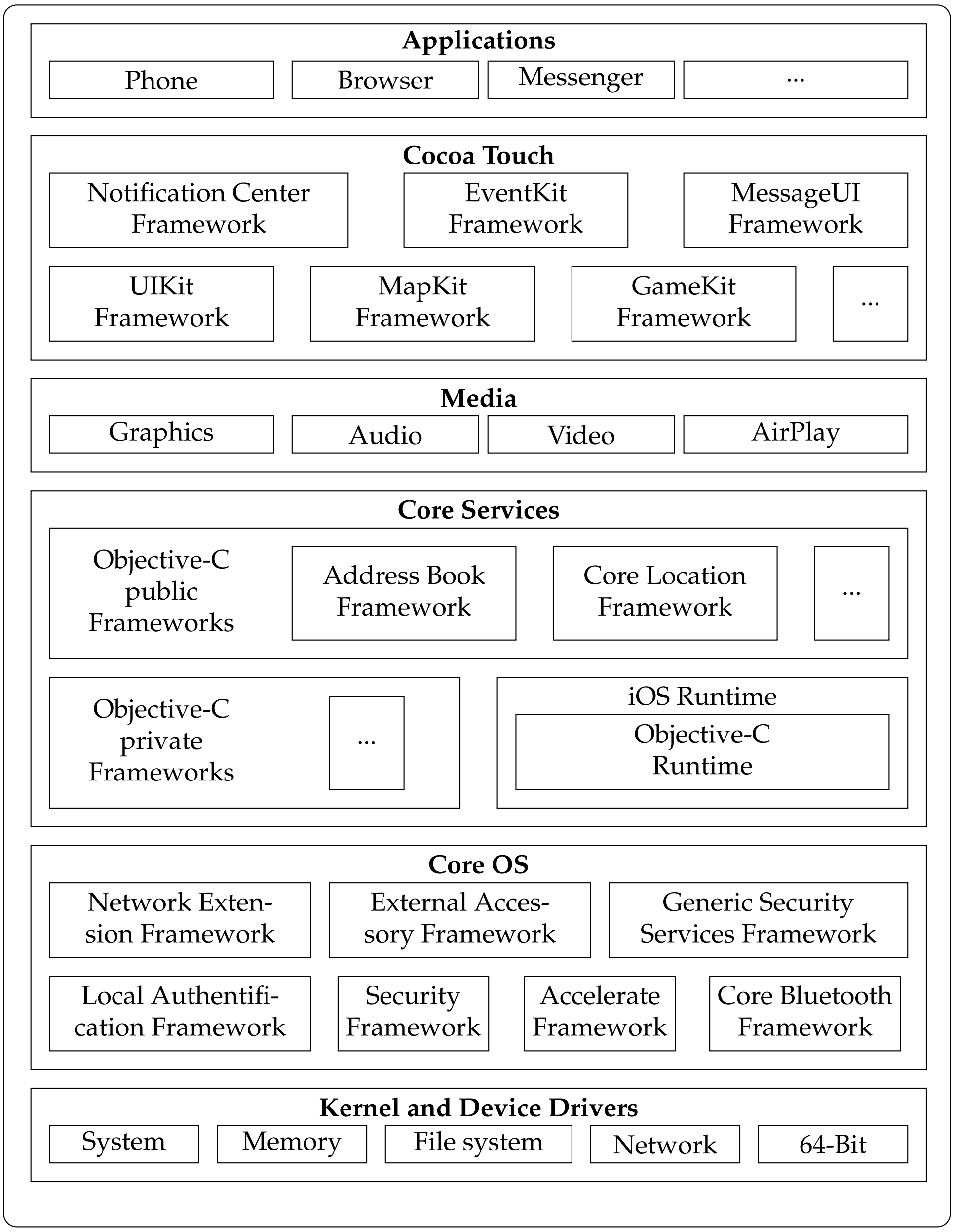


FIGURE 2.6: iOS software stack

(Core Bluetooth Framework), accessing external devices which are connected via the docking interface (External Accessory Framework), secure applications (Generic Security Services Framework/Security Framework), local authentication (Local Authentication Framework), and connections and configurations (Network Extension Framework) of Virtual Private Networks (VPN). The *Core Services* layer provides the Objective-C runtime and the Objective-C private and public frameworks. The public frameworks can be accessed by every third-party application while the private frameworks are reserved for manufacturer applications, i.e., Apple's own applications. The *Media* layer provides many frameworks for processing media of different kinds. The *Cocoa Touch* layer contains the key frameworks for building iOS applications. These frameworks (e.g., the UIKit Framework) provide the basic application infrastructure such as user interface management, the model-view-controller stubs, graphics and windowing support, etc. Finally, the *Applications* layer contains the pre-installed mobile applications as well as the user-installed applications. The latter ones are distributed over a mobile application store.

2.55 From a mobile application developers' perspective, an iOS application contains compiled and executable files (prepared for ARM architecture), NIB files (NeXT Interface Builder) which contain the layouts, and resources like videos, images, and audios. All artifacts are bundled to an .app file (Application bundle file) which in turn is enveloped by an .ipa-File (iOS application archive file). An essential file of this archive is the Info file (Info.plist), which contains the meta-information of the particular mobile application. The archive may contain standard Objective-C classes which are not specific to iOS, as well as classes which relate to the frameworks shown in Figure 2.6. Typical iOS-specific classes are *UIViewController* which are used to interact with the user through the graphical user interface. A transition between view controllers can be realized with *Seque*s.

2.56 A model-driven development infrastructure designer must keep in mind that the generation and code compilation of iOS applications requires the proprietary XCode building tool in order to seal and encrypt the resulting binary. Moreover, the set-up of a project structure inside or outside XCode might be not supported as XCode is a proprietary IDE.

2.4.3 Comparing Android and iOS

2.57 As shown in Figures 2.5 and 2.6, the architectures of both platforms follow a layered architecture containing similar components. The biggest difference according to the software platform is the underlying hardware platform. iOS is limited to proprietary devices (e.g., iPhone and iPad), whereas Android can be found on devices manufactured by different vendors.

2.58 From a software developer's perspective, both platforms follow a declarative editor-based definition of the graphical user interface (XCode - iOS/XML - Android). Business logic can be specified in commonly-used programming languages (Objective-C – iOS/Java and C – Android). However, mobile applications for iOS can only be developed with the XCode IDE, which runs only on macOS (Apple). In turn, different IDEs (Eclipse, Android Studio) for the development of Android applications are available for different operating systems (e.g., macOS, Windows, and Linux). Since both platforms generally follow the same concepts, the market shows no clear leader.

2.59 Hence, the selection of a native platform depends more on individual factors (e.g., personal technical skills, license budget, and targeted user group). However, Android is often considered as more cost-effective and used widely in an academic context. For more details, please refer to Goadrich & Rogers [GR11] and Gronli et al. [Grø+10].

2.4.4 Cross-Platform Technologies

2.60 As shown in the introduction, the cross-platform approach follows the credo "write once, run everywhere" whereas the model-driven development approach sound "model once, run everywhere". These statements show that both approaches are very strong competitors and mobile application developers, especially technical experts, often have difficulties to choose between these approaches. Hence, we also present the concepts of cross-platform development based on a current cross-platform development framework. Additionally, one not pursued possibility of the model-driven development approach is the generation of cross-platform program code. Indeed, model-driven development could cover the cross-platform approach in case that the model-driven development infrastructure provides a code generator for cross-platform languages. The implementation of a cross-platform code generator is advised, provided high platform coverage is pursued but a moderate hardware access can be accepted. Thus, the implementation of one cross-platform code generator could be more cost-effective than the implementation of one or more native code generators.

A popular, state-of-the-art cross-platform technology is *Apache Cordova* [03] (formerly PhoneGap). Apache Cordova uses CSS3 (Cascading Style Sheets – Level 3), HTML5 (Hypertext Markup Language Version 5) and JavaScript as a platform-independent programming language. The framework supports different platforms, such as Android, iOS, Blackberry, and Windows Phone. An Apache Cordova application contains HTML5 and JavaScript. It can make calls to native application code of the particular platform it is running on. Thus it follows both a runtime approach by executing the web-based parts and a cross-compiling approach by providing the ability to execute native application parts. Native routines must be implemented beforehand on all targeted platforms by the IDE developers. These functions are stored in API libraries. Mobile application developers that need further functionality may create their own plugins. Similar to the traditional development approach, these plugins must be manually implemented for each target platform (e.g., iOS and Android). **2.61**

Figure 2.7 shows the architecture of an Apache Cordova application running on the Android platform. The implementation looks similar for other software platforms. The application is separated into a *Web Architecture* part (left-hand side) and an *Android Cordova Container* (right-hand side). The *HTML Android Application* container contains the developed application, which is separated into a *UI Layer* (HTML, CSS, and JavaScript; cf. Oehlman and Blanc [OB11]) and an *Application Logic* layer. The *HTML Android Application* makes HTML/JavaScript and Cordova JavaScript API calls to the *Android WebView* component that in turn make Android-API calls to the Android platform. Moreover, the Android WebView component delegates the Cordova JavaScript API calls to the responsible Cordova plugins. These plugins make Android API calls to the Android platform. Thus HTML Android Application can reach any native functionality either by preexisting or custom plugins. **2.62**

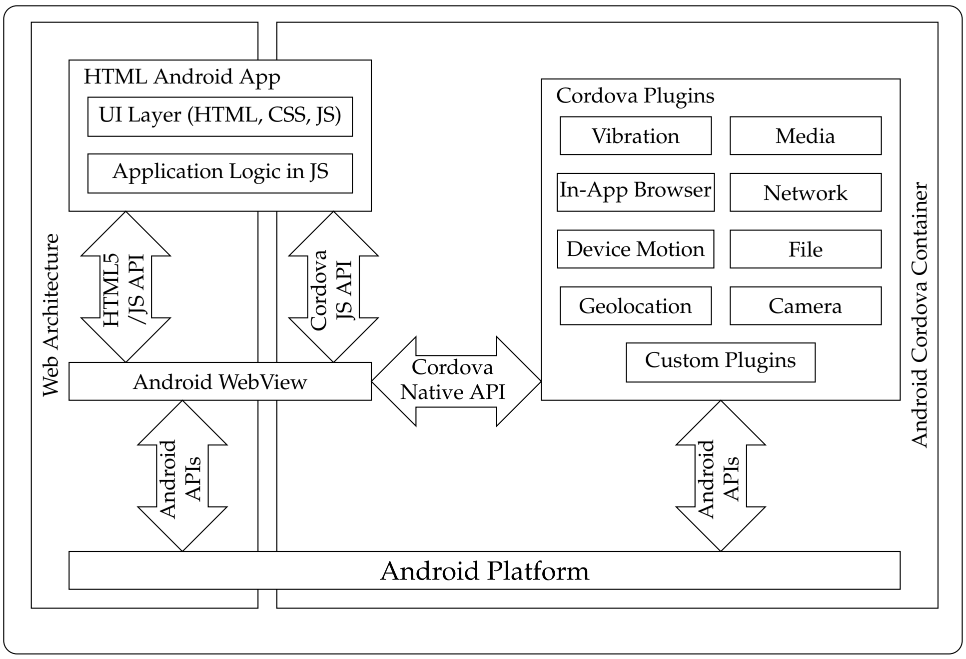


FIGURE 2.7: Architecture of an Apache Cordova application running on Android (cf. Brucker and Herzberg [BH16])

From a mobile application developer's perspective, the core of a Cordova application is provided by the HTML, CSS, and JavaScript code inside the folder *www*. The meta-information is stored in the *config.xml* file. These core artifacts are platform **2.63** independent. Mobile application developers might add platform-specific code inside the folder *plugins*. The platform-independent program code is wrapped into a hybrid and platform-specific mobile application.

2.64 Even though the development of a cross-platform code generator is not intended by this work, a model-driven development infrastructure designer must keep in mind that the cross-platform approach might limit the architectural design of a mobile application. For example, different architectural patterns (e.g., Model-View-Controller) are not feasible since the cross-platform approach follows a flat architectural design by putting the application logic only to one single layer which in general seems not to be adequate.

2.65 As mentioned in the introduction, we focus on native application development due to their full capability in terms of hardware access. In particular, we contribute code generators for both Android and iOS platforms. However, using a model-driven development process, we are potentially able to provide code generators for cross-platforms (e.g., Apace Cordova).

2.4.5 Comparing Cross-Platform Technologies and MDD

2.66 A common characteristic of both approaches is the abstraction of the software platform. Cross-platform technologies reuse mostly web-based languages, whereas the model-driven development approach reuses existing general-purpose modeling languages or domain-specific modeling languages. From the developers' perspective, cross-platform approaches provide no abstractions in terms of technical concerns. Hence, cross-platform approaches are suitable for technical experts. The model-driven development approach might also be useful for domain experts with less technical skill. Moreover, cross-platform approaches are limited when it comes to accessing hardware. Such device-specific parts cannot be implemented in cross-platform languages, but can be realized using native languages.

Chapter 3

Agile Bottom-Up Development of Domain-Specific IDEs for Model-Driven Development

While developing the model-driven development infrastructure[1], the chosen software development process once again employs a model-driven development model [Völ+13] as well as agile software development methods [09]. We align domain-specific, platform-independent abstractions provided by model-driven development with agile principles such as *quick response* and *early delivery*. Experience has shown that model-driven and agile practices complement each other well [Kul+11] [ZP11]. Having developed a domain-specific modeling language, the development of corresponding model editors and code generators is facilitated by a wealth of meta-tools: GMF [65], Sirius [70] and Xtext [72] for model editor development, and Xtend [Bet13] and EGL [61] for generator development. **3.1**

In the state-of-the-art process of using these meta-tools, the developer analyzes one or several reference applications completely and extracts knowledge to specify the domain-specific IDE components, namely the domain-specific modeling language, corresponding model editors, and code generators. This approach, referred to as bottom-up development [BS13], assumes that full reference applications are provided upfront, which is reasonable if the involved technologies and user requirements are stipulated at the start of the project. In rapidly evolving software domains, however, this assumption does not hold anymore: due to changing user demands and underlying technologies, a domain-specific modeling language is exposed to evolution during its whole lifespan. Additionally, a domain-specific modeling language may change due to refinement and extension steps. Hence, enabling the co-evolution of the domain-specific modeling language and depending artifacts, such as the textual or graphical model editor and the code generators, poses a challenge. The following research question arises: how can domain-specific IDEs be developed systematically in the presence of modeling language evolution? **3.2**

Based on these experiences and existing tools, we apply a modified software development process called the *agile bottom-up development process of domain-specific IDEs for model-driven development* [Vau+15]. This process focuses on the co-evolution of a domain-specific modeling language, its model editors, and code generators. The key is to organize language evolution into fine-grained evolution steps: in each step, prototype models are employed to generate one or several application prototypes. The mobile application developer manually modifies the prototypes as required for the evolution step. The IDE developer then identifies aspects concerning the domain-specific modeling language, model editors, and code generators. These aspects are used as input for their synchronous evolution. Afterward, the application **3.3**

[1] Not to be confused with the software development of mobile applications itself while using the model-driven development infrastructure.

prototype is no longer required. The process is not designed for any specific agile methodology, but can be aligned with agile methodologies, e.g., Scrum.

3.1 Agile Bottom-Up IDE Development Process

3.4 The agile bottom-up IDE development process applied in this thesis consists of three steps: first, to define an initial domain-specific modeling language and IDE, a domain analysis is carried out, involving the extraction of domain concepts and generator templates from existing reference applications.

3.5 Second, in the course of continuous language and IDE development, infrastructure developers, as well as mobile application developers, perform evolution steps, including the generation and modification of prototypes and successive evolution of the domain-specific modeling language and IDE.

3.6 Third, evolution steps may require a follow-up migration step to reconcile inconsistencies introduced in existing prototype app models during the evolution step. These app models are model-based descriptions of prototypes. For each of these activities, we outline the involved manual and automated tasks, and the tools supporting these tasks.

3.1.1 Domain Analysis

3.7 Different factors can cause a change to the model-driven development approach: first, in large software projects, a lot of boilerplate code may exist due to similar use cases. Second, a number of separate unrelated applications might show similarities in structure and behavior. Third, it may be required to deploy one individual application to several target platforms. In each of these scenarios, the abstraction level of development can be lifted by using domain-specific modeling languages with code generation facilities. The initial step to establish such a domain-specific modeling language is called *domain analysis*, which is based on one or several reference applications.

3.8 Domain analysis involves three steps: *quality assurance*, *domain concept identification*, and *template extraction*. *Quality assurance* means ensuring that the existing applications exhibit high quality, rendering them suitable as reference applications for code generation. This task involves the identification of anti-patterns and refactoring toward design patterns. During *domain concept identification*, concepts recurring throughout the reference applications are identified; they are reflected in the model elements of the domain-specific modeling language. The aim of *template extraction* is to specify generator templates: a generator template represents a unit of code with gaps. The gaps are filled during application development by the generator, using application-specific information derived from instances of the domain-specific modeling language.

3.9 Quality assurance can be partly automated using static analysis tools supporting the detection of anti-patterns and code smells [Nov+10]. A promising technology to detect recurring concepts is automated clone detection [Roy+09]. To our knowledge, there is no specific tool to manage the extraction of templates based on reference applications, thereby leaving it a fully manual step.

3.1.2 Continuous Language and IDE Development

3.10 We propose to develop IDE components, notably textual or graphical model editors and code generators, in fine-grained iterations (cf. Figure 3.1): first, infrastructure and mobile application developers decide on the next feature that should be supported by the domain-specific modeling language and its IDE. The mobile application developer then goes on to generate one or more prototypes from app models and to manually extend these prototypes creating the code required to implement this feature. The extension is then analyzed by the infrastructure developer, which results in a synchronous evolution step of the domain-specific modeling language and its IDE.

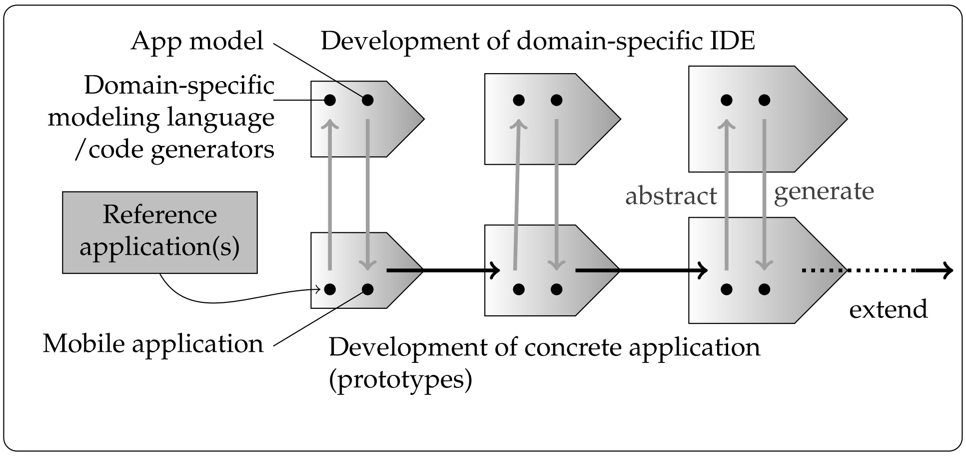


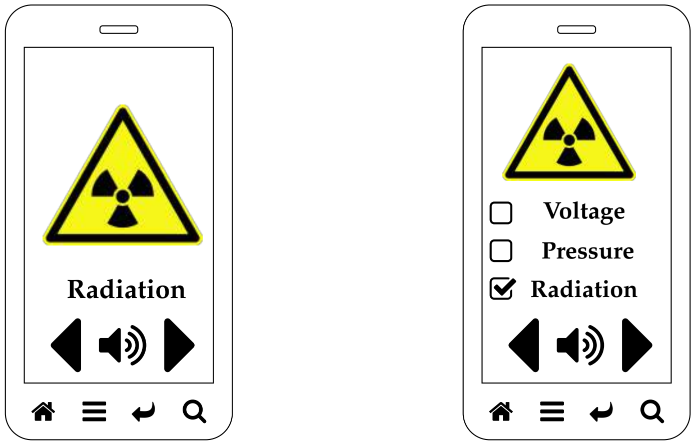
FIGURE 3.1: Agile MDD process in action: fine-grained evolution steps

In this approach, the mobile application developer is required to inspect the generated prototypes and then extend them to incorporate new features. Therefore, it is essential that generated mobile applications are working software systems and that the generated code is of good quality, i.e., well-structured and easy to understand. As an aid to support the comprehension of the generated code, we provide a mapping between domain-specific modeling language elements and the individual code generator templates involved in implementing these elements. In our experience, such a mapping has proven itself valuable. **3.11**

Example (Language design and development iteration). To illustrate an evolution step, we implement an eLearning application for safety instructions. The eLearning application, illustrated in Figure 3.2, comprises two use cases: the first use case, called *learning mode*, concerns learning using different media types (e.g., videos, pictures, and sound recordings). The second use case, called *testing mode*, allows practicing learned content using assignment tasks. **3.12**

The developed metamodel includes style settings and generic page types serving different purposes. For example, there is a *ViewPage* for displaying objects and an *EditPage* for modifying them. To offer the eLearning functionality, we introduced an *eLearningPage* into the metamodel. As Figure 3.2 shows, the purpose of the *eLearningPage* is to present learning content (*learning mode*) or provide a self-test format (*testing mode*). The *eLearningPage* hides the technical details (e.g., playing the sound file, loading media files) from the mobile application modeler. Adding the *eLearningPage* was the only domain-specific modeling language extension required to implement this application. **3.13**

After having extended the domain-specific modeling language, the graphical model editor was regenerated. The code generators had to be then adapted to the new language elements. In order to process the new *eLearningPage* element, a new template (*eLearningPageGenerator*) was added to the generator. This template initially generated an empty Android activity or iOS view. We then extended the empty mock class with the required code. After testing, we abstracted the inserted code to code templates. The iteration ended when the regenerated application fulfilled **3.14**



(A) Learning mode (B) Testing mode

FIGURE 3.2: eLearning application for a safety instruction

the same requirements as the extended prototype. The prototype is then no longer required. □

3.15 Various meta-tools allow specifying model editors, transformations, and further tools. GMF, Xtext, and Sirius support high-level specification of textual and graphical model editors. ATL [58], Henshin [66], ViaTra [VB07], and many more support the specification of model translations, simulations, and optimizations. There are further meta-tools for IDE components such as EMF Refactor [63] for model quality assurance, and EMF Compare [62] and SiLift [Keh+13] to support version management features. Since continuous language evolution results in continuous IDE evolution, co-evolution processes are important to be considered and supported by tools. Therefore, meta-tools are needed. All dependent artifacts such as instance models, model transformations, especially code generators, model editor specifications, model quality assurance, and version management tools can be migrated with those tools. Future research is needed to automate these migrations.

3.1.3 Migration of App Models

3.16 Since app models are directly dependent on the evolution of their domain-specific modeling language, they have to be kept consistent with the domain-specific modeling language. One possibility is to only make changes that do not necessitate adapting the software systems on lower layers. However, this might lead the solutions to get compromised in language design. The alternative is to migrate them accordingly, hence to allow the free development of domain-specific modeling languages.

3.17 Co-evolution tools such as Edapt [60] and Flock [Ros+10] are available, but they still show some limitations: for instance, Edapt supports the evolution of metamodels using predefined operations and the automatic deduction of a suitable migration script for all instance models. However, integrating these predefined operations requires a significant adoption of existing modeling workflows and tools. Consequently, migration processes are currently performed manually, which can be tedious and error-prone. In the future, we aim to provide tool support for the automated co-evolution of app models. We intend to base these tools on results concerning the co-evolution of language metamodels and instance models [Gru+07] [Cic+08].

3.2 Three-Tier Agile Process Model

From a global perspective, considering all processes and tools involved in the model-driven development of domain-specific IDEs leads to three tiers of software development: the development of concrete applications, the development of domain-specific IDEs for model-driven development, and the development of meta-tools to specify IDE components. Readers should carefully differentiate between these various tiers of development. For example, the generation of a model editor from a metamodel using meta-tools should not be confused with the generation of a mobile application from an app model by the domain-specific IDE. **3.18**

Although the main contribution of this thesis is a domain-specific IDE for model-driven development of mobile applications, we also state that concrete applications and meta-tools shall be developed based on agile principles as well. To quickly respond to new user demands and technologies, all involved software systems should be developed continuously, their development incorporating short feedback cycles based on running software. **3.19**

This set of requirements leads to the stipulation of a three-tier agile development process model, as outlined in Figure 3.3. In the domain of mobile applications, for example, a concrete application is a mobile application that is developed using an IDE for model-driven development of mobile applications. Meta-tools such as model editor generators or model-to-code transformation approaches can be used to specify model editors and code generators of these IDEs. **3.20**

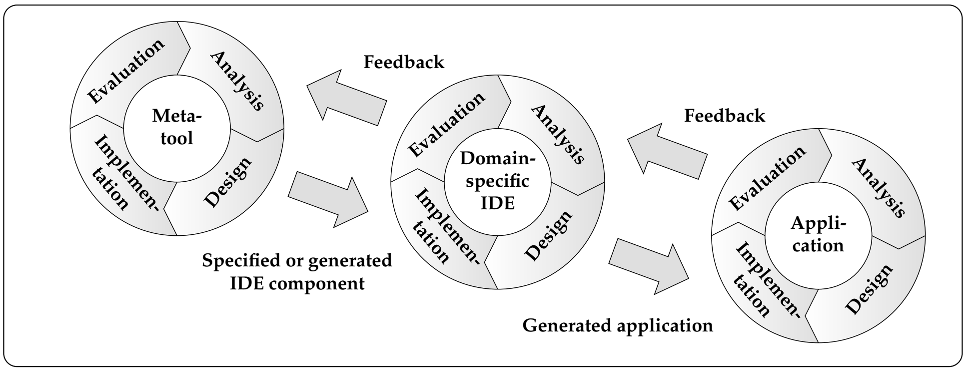


FIGURE 3.3: Three-tier agile software development process model

The interplay of three different kinds of software projects leads to challenges: changes in one software project can affect the other projects. These challenges are intensified by different life cycles and change frequencies. While applications are quickly developed by the model-driven development process, IDE development is much slower, and meta-tools are usually developed completely independently of concrete IDEs. **3.21**

3.3 Demonstration

The preceding sections demonstrate an iteration of a process that claims to support infrastructure developers in developing a domain-specific model-driven IDE under constantly evolving modeling languages. Referring back to the initial research question "How can domain-specific IDEs be developed systematically in the presence of model language evolution?" we will evaluate this research question and the corresponding contribution by answering two sub-questions: given the proposed process of agile bottom-up development of domain-specific model-driven IDE development, we are interested in a) whether the process is applicable at all (Question 1) and b) how useful is such a process (Question 2). **3.22**

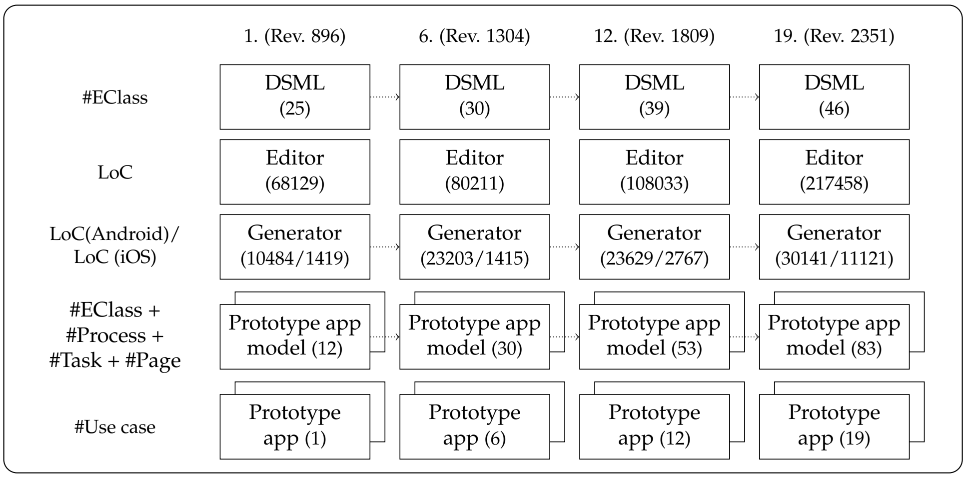


FIGURE 3.4: Continuous language, IDE, and prototype extensions

3.3.1 Setup

3.23 We evaluate our research questions based on the data collected during our research project (cf. Chapter D.1), thereby we have no controlled experiment with external test subjects. We collect data (from 19 iterations) from our own research project's code repository system. Figure 3.4 shows four milestones achieved during the development of the model-driven development infrastructure. We can see the incremental growth of both the domain-specific IDE and the prototype (generated from the app model). Each iteration realizes an additional use case. During the first six iterations (Rev. 1304), the core functionalities were implemented, followed by the enhanced functionalities. We changed the domain-specific modeling language 26 times within a period of about 18 months to cover the 19 use cases of two reference applications (cf. Sections D.2 and D.3). Most of the changes were pure extensions of the domain-specific modeling language (from 25 up to 46 elements), but they often affect existing language elements. We developed several example mobile applications of different kinds simultaneously through the course of the development of the domain-specific IDE as case studies.

3.3.2 Applicability of the Approach (Question 1)

3.24 In order to show the applicability of the proposed agile bottom-up development process for domain-specific IDEs, we want to emphasize that during the complete implementation of the model-driven development infrastructure only the proposed process was applied. The resulting model-driven development infrastructure is indeed no product of an incremental non-agile development process, which is the state of the art. Based on the *early delivery* of the model-driven development infrastructure after the first initial revision (Rev. 896) the infrastructure developers get a *response* from the users, i.e., mobile application developers. This response (cf. Section 6.7 for an example) leads to new requirements or refinements of the existing requirements. In order to react to these requirements, the infrastructure developers must act in an agile way. Moreover, other infrastructure extensions such as eLearning features (cf. Section 12.3.2) as well as online and offline capability of mobile applications (cf. Section 14.5.2.1) were not an initial goal of the research project and subsequently added. The implementation of these additional features requires steady adaptation and refactoring of existing components in an agile way. During the implementation of the model-driven development infrastructure for mobile applications, the different infrastructure developers apply agile methods. In this way, they follow the agile bottom-up development process for domain-specific IDEs. To sum up, by using our proposed methodology to develop our own IDE used for our research, we demonstrate that at least one larger MDD infrastructure has been successfully developed using this approach.

3.3.3 Usefulness of the Approach (Question 2)

Regarding the second research question, we want to discuss how beneficial the proposed agile bottom-up development process for domain-specific IDEs is. Unfortunately, we cannot compare our agile bottom-up development process to non-agile bottom-up development processes due to deviations regarding the considered domain, project size, and many other factors. **3.25**

However, through the acquisition of the key data (e.g., model elements of the DSML, lines of code, number of realized use cases, etc.) during our research project (see Figure 3.4), some conclusions can be drawn. We assume that the model-driven development infrastructure's artifact sizes can be compared and look similar to the artifact sizes at the end of a (typical) non-agile development process. We found the early delivery of the model-driven development infrastructure compared to a non-agile process to be very useful. In particular, the early delivery of the graphical concrete syntax inside the graphical model editor has proven useful. Non-agile methods are limited to known facts that are given in the literature (e.g., design guidelines) and the experience of technical experts. In addition to this, our approach benefits from the mobile application developers' opinions and suggestions. Another advantage of an early delivery in the area of model-driven infrastructure development is the provision of test models. Test models can be created with the already working model-driven development infrastructure in order to test the code generators. **3.26**

Even if the same result can be reached by a non-agile process, the agile method allows the infrastructure developer to focus on the current task and reduces the overall complexity of development process. In contrast to traditional application development tasks, the development tasks of infrastructure developers are more complex. A typical task consists of many steps (introducing a new language element in the domain-specific modeling language, modifying and extending the graphical model editor, modifying and extending the code generators, extending and changing existing test models, testing the generation process, and testing the runtime behavior of the mobile application) and thus the agile and incremental development process relieves an infrastructure developer. During a typical incremental step, approximately 4,000 lines of code are added to the model-driven development infrastructure. **3.27**

We evaluate the use of meta tools and found a positive impact on productivity. After each domain-specific modeling language modification, the graphical model editor, a key component of the domain-specific IDE, was regenerated. Figure 3.4 shows the growth in size of the graphical model editor and the Android code generator: in terms of lines of code, the graphical model editor was at least three times larger than the code generator during all iterations. The generation of this large percentage of the codebase helped to shorten the development cycles: changes to the domain-specific modeling language were immediately available in the graphical model editor. This is very useful according to the different infrastructure developers who worked on the model-driven development infrastructure. Instead of spending a lot of time to become familiar with the architectural structure of the graphical model editor, they could make changes to the domain-specific modeling language after a short introduction and generate the graphical model editor anew immediately. This enabled many infrastructure developers to contribute over the course of the research project. **3.28**

Chapter 4

Requirements for Model-Driven Development of Mobile Applications

The majority of research products of this thesis are components of a software system, here the model-driven development infrastructure for mobile applications. The corresponding requirements must be defined both to specify the system upfront and to evaluate the system at a later stage. To avoid a subjective bias introduced by using requirements defined by ourselves, we ask domain experts from industry for their support regarding the requirements of the model-driven development infrastructure. As part of a research project[1], they discuss and define the requirements for a model-driven development infrastructure. **4.1**

As well as collecting requirements in cooperation with these domain experts, we review existing mobile applications from our project partner from industry and extract general features of these manual-coded mobile applications. Our model-driven development approach should be able to provide the features we thus identify. At the time of this requirement elicitation, we only conduct a high-level review of these existing mobile applications[2]. However, the same mobile applications later serve as reference applications for code extraction (cf. Section 7.1). **4.2**

In difference to a traditional non-iterative application of requirement engineering that delivers a very detailed and complete requirement specification upfront, we could not specify the system completely in advance. The main reasons for this are the complexity and the high number of specifications needed to describe the model-driven development infrastructure completely. Hence, the domain experts contribute just the following requirements to frame the desired model-driven development infrastructure. They also provide the mentioned reference applications as a kind of running specification of the desired output of the model-driven development infrastructure (i.e., generated mobile applications). It was agreed that the used software development process model (cf. Chapter 3) should provide steps to refine the requirements and evaluate intermediate results (e.g., the domain-specific modeling language) in an agile way. **4.3**

The following three paragraphs present the requirements according to the *modeling language*, the *architecture* of the generated mobile applications, and the *tool*s which should be implemented. These requirements focus only on the model-driven development infrastructure, while further requirements dealing with the context support of mobile applications are presented in the second part of this thesis (cf. Chapter 10). **4.4**

4.1 Modeling Language Requirements

We generally require a meta-modeling approach in order to define the domain- **4.5**

[1] Refer to Section D.1 for the key data of the research project.
[2] Refer to Sections D.2 and D.3 for a short explanation of the considered mobile applications.

specific modeling language for mobile applications. We use the term *meta-model* synonymously with the domain-specific modeling language to be designed for the domain of mobile applications. The following three aspects sketch the content and purpose of app models described with the domain-specific modeling language:

4.1.1 Detailed Data Modeling

4.6 **Description:** In order to design mobile applications for different domains, the domain-specific modeling language should facilitate the modeling of an application-specific data model.

4.7 **Explanation:** The domain experts require a mobile application to have an individual and non-generic data model similar to one of the applications used later as a reference application. The advantage of an individual data model over a generic data model is that user entries are better structured and can be validated automatically. Nevertheless, the question arises whether the data model should be fixed before runtime (at design time) or flexibly configured while the mobile application is already deployed. The latter variant poses serious problems because allowing the definition of the data model to change at runtime requires the migration of data and other dependent artifacts (e.g., generated graphical user interfaces, database schemes). Hence, the domain experts agree that the data model is a *detailed design model*.

4.8 **Acceptance:** The requirement is fulfilled if the domain-specific modeling language – especially the data modeling part – allows an appropriate modeling of various domains where the mobile application should be used. The targeted domains are eLearning, touristic and educational applications (e.g., digital guides for conference, exhibition and museum visitors), as well as environmental and manufacturing information systems.

4.1.2 Abstract and Detailed Behavior Modeling

4.9 **Description:** Similar to the detailed modeling of data, the domain-specific modeling language should provide application-specific behavior modeling in order to specify application logic. However, the individual modeling of standard behavior (e.g., CRUD[3]) should be optional.

4.10 **Explanation:** The ability to model custom behavior is an essential requirement to create mobile applications with an application-specific function and behavior. However, it should not be necessary for mobile application developers to model standard behavior (e.g., CRUD). Instead, it should be possible to use predefined standard processes accessible by abstract modeling language elements. Besides, the question arises whether or not the behavior model should be interpreted at runtime. Since the paradigm of model-driven development is to translate an app model into an implementation, an exclusive interpretative approach would be contradictory to the paradigm of code generation. Although domain experts favor an interpretative usage of the app model, we agree to change the requirement to support a so-called *hybrid* approach. In this approach, mobile application developers can model individual behavior inside a design model. Afterward, using the model-driven development approach the code generators will translate the behavior model into a static implementation. The *providing users* (cf. Section 1.4.1) might use a runtime behavior model and determine which parts of the implementation should be instantiated or not, but they cannot add additional functionality to the generated mobile applications. A behavior model can be an *abstract* or *detailed design model* since it facilitates both detailed modeling and modeling of standard behavior. If

[3] CRUD stands for Create, Read, Update, Delete, and describes the generic set of operations that can be performed on an entity of the data model.

mobile application developers require only standard behavior (e.g., CRUD) and standard design, they can use abstract modeling elements. In turn, application-specific behavior (e.g., calculating an order discount) can only be modeled with non-abstract model elements, i.e., detailed modeling of the application-specific process is required. Additionally, it can be used as a *runtime model* with the mentioned limitations.

Acceptance: The requirement is fulfilled if the domain-specific modeling language – especially the behavior modeling part – provides individual process definitions. The defined processes must be configurable at runtime as described before. The domain-specific modeling language should provide abstract modeling elements with a standard behavior (e.g., CRUD). **4.11**

4.1.3 Abstract Graphical User Interface Modeling

Description: Mobile application developers should be able to model the graphical user interface (GUI) of a mobile application in an abstract way. Since the abstraction level is very high, mobile application developers only specify the purpose of a graphical user interface, e.g., editing an object, showing a map, and taking a picture. **4.12**

Explanation: Following the opinion of the domain experts, we will not pursue the requirement to develop a novel domain-specific modeling language for specifying graphical user interfaces. Hence, the graphical user interface model will need to ensure that mobile application developers can specify user interaction at a very abstract level. They should only specify which types of dialogs appear within a process (e.g., edit an object, show a list of objects). The navigation between dialogs and the internal structure should be derived automatically from the other model parts, i.e., data and behavior model. However, mobile application developers should be able to define default and individual styles of the graphical user interfaces. With regards to whether the GUI model should be a design time or runtime model, we agree with the domain experts to use this model in both variants. Hence, the GUI model is used as an *abstract design model* as well as a *runtime model*. **4.13**

Acceptance: The requirement is fulfilled if the domain-specific modeling language – especially the GUI modeling part – provides abstract modeling elements for the modeling of the graphical user interface, which can also be interpreted at runtime by the generated mobile applications. **4.14**

The following requirements deal with improper model structures or models with quality defects: **4.15**

4.1.4 Well-Formedness of the App Model

Description: Well-formedness rules restrict improper model structures to prevent incorrect code generation. **4.16**

Explanation: To get consistent app models, well-formedness rules must be defined in addition to the declarative metamodel. For example, they forbid using a white font color on a white background. Before translating the app model into native program code, the code generators should validate the app models automatically with respect to the structural consistency provided by the metamodel and the additional well-formedness rules. The well-formedness rules should be specified in the Object Constraint Language (OCL) in accordance with the object-oriented meta-modeling approach. In practice, an initial set of well-formedness rules will be gradually built up by incorporating insights from improper, but positively validated, app models. **4.17**

Acceptance: The requirement is fulfilled if a mobile application developer does not run into compile or runtime errors due to an improper app model. **4.18**

4.1.5 Model Quality Assurance

4.19 **Description:** Quality assurance in the form of metric functions and model smell definitions, as well as refactoring proposals should be provided for the domain-specific modeling language.

4.20 **Explanation:** Although an app model may be well-formed and consistent, it may still contain quality defects. For example, an extensive data class in the data model leads to a confusing dialog with a lot of input fields. Metrics functions should count this and indicate adverse conditions.

4.21 **Acceptance:** The requirement is fulfilled if a quality assurance technique can be provided for app models.

4.2 Architectural Requirements

4.22 The architectural requirements of mobile applications to be generated reflect the main features of the reference applications. These could be categorized as data-driven, single user applications, which are able to work in a standalone mode. A general architectural feature of the mobile applications to be generated is that they are implemented as native applications.

4.2.1 Data-Driven Mobile Applications

4.23 **Description:** The generated mobile applications should be able to store acquired data permanently.

4.24 **Explanation:** The generated mobile applications focus on data processing. The available entities and their relations are modeled by the individual domain or data model. We require the mobile applications to be generated to reflect this data model and to provide corresponding data structures (e.g., POJOs). Moreover, the architecture should provide an abstract data access layer that provides mechanisms such as serialization and deserialization of application data. This access layer should encapsulate the particular technology used for serialization and deserialization (e.g., file-based, relational local database).

4.25 **Acceptance:** The requirement is fulfilled if the generated mobile applications can serialize the acquired data. The data should be available even after a restart of the mobile application or system.

4.2.2 Single User System with Back-End Access

4.26 **Description:** The generated mobile applications should be able to work either in an offline mode (i.e., permanently offline) or in a permanently online mode. Additionally, mobile users who are usually working offline should be able to acquire existing data from a back end.

4.27 **Explanation:** At first, the mobile applications should be able to operate independently from a network connection. However, it is often required that preexisting data be distributed to different mobile clients. Hence, the mobile applications should be capable of accessing a back end (e.g., a web server) and of downloading data records or media files. In turn, it is not required that mobile users reintegrate modifications to the back end or work concurrently (e.g., as a multi-user system) on the data at the back end. In such a case the generated mobile application must be permanently online. Besides, the generated mobile applications could be equipped with the data prior to their deployment.

Explanation: The requirement is fulfilled if the generated mobile applications are able to work either (i) in an offline mode where data records can be retrieved from a back-end server or (ii) permanently online. **4.28**

4.3 Tool Requirements

Finally, the tool requirements describe the requirements for the tools that support the model-driven development process of mobile applications. The main requirements deal with the modeling and the code generation. **4.29**

4.3.1 Graphical Model Editor

Description: In order to create app models, mobile application developers need a model editor that provides the domain-specific modeling language. A model editor requires a concrete syntax, which can be textual or graphical. **4.30**

Explanation: Based on the domain-specific modeling language, mobile application developers want to create valid instances of this metamodel. This task should be supported by a model editor implementing the domain-specific modeling language and the well-formedness rules. The requirement is to create a graphical model editor, while an alternative approach for a model editor could be a textual model editor. The kind of syntax (i.e., textual or graphical concrete syntax) will be often determined by the kind of desired user group. A textual syntax might be very useful for technically-skilled modelers because they are probably familiar with textual programming languages. However, as we focus on business experts, a graphical syntax visualizes the modeled application in a better way, and developers do not have to learn textual syntax. A graphical model editor requires a graphical concrete syntax, sometimes called *visual language*. The graphical model editor should reflect the different aspects of modeling (e.g., data, behavior, and graphical user interface) and provide a freehand-editing mode (cf. [Bar+99]). Additionally, the graphical model editor should partially provide textual inputs, e.g., for logical expressions inside the process model, because, according to the domain experts, a visual modeling of logical expressions is not very convenient. **4.31**

Acceptance: The requirement is fulfilled if a graphical model editor facilitates freehand-editing, the creation of app models, and the validation of the created app models. **4.32**

4.3.2 Code Generator

Description: One or more code generators should provide the generation of *native* mobile application for specific platforms (e.g., Android, iOS). However, cross-platform solutions do not have to be taken out here. A corresponding code generator could be developed as future work. **4.33**

Explanation: Following the model-driven development approach, the *platform* is no longer relevant if a code generator is available for the targeted software platform. Thus, model-driven development can generally support platform contexts (cf. Figure 1.2). After considering the domain experts and the available reference applications, we decide to support *Android* and *iOS*. The scope of generation comprises all the artifacts of mobile applications for the desired platforms including project structure for the relevant IDE (e.g., Eclipse ADT, Android Studio, and XCode). The mobile applications should be completely generated and directly runnable without **4.34**

manual completion or extension by the mobile application developer. This includes the generation of application logic, the graphical user interface (layouts), a data and persistence layer, and other resources. The generated mobile application should follow the Model-View-Controller (MVC) architecture. Besides, the generated application versions for different platforms (e.g., Android, iOS) should be as similar as possible concerning the architecture and their graphical user interfaces.

4.35 **Acceptance:** The requirement is fulfilled if a representative set of app models has been used for the generation of runnable mobile applications for different platforms.

4.4 Discussion

4.36 Finally, we verify and validate the requirements summed up in Table 4.1. Please note that this table also contains the requirements that will be defined in the second part of this thesis. According to the chosen agile software development process model, we cannot completely pass-through the traditional process phases, such as requirement elicitation, analysis, and validation [Pae+03]. That is because some of the already defined requirements (e.g., architectural requirements) are intended to be abstract and become tangible as soon as they pass the various iterations of the agile bottom-up development process (cf. Figure 3.4). Despite this, the presented requirements can be considered as robust because they were proposed and developed by external domain experts with a lot of experience. However, we will check four general points according to Boehm's guidelines to verify and validate software requirements [Boe84] – (i) completeness, (ii) consistency, (iii) feasibility, and (iv) testability – to certify that the defined requirements offer an acceptable initial description of the system to be implemented. We have to remember that the requirements concern two levels. First, from a bottom-up perspective, we have to consider the requirements that refer to the output of the model-driven development infrastructure, which is a mobile application. Second, the model-driven development infrastructure itself must satisfy several requirements. Therefore, modeling and tooling requirements refer to the infrastructure, whereas architecture requirements refer to the resulting mobile application.

TABLE 4.1: Requirements of the MDD framework for mobile applications

Part	Requirement No.	Aspect	Requirement name
I	4.1.1	Modeling	Detailed Data Modeling
I	4.1.2	Modeling	Abstract and Detailed Behavior Modeling
I	4.1.3	Modeling	Abstract Graphical User Interface Modeling
I	4.1.4	Modeling	Well-Formedness of the App Model
I	4.1.5	Modeling	Model Quality Assurance
I	4.2.1	Architecture	Data-Driven Mobile Applications
I	4.2.2	Architecture	Single User System with Back-End Access
I	4.3.1	Tooling	Graphical Model Editor
I	4.3.2	Tooling	Code Generator
II	10.1.1	Architecture	Support of User Roles (User Context)
II	10.1.2	Architecture	Heterogeneous Device Support (Device Context)
II	10.1.3	Architecture	Interoperable, Multi-User Systems
II	10.1.4	Architecture	Online and Offline Capability (System Context)
II	10.2.1	Modeling	Declaration of Online- and Offline-Capable Data
II	10.3.1	Modeling	Provider Model Editor
II	10.3.2	Tooling	Simulation System

4.37 *Completeness:* Boehm stated that a specification is complete if all parts of a system are presented, and each part is also fully presented according to its requirements.

The initial presentation of the model-driven development process (cf. Section 2.1 and 2.2) indicates all parts of a model-driven development system. Hence, we can say that the requirements cover all the parts mentioned therein. Each part is covered by several requirements, which implies a complete presentation of the requirements according to the component itself.

Consistency: Consistency has two dimensions. First, *internal* consistency needs the requirements not to be contradictory to each other. So far, this is not critical because there is only a small set of requirements. However, while using an iterative approach and adding more requirements in a following iteration, we must always consider the existing requirements. Second, *external* consistency needs the requirements to correctly fit to the referenced specifications, standards, and technologies. For example, the required freehand-editing mode of the graphical model editor must be compatible with the chosen meta-tools for creating graphical editors. Otherwise, the external consistency is violated. **4.38**

Feasibility: Boehm gave a wide definition of *feasibility*. He did not only consider the feasibility of the software system, but also its feasibility in terms of maintainability and changeability. This is as relevant as ever since the domain of mobile applications is very volatile and fast-moving. An agile bottom-up development process deals with the feasibility criterion very well, because it matches the fast-changing domain and the maintainability of the resulting system. **4.39**

Furthermore, Boehm stated that the requirements should be evaluated on the basis of the *human-*, *resource-*, and *program engineering factors*. The *human engineering factor* asks whether the specified system provides a satisfactory way for the users to perform their tasks. Unfortunately, there is little knowledge about how to evaluate the design of a model-driven development infrastructure upfront. Hence, the intention is to deliver the model-driven development infrastructure, especially the graphical model editor, as soon as possible, and refine the requirements based on user feedback, if needed. The *resource engineering factor* is a classical cost factor [BP88] and must be considered while defining the requirements. The last factor, *program engineering*, concerns portability and maintainability. Since we build on an existing framework (Eclipse Modeling Framework – EMF [Ste+09]), we can positively evaluate the *program engineering factor* because the chosen platform is well-known and used widely in industry and academia. **4.40**

Testability: We have to admit that the presented requirements are not specific enough for the derivation of test cases that are appropriate for a pass/fail acceptance test. Besides, they are not complete according to the chosen iterative approach. Despite, every contribution is evaluated separately. Due to the different kinds of contributions, the evaluation methods must be chosen carefully, i.e., the testability and the test approaches might differ. For example, the evaluation of the domain-specific modeling language involves, among other techniques, user interviews and observations because human interaction plays a role when domain-specific modeling languages and the corresponding model editors are developed. On the other hand, accurate code generation from different (randomly created) app models can be tested automatically using unit tests. **4.41**

Chapter 5

Domain Analysis (Model-Driven Development and SE of Mobile Applications)

The term *domain analysis* was defined by Neighbors [Nei80] as "the activity of identifying the objects and operations of a class of similar systems in a particular problem domain." Our problem domain deals with mobile applications and their development. Hence, we should not only consider the features of mobile applications but also the features of existing and future development environments (e.g., a model-driven development environment). **5.1**

Figure 5.1 shows the input and output artifacts of a general domain analysis activity. In the course of this work, we will use most *sources of domain knowledge*, namely *technical literature, existing implementations (reference applications), expert advice*, and *current requirements* for our domain analysis. Based on this analysis, we develop several novel contributions. The output of a domain analysis activity consists of one or more domain models of different types. Domain models might be *taxonomies, standards, functional models*, and *domain languages*. These products of the domain analysis can again serve as an input of a domain analysis, indicated by the cycle in Figure 5.1. Hence, domain analysis is an iterative process, which will be applied again in the second part of this thesis to extract and implement further features of the desired domain (cf. Chapter 11). **5.2**

Based on the different domain analysis approaches listed by Frakes and Kang [FK05][1], we select the feature-oriented domain analysis (FODA) approach proposed by Kang et al. [Kan+90]. The feature-oriented domain analysis approach consists of the following steps: (i) *context analysis*, (ii) *domain modeling*, and (iii) *architecture modeling*. To develop a model-driven development infrastructure, especially at first a domain-specific modeling language, we focus only on the *domain modeling* step and ignore the other steps. One result of the *domain modeling* step is a *feature model* lending the name to the feature-oriented domain analysis. Feature models represent the mobile end-user's perspective of the capabilities of the mobile applications in a domain or the mobile application developer's perspective of available architectures and development techniques. A feature model is very useful to develop the domain-specific modeling language at a later time. The *feature model* will be developed during the following *feature analysis*. **5.3**

According to Kang et al. [Kan+90], the feature analysis has the following five steps: (1) collecting sources of domain knowledge (Section 5.1), (2) identifying features (Section 5.2), (3) abstracting and classifying the identified features as a model (Section 5.3), (4) defining the features, and (5) validating the model (Section 5.5). We join the steps (2) and (4) because feature identification and definition are strongly related. **5.4**

[1] The authors originally worded them as domain engineering approaches.

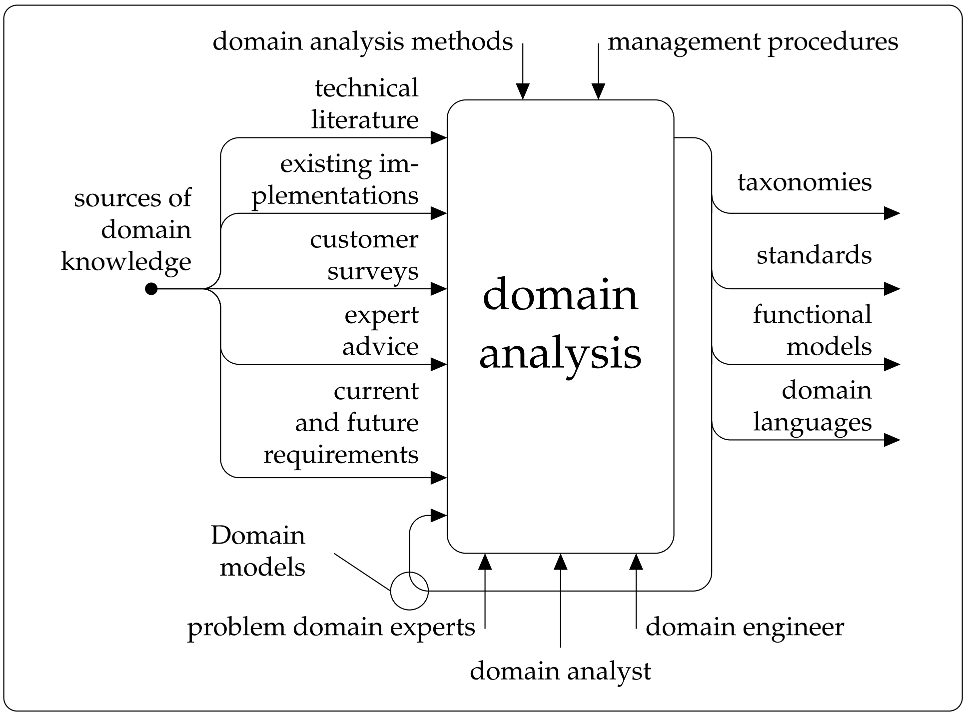


FIGURE 5.1: Domain analysis with input and output artifacts (taken from [PD90])

5.1 Sources of Domain Knowledge

5.5 As Figure 5.1 shows, taxonomies might be an output of a domain analysis. Since the literature already provides several taxonomies related to mobile applications (e.g., [Nic+07] [KEG12] [Emm+13] [Abo+14]), to software engineering of mobile applications/ubiquitous computing (e.g., [Abo+99], [Mod+06], [Lup+09], [Jeo+07], [Mad+02]), and to model-driven development and model transformation (e.g., [MG06], [Deg+14]), we could use them as an input for a feature-oriented domain analysis. Using the literature as a source of domain knowledge will simplify the validation of the feature model later because scientific literature is an external source of knowledge and quite robust against a subjective bias. However, some features are proposed by us independently from the existing literature. Thus, the resulting feature model will reflect more than the state-of-the-art features.

5.2 Feature Identification and Definition

5.6 The feature identification delivers initially only a non-hierarchical and non-interrelated collection of features. Using different sources of domain knowledge sometimes makes a clarification and alignment of the vocabulary (e.g., in a domain terminology dictionary) necessary, but we will not provide such a domain dictionary as the referred literature is homogeneous and generally free of overlaps with respect to the used terms.

5.7 The following feature list starts with a root element followed by the features. We call features with sub-features *feature groups* (Batory [Bat05] calls them *compound features*) and otherwise *primitive features*. Moreover, not all features can be mapped to an implementation artifact because they might be focused on methodical aspects (e.g., the mobile application development process). Following Thüm et al. [Thü+11], we will call these features *abstract* and otherwise *concrete*.

5.8 The following feature definitions listed the main features of a development environ-

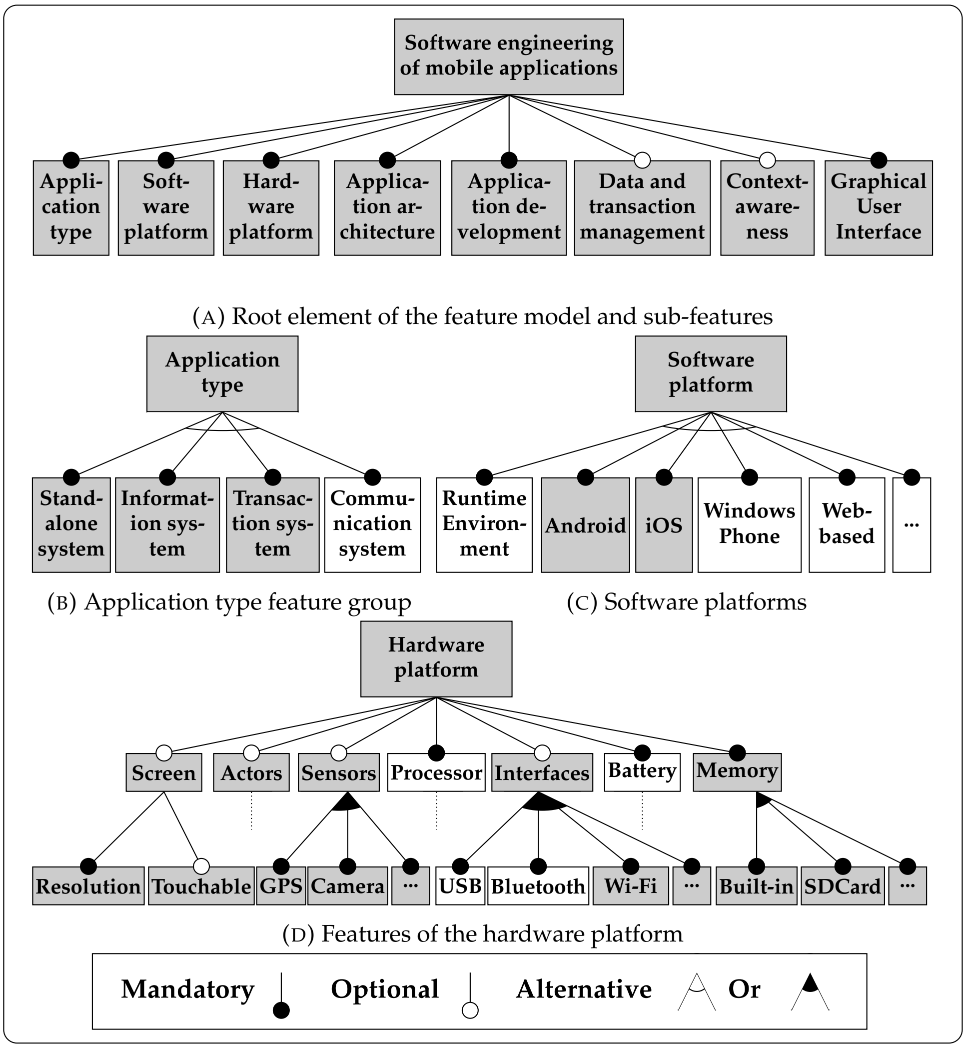


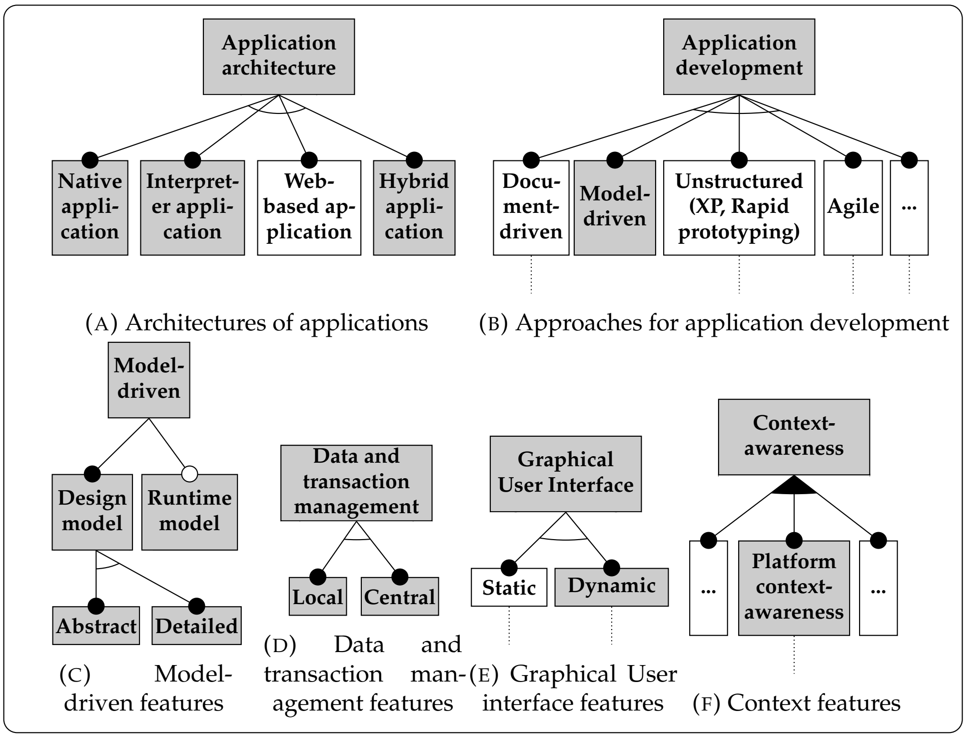
FIGURE 5.2: Feature model (Pt. I/III)

ment for mobile applications and mobile applications.

Software engineering of mobile applications: This is the root element of the feature model. It comprises the features mentioned hereafter that characterize the engineering of mobile applications, as well as the features of the engineered mobile applications itself. **5.9**

Application type: the feature group *application type* holds high-level classifications of the mobile applications concerning the data usage. A *standalone system* (e.g., a calculator application or gaming application) usually does not require any data from other systems. An *information system* feature denotes that a mobile application reads remote data (e.g., passenger information system, dictionary, encyclopedia), while a *transaction system* is an interoperable mobile application reading and writing information from/to a back-end server. Usually this involves more than one mobile application, i.e., transaction systems are multi-user systems. Finally, *communication systems* are also interoperable but require additional real-time services. The items of the feature group *application type* depend on and influence other features (e.g., replication, synchronization of data) as defined by the feature composition rules. **5.10**

Software platform: The choice of the software platform has a major effect on the development process (e.g., available IDEs, programming languages, technologies, available hardware devices). According to the feature group *software platform*, a **5.11**



(A) Architectures of applications (B) Approaches for application development (C) Model-driven features (D) Data and transaction management features (E) Graphical User interface features (F) Context features

FIGURE 5.3: Feature model (Pt. II/III)

mobile application developer may select *iOS*, *Android*, *Windows Phone*, a cross-language *runtime environment* or any *web-based* platform.

5.12 **Hardware platform:** The feature group *hardware platform* is by far the group with the most conceivable features. For a mobile application developer, the available *sensors*, the *screen* size, the available *interfaces*, the *memory* size, and the *processor* are relevant.

5.13 **Application architecture:** The *application architecture* of a mobile application can be *native*, *interpretive*, *web-based* or *hybrid*. Native architectures may also support the interpretation of runtime information.

5.14 **Graphical User interface:** A *graphical user interface* is either *static* or *dynamic*. Static graphical user interfaces cannot be changed at load- or runtime, while dynamic graphical user interfaces can be adapted to device properties or user needs. Additionally, *dynamic* graphical user interfaces may be self-adaptive.

5.15 **Data and transaction management:** The feature group *data and transaction management* provides *local* or *central* data and transaction management. This feature indicates whether a mobile application operates offline or needs permanent network connection (online).

5.16 **Application development:** The *development* of a mobile application is usually no feature of the final product. However, Kang et al. [FK05, Sec. 5.1.4.1] argue that application features can also be implementation techniques. Hence, *document-driven* development and *model-driven* development can be the features of the feature group *Application development*. According to the presented approach, the feature *model-driven* development deals with *design model*s and *runtime model*s. In turn, *design model*s and *runtime model*s can be *abstract* or *detailed*.

5.17 **Context-awareness:** The feature group *context-awareness* comprises different contexts. Within the first part of this thesis, we will only deal with the *platform context* due to complexity reasons. Further contexts will be introduced in the second part of this thesis (cf. Chapter 11).

Platform context-awareness: The feature *platform context-awareness* requires different software platforms if instantiated. **5.18**

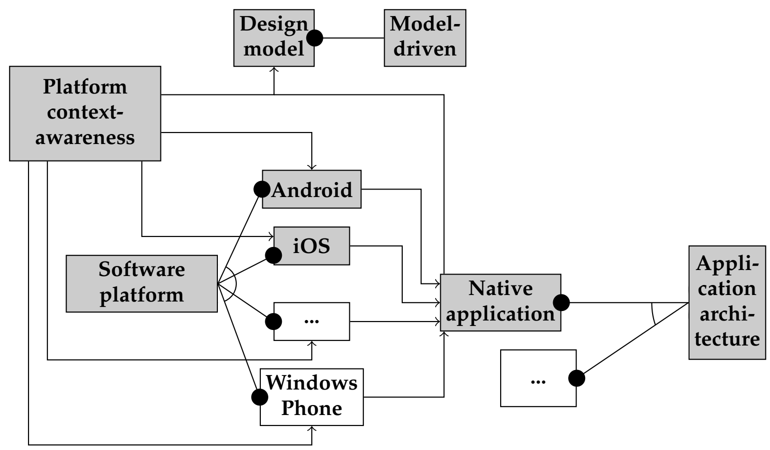
5.3 Feature Model

We propose the hierarchical feature model shown in Figures 5.2 and 5.3 as a domain model reflecting mobile applications and their development. The gray colored feature groups or features form the focus of this thesis. Moreover, the feature analysis will be refined in the second part of the thesis; hence, the feature model of this chapter only shows the first two parts of the overall feature model. **5.19**

5.3.1 Feature Composition Rules

Using the feature model in its initial state may deliver misleading characterizations of mobile applications and development processes because features may still be contradictory. Hence, the features of a feature model need additional relations between each other. A *requirement* relation between two features indicates that a feature requires another one. In turn, an *exclude* relation indicates that a feature excludes another one. Figure 5.4 shows the composition rules which are relevant in the context of this work. The composition rule set will be also refined in the second part of this thesis. **5.20**

For example, the *software platform* feature *Android* requires the *native application* feature. Although it was not proposed in the literature to model relations with a higher multiplicity than one, we found some cases where such a notation is helpful. For example, a *native application* does not necessarily require a *design model* feature. However, *platform context-awareness* and *nativeness* (Native application) can only be ensured by a *design model*, i.e., a *model-driven* development process. **5.21**



(A) Composition rules for the feature groups *Platform context-awareness*, *Software platform*, *Model-driven*, and *Application architecture*

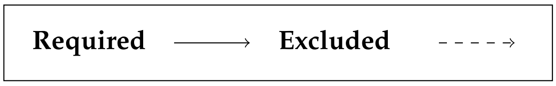


FIGURE 5.4: Feature composition rules (Pt. I/II)

5.3.2 Feature Binding

5.22 Considering the features given in the feature model, the question arises as to when a mobile application developer or mobile user is able to establish or use a feature. Traditionally, feature models describe product configurations of software products that have to be configured beforehand. Hence, the resulting features of the software product are called *compile-time features*, because they can only be instantiated at design time. Another group, *load-time features* might be selected at the start-up time of the application but remain fixed during its execution. As shown in Figure 1.3, our modeling approach should support load-time features. Finally, *runtime features* may be changed during the execution of a mobile application.

5.4 Focused Features

5.23 In this part of the thesis we are heading toward the model-driven development of *native* mobile applications for different *software platforms* (e.g., Android, iOS). Generated mobile applications are kinds of *standalone-*, *information-* or *transaction systems*, rather than *communication systems*. According to the model-driven development approach, the mobile application developer specifies mobile applications by creating an app model at design time (*design model*). The mobile application developer can either use predefined *abstract* model elements (e.g., CRUD) or compose different modeling elements to create more *detailed* functionality that better matches the requirements. This enables rapid prototyping, while still ensuring the possibility of later refinement. Finally, the requirement of working either online or offline leads to a *local* or *central data and transaction management*.

TABLE 5.1: Mapping of the focused feature groups to the requirements of the MDD framework (Pt. I/II)

Part	Requirement No.	Requirement name	Feature group/s
I	4.1.1	Detailed Data Modeling	Model-driven
I	4.1.2	Abstract and Detailed Behavior Modeling	Model-driven
I	4.1.3	Abstract Graphical User Interface Modeling	Model-driven, Graphical User Interface
I	4.1.4	Well-Formedness of the App Model	Model-driven
I	4.1.5	Model Quality Assurance	Model-driven
I	4.2.1	Data-Driven Mobile Applications	Data and transaction management
I	4.2.2	Single User System with Back-End Access	Application type
I	4.3.1	Graphical Model Editor	Model-driven
I	4.3.2	Code Generator	Context-awareness (Platform context), Software platform, Application architecture

5.24 The rationale behind this selection of features is given by the challenges presented in Section 1.2. This work aims to cover issues that are not covered satisfactorily by the state-of-the-art solutions (cf. Chapter 8). Since our first goal consists in providing a model-driven development infrastructure for different native platforms, our primary features are *platform context-awareness* and *model-driven* application development. Since existing model-driven development approaches provide code generation for different native platforms, our model-driven development infrastructure also focuses on different application types (e.g., permanently offline or permanent online mode) with different architectures. Due to the composition rules of the feature model (see Figure 5.4), we have to additionally consider all required features. Moreover, the focused features can be mapped to the requirements given in the preceding chapter and the requirements which will be introduced in the second part of this thesis. Table 5.1 shows which requirement (from the preceding chapter) can be addressed through which feature group.

5.5 Evaluation

In order to evaluate a feature model, there are two approaches: first, Lee et al. [Lee+02, Sec.4] propose several guidelines (e.g., domain selection and scoping, feature identification and categorization, feature organization, feature refinement) to validate a feature model and propose refactorings to improve insufficient feature models. These guidelines guarantee the quality and structural integrity of feature models. However, the characterizing power of a feature model can be tested and ensured only by the second evaluation method: Kang et al. [Kan+90, Sec. 5.1.4.3] propose using domain experts, who are not consulted while creating the feature model, and existing mobile applications, which should be characterized by the feature model. Similar to the domain experts, the selected mobile applications should not be used earlier for the derivation of the particular features of the feature model. **5.25**

While we apply already Lee's guidelines and refactorings already during the construction of the feature model, we can nonetheless report that the feature model was successfully reviewed by external domain experts from academia (cf. Vaupel et al. [Vau+18b]) as well. **5.26**

Chapter 6

Domain-Specific Modeling Language and Model Editors

The core of our model-driven development infrastructure is the domain-specific **6.1**
modeling language captured in a metamodel. In this chapter, we first present the main design decisions that guided us during the design of our domain-specific modeling language for mobile applications. Thereafter, we present the defining metamodel including selected well-formedness rules restricting allowed model structures. To illustrate the domain-specific modeling language, we show selected parts of a simple phone book app model, which will be used as a running example in this chapter. Furthermore, we provide a description of an application programming interface of the domain-specific modeling language that serves as a language library for all further components of the model-driven development infrastructure (e.g., textual or graphical model editors, model quality tools, code generators). Subsequently, the facilities to create app models by a graphical model editor will be presented. Finally, the presented domain-specific modeling language and its model editors are evaluated with respect to the requirements, the design guidelines for domain-specific modeling languages and the guidelines for user interface description languages.

6.1 Design Decisions

Based on the results of our domain analysis, we want to support the generation **6.2**
of mobile applications that can be flexibly configured by providing users. This requirement is reflected in our modeling approach by distinguishing two levels of modeling: *app models* specify all potential facilities of mobile applications, and *provider models* define the actual variants of the mobile applications. We illustrate this general modeling approach in Figure 6.1. While *app models* are used to generate Android and iOS projects (Step 1) to be deployed afterward (Step 2), *provider models* are interpreted at runtime by the generated Android and iOS applications (Step 3). Provider models can therefore be used to change functionality without redeployment of the mobile application. Provider models can be executed in two ways: usually, a provider model is interpreted at runtime, since it does not have to exist at design time. However, it is also possible to make it available at design time by adding it to the resources of its generated mobile application projects. It will then be considered in the build process.

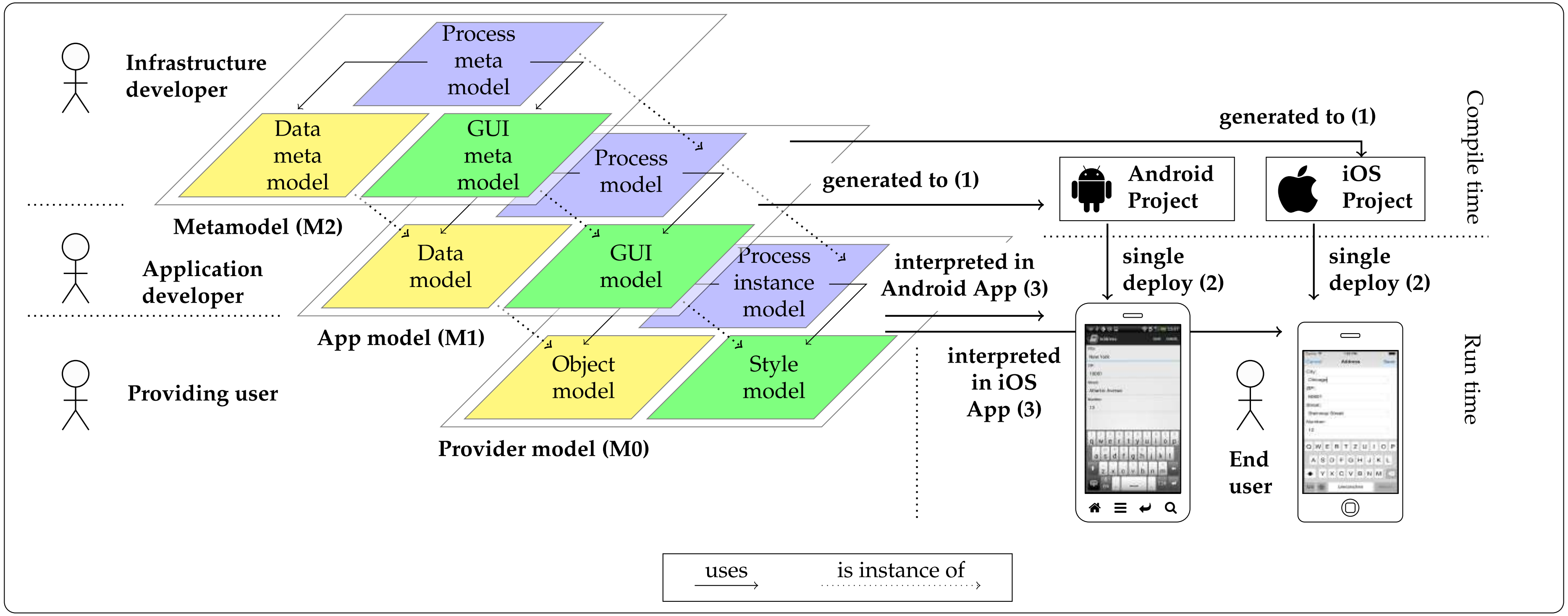


FIGURE 6.1: Detailed multi-level modeling approach

The general approach to the domain-specific 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* that defines the behavior of a mobile application in form of processes and tasks. The data model and the GUI model do not have a direct link. However, the process model includes dependencies to both sub-models (see Figure 6.1) by referring to their elements. The GUI model just contains an abstract definition of pages (see Figure 6.6), instead of detailed layout descriptions. The data model indirectly defines the structure and the default layout of the graphical user interface. We will see later, during the explanation of the code generators, how the data model is used for the automatic generation of the graphical user interface (cf. Figure 7.5). **6.3**

A provider model contains an *object model* defining an object structure as an instance of the data model. It also contains a *style model* defining explicit styles and pages for customized graphical user interfaces. Finally, it contains a *process instance model* selecting a subset of processes and providing them to specify the behavior of the intended variant of the mobile application. Similar to the app model, object and style models are independent of each other but used by the process instance model. **6.4**

For the design of the domain-specific modeling language, we follow the overall credo: "Model your mobile application as abstract as possible and as concrete as needed." This means that the standard design and behavior of a mobile application can be modeled in an abstract way. This can be very useful for the rapid prototyping of mobile applications. A mobile application with a large share of custom behavior needs an app model with more details than a mobile application with a large share of standard behavior. In fact, all special styles, pages, and processes that may be used in the intended mobile application have to be defined in the app model. Since the provider model will be defined by domain experts, these are already completely domain-specific and follow the predefined app model. Provider models support the development of software product lines in the sense that a set of common features are shared and some variability is supported. Differences between mobile application variants are modeled separately by different provider models. **6.5**

The GUI model specifies views along with their purposes, e.g., viewing and editing an object, searching objects from a list and showing search results, doing a login, and choosing a process from the set of available processes. A GUI model is not usually intended to specify the inherent hierarchical structure of graphical user interface components as in rich layout editors like the Interface Builder [Pip10], Android Common XML Editor [GR11], and Android Studio [Zap13]. However, the model can be gradually refined to obtain more specificity in the generated mobile application. Style settings are specified independently from views and follow the same design idea, i.e., the more default the look and feel, the more abstract the model. **6.6**

Processes and tasks covering usual purposes such as CRUD functionality including searching, choosing processes, as well as invoking graphical user interface components, operations and processes. The modeling of custom behavior is supported by more specific model elements such as control structures and assignment tasks. More specific purposes may be covered by utilizing the well-known concept of libraries, i.e., a basic language is extended by language components for different purposes, as is done for LabView [Bis11]. **6.7**

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). However, 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. **6.8**

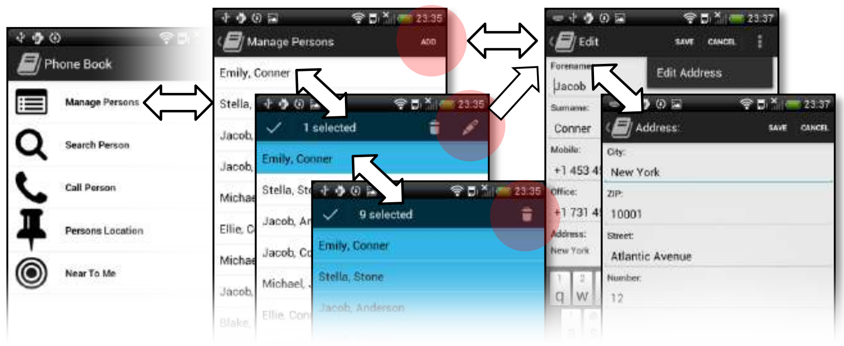
6.2 Eclipse Modeling Framework (EMF)

6.9 Prior to the definition of the domain-specific modeling language by the metamodel, we introduce the Eclipse Modeling Framework on which the metamodel development and the software development process of the model-driven development infrastructure are based. Our development goal is an Eclipse-based model-driven development infrastructure shaped by different plugins created by EMF and further frameworks that are introduced later. Using EMF serves three goals: first, EMF provides meta-modeling facilities that can be used to model metamodels. EMF uses an object-oriented meta-modeling language called Ecore, which conforms to the Essential Meta Object Facility (EMOF). Second, EMF provides a code generation facility for building tools (e.g., Eclipse plugins) and general application code. Tools can be standard model editors such as *tree*- or *table*-based model editors. This generative approach of EMF fits well to the chosen software development process model (cf. Chapter 3). Modifications and extensions of the domain-specific modeling language become effective immediately after a regeneration step of the plugin. While generating code, EMF follows the model-driven development approach (cf. Figure 2.2), i.e., transforms *platform-independent model*s into *platform-specific model*s (cf. *GenModel* model in Figure 6.11). Configuring the latter ones provides a customization on some points. Finally, EMF includes an Eclipse-based runtime component that enables the execution of generated tools (e.g., standard or customized model editors).

6.3 Domain-Specific Modeling Language Definition

6.10 After having presented the main design decisions for our domain-specific modeling language and technological setting, we will now focus on the metamodel and apply the modeling features of EMF in order to create our domain-specific modeling language PIMAR (Platform Independent Mobile Augmented Reality), which is also the project name of the thesis-related research project. While the data model is defined by the original Ecore model, two new Ecore models have been defined to model behavior and graphical user interfaces of mobile applications. We present the metamodel stepwise in accordance with the three sub-models (data-, GUI- and behavior- model part). An instance model is presented subsequently to each sub-model to demonstrate the modeling process of mobile applications. The *app model* of a simple phone book application introduced hereafter serves as the running example application for this chapter. We focus on selected model parts of this example application; the whole *app model* can be found at the end of this thesis in Appendix B.1.

6.11 **Example** (Simple phone book application). One of the core applications for smartphones are phone books for managing personal contacts. In this example, we show a simple phone book application for adding, editing, and searching contact information about persons. Figure 6.2 shows selected screenshots of the phone book application, generated by our model-driven development infrastructure. The arrows indicate the order of views shown. The first sub-figure (Figure 6.2a) shows the main menu leading to a standard CRUD process (*Manage Persons*) to create, edit, and delete persons. The standard behavior and graphical user interface for this task have been generated from a simple CRUD process (see Figure 6.8b), which the mobile application modeler has created before. Instead of using standard CRUD processes, the mobile application modeler may create more customized processes (see Figure 6.8c for a custom creation process) to cover further requirements (e.g., customized navigation and styles). Figure 6.2b shows customized CRUD processes from the mobile user's perspective. These variants can be instantiated at runtime. Moreover, the generated mobile application considers the user location context to find contacts near the current location of the mobile user (Figure 6.2c). Phone numbers are connected with the phone app in such a manner that whenever the mobile user selects a phone number, it automatically starts dialing (Figure 6.2d). □



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

(B) Main Menu with customized *Create*, *Edit*, *Delete Person* Process

(C) *NearToMe* Process

(D) *Call Person* Process

FIGURE 6.2: Screenshots of the simple phone book application

6.3.1 Data Model

Figure 6.3 shows the core elements of the Ecore metamodel. Mobile application modelers can create classes with typed attributes and connect them with different types of references (e.g., inheritance, aggregation, association). Additionally, a given data model is equipped with domain-specific semantics. Data models are not only used to generate the underlying object access facilities but also influence the presentation of data through the graphical user interface. Sub-objects, for example, result in a tabbed presentation of objects, attribute names are shown as labels (if not redefined), and attribute types define the appropriate edit elements such as text fields, check boxes, and spinners. Furthermore, data models determine the behavior of predefined CRUD processes in the canonical way (cf. Figure 7.5). Class and 6.12

attribute names are not always well-suited to be viewed in the generated mobile application. For example, an attribute name has to be a string without blanks and other separators, while labels in mobile application views may contain several words, e.g., “Mobile phone number”. In such a case, an attribute may be annotated by the intended label.

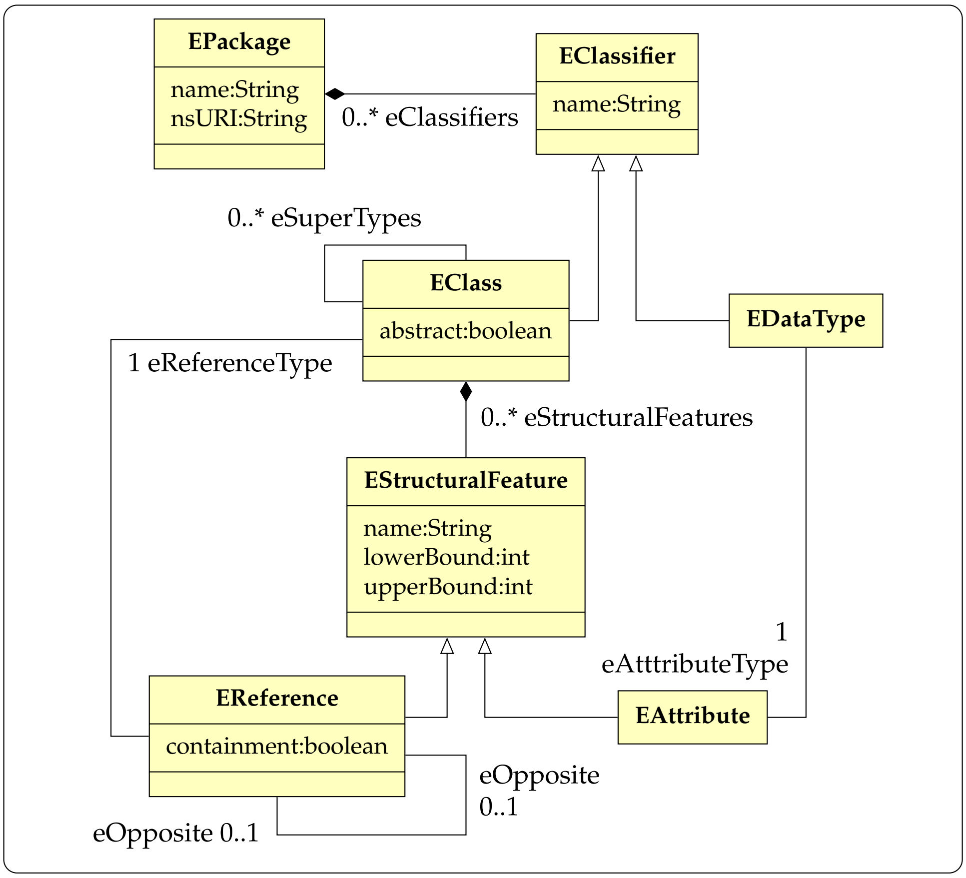


FIGURE 6.3: Ecore metamodel (excerpt) used for data structures

6.13 **Example** (Data model of the simple phone book application). Figure 6.4 shows a simple data model as an Ecore model[1]. Splitting the contact data into the classes *Person* and *Address* has advantages. As generic data views are automatically generated for all classes, it makes sense to minimize the number of attribute per class. This ensures that not too much information is presented in one view. *PhoneBook* is just a container for *Person*s and not intended to be viewed.

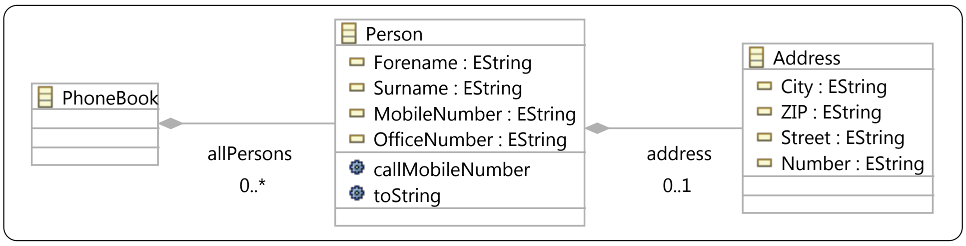


FIGURE 6.4: Data model of the simple phone book application (excerpt)

LISTING 6.1: Injection of custom code (Method `callMobileNumber()`)

```
String uri = "tel:" + this.getMobileNumber();
```

[1] This figure already uses the graphical concrete syntax that will be presented in detail later.

```
Intent intent = new Intent(Intent.ACTION_CALL);
intent.setData(android.net.Uri.parse(uri));
intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
context.startActivity(intent);
```

Listing 6.1 shows a custom code annotation, which implements the method *callMobileNumber()* of the class *Person*. The annotated code is adopted automatically by the code generator. □ 6.14

6.3.2 Graphical User Interface (GUI) Model

The metamodel for GUI models is shown in Figure 6.5. Different kinds of graphical user interfaces are modeled by different kinds of pages (e.g., *ViewPage*, *EditPage*, and *MapPage*). Each of theses pages has a predefined (generic) structure of graphical user interface components and follows a specific purpose. For example, the purpose of the *EditPage* is to edit an object (e.g., Address). Our GUI model reuses the existing data model that holds a description of the objects. Additionally, mobile application modelers can set style and presentation properties. The different style setting elements of our GUI model provide this aspect of presentation. These elements influence the style of *Pages* (*PageStyleSetting*) in general and in particular of *Menus* (*MenuStyleSettings*), *Lists* (*ListStyleSettings*), and *Selections* (*SelectionStyleSettings*). A dialog sub-model (cf. Trætteberg [Træ02]) does not exist in our modeling approach. This conversational aspect of the graphical user interface is covered by pages that implicitly contain the necessary dialogs. 6.15

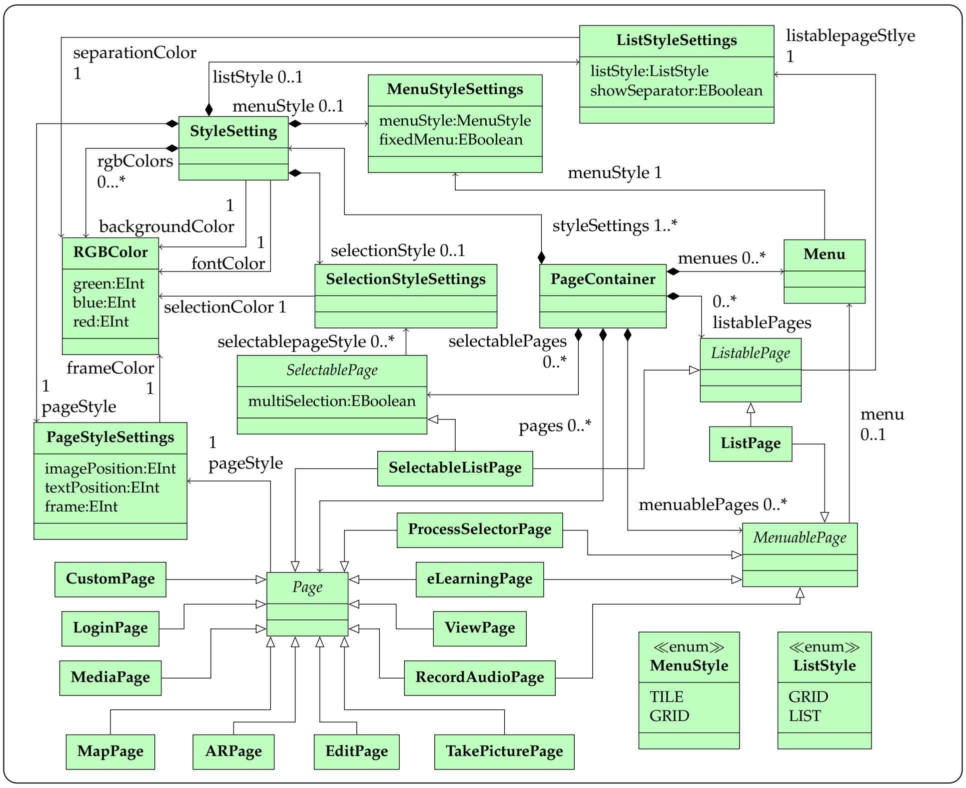
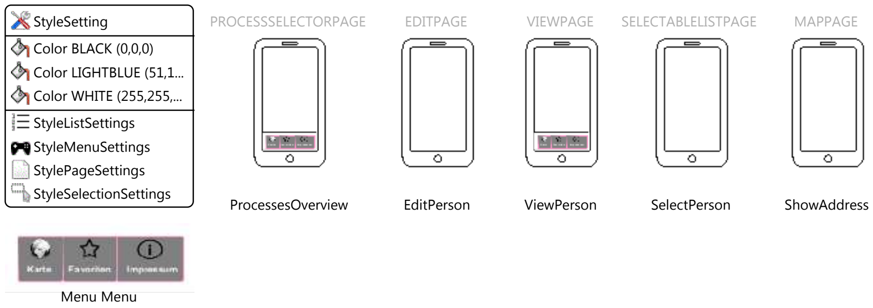


FIGURE 6.5: Ecore model for defining the graphical user interface of mobile applications

6.16 **Example** (GUI model of the simple phone book application). The graphical user interface of our simple phone book application is modeled in Figure 6.6.



(A) GUI model of the simple phone book application

Property	Value
Menu	Menu Menu
Name	ProcessesOverview
Page Style	Style Page Settings 2

(B) Properties of *ProcessSelectorPage*

Property	Value
Name	EditPerson
Page Style	Style Page Settings 2

(C) Properties of *EditPage*

FIGURE 6.6: GUI model of the simple phone book application

6.17 This part of the app model is 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 structure. Besides, the order of invocation and data flow between the pages is not expressed in this model, as they depend on the behavior model. □

6.3.3 Process Model

6.18 Figure 6.7 shows the first part (cf. Figure 6.15 for the second part) of the metamodel for process models of mobile applications. This metamodel is influenced by the language design of BPMN [31] and (WS)-BPEL [39]. Since BPMN does not itself provide a built-in model for describing data structures, we have to reuse the data model provided by EMF. Thus, it is natural to adopt the required BPMN/(WS)-BPEL model elements to our own EMF-based domain-specific modeling language. Many of the BPMN/(WS)-BPEL language elements have been removed (e.g., the error handling of (WS)-BPEL and the events of BPMN). The standard set of behavior constructs is extended by CRUD functionality on the data model, input/output facilities referencing the GUI model, platform-independent permissions, and CRUD privileges, where we do not find any adequate constructs in BPMN and (WS)-BPEL.

6.19 The main constituents of a process model are processes that may be defined in a compositional way. In particular, the modularity and reusability of existing processes requires minimal effort for process modeling. The model element *InvokeProcess* calls a sub-process. When invoking a process, the kind of invocation – *synchronous* or *asynchronous* – has to be specified. Long-lasting processes (e.g., processor-intensive or network-intensive processes) should be marked as *asynchronous*. These processes will be run in the background. Each process has a name and several *variable*s that may also serve as (return) 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 *task*s ordered by typical control structures and potentially equipped with permissions. *Permissions* indicate the required rights (e.g., network, file access, Global Positioning System – GPS) of the mobile applications.

There is a number of predefined tasks covering basic CRUD functionality on objects (e.g., *Create*, *Read*, *Update* and *Delete*), control structures (e.g., *If*, *If-Else*, and *While*), the invocation of an external operation (*InvokeOperation*) or an already defined process (*InvokeProcess*), as well as the view of a page (*InvokeGUI*). While the task *CrudGui* covers the whole CRUD functionality with corresponding views, *Create*, *Read*, and *Delete* just cover single internal CRUD functionalities. *Privileges* can limit object access (e.g., Read-only, Modify, and Modify & Create) of the element *CrudGui*. An *InvokeGUI* task refers to a page defined in the GUI model. The *ProcessSelector* points to all processes that should be available in the main menu of the mobile application (see screenshots in Figure 6.2a and Figure 6.2b first screen from left).

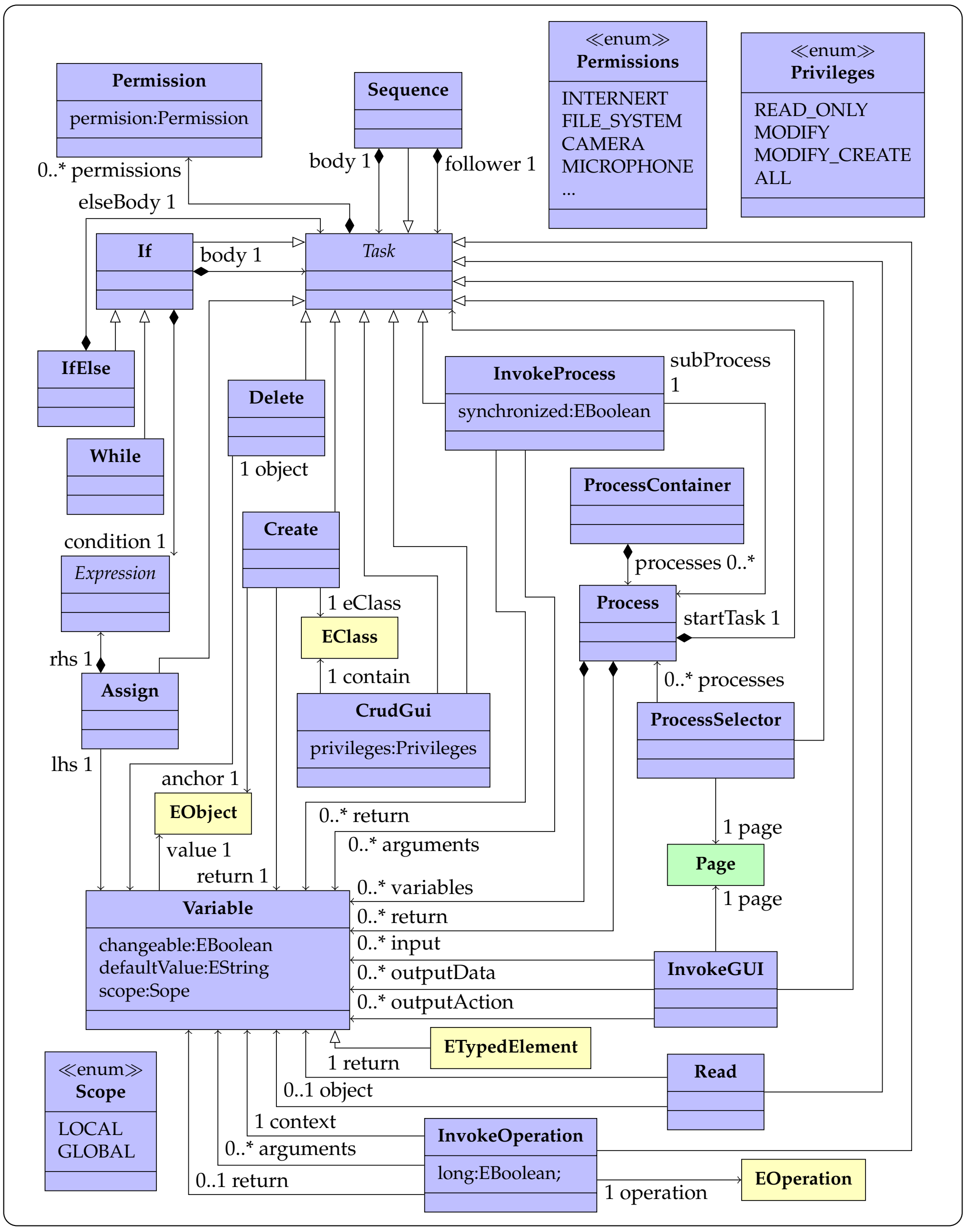
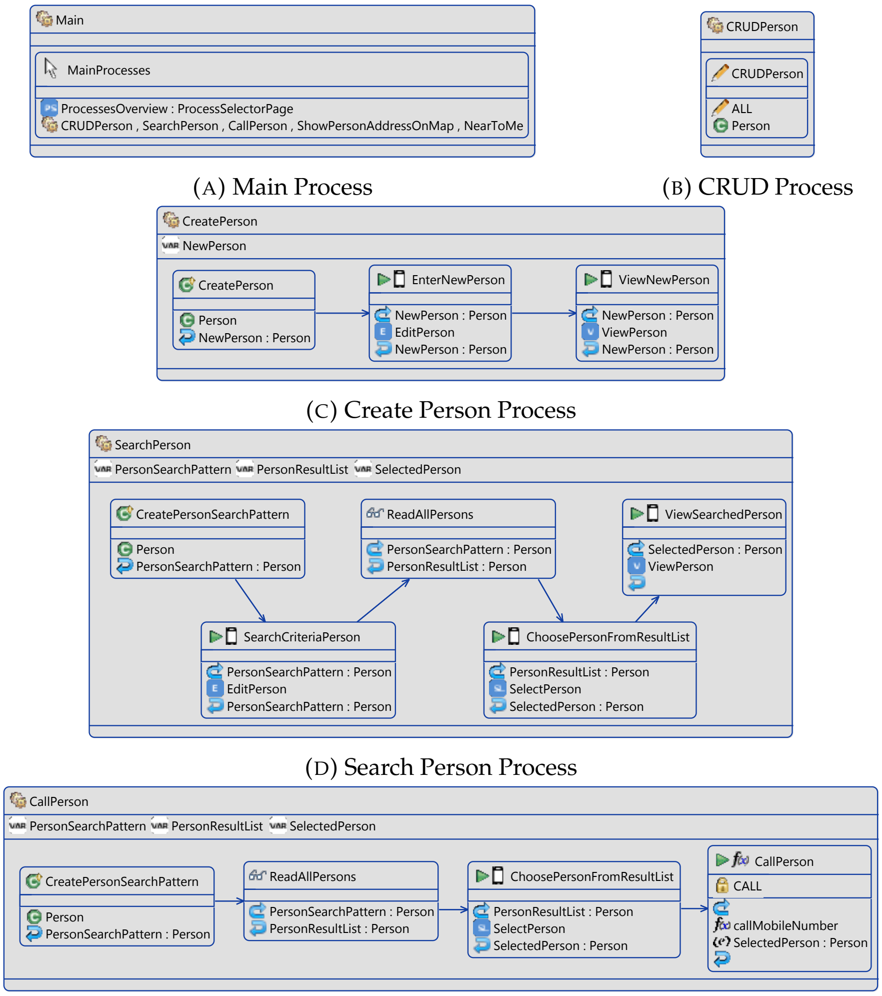


FIGURE 6.7: Ecore metamodel for defining the behavior of mobile applications (Pt. I/II)

6.20 **Example** (Process model of the simple phone book application). The behavior of the phone book application is modeled by a process selector as main process that contains processes for all use cases provided. Figures 6.8a and 6.8b show processes, with *Main* being a process selector and *CRUDPerson* covering the whole CRUD functionality for contacts. Figure 6.8c shows an individual process to create persons. Figure 6.8d shows the definition of a search process where first a search pattern is created that may be edited in an *EditPage* and then it is passed to a *Read* task 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 6.8e shows how to connect to a phone app to call a person. After searching for a person, the operation *callMobileNumber()* is invoked on the selected *Person* object. Just a few lines of platform-specific native program code are needed to start the corresponding Android activity or iOS service. This operation operation is implemented manually. Appendix B.1.3.13 shows the process *NearToMe* defining situation-dependent behavior in the sense that all persons of the phone book with an address near to the current position are displayed.



(A) Main Process

(B) CRUD Process

(C) Create Person Process

(D) Search Person Process

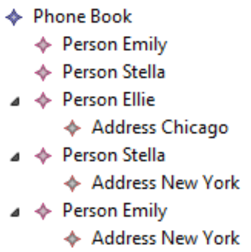
(E) Call Person Process

FIGURE 6.8: Process model of the simple phone book application (excerpt)

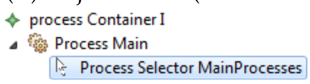
6.21 Since all three metamodel parts are Ecore models, each model element can be annotated to cover additional generator-relevant information or just comments. □

6.3.4 Provider Model

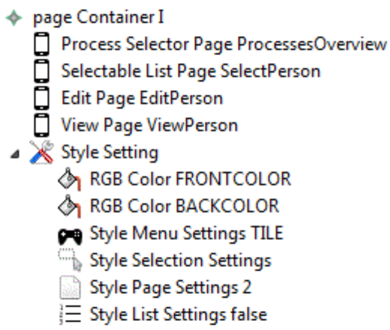
In order to reconfigure mobile applications at runtime, we use provider models 6.22
(i.e., object model, process instance model, and style model), which are interpreted at runtime by the mobile application. The object model is an instance of the data model, which was modeled beforehand by the mobile application developer (see Figure 6.4). In contrast the process instance model and the style model are on the same abstraction level as the GUI model and process model, respectively (compare Figures 6.6, 6.8, and 6.9). To be more precise, the process instance and style model share the same syntax. For example, the style model (if defined) overwrites the setting of the GUI model at runtime. Given that the providing user does not model such a provider model, the mobile application creates an (empty) default provider model from the app model. To the mobile application, it makes no difference whether the provider models are created at design time (and bundled with the generated mobile application) or imported after installation of the mobile application. In turn, the generated mobile applications do not need to be reinstalled to cover device- or user-specific requirements. It is sufficient to load a respective configuration in the form of a provider model.



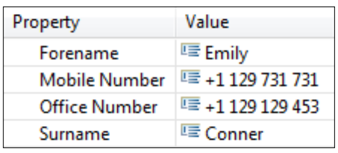
(A) Object model (tree-based editor)



(B) Process instance model (tree-based editor)

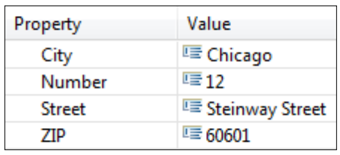


(C) Style model (tree-based editor)



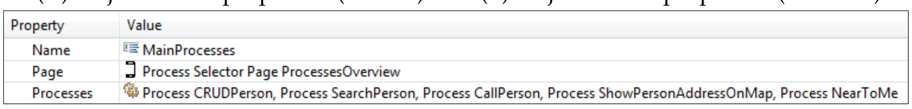
Property	Value
Forename	Emily
Mobile Number	+1 129 731 731
Office Number	+1 129 129 453
Surname	Conner

(D) Object model properties (Person)



Property	Value
City	Chicago
Number	12
Street	Steinway Street
ZIP	60601

(E) Object model properties (Address)



Property	Value
Name	MainProcesses
Page	Process Selector Page ProcessesOverview
Processes	Process CRUDPerson, Process SearchPerson, Process CallPerson, Process ShowPersonAddressOnMap, Process NearToMe

(F) Process instance model properties (ProcessSelector)

FIGURE 6.9: Provider models of the simple phone book application

Example (Provider model for the simple phone book app model). The object model 6.23
of an initial *provider model* contains an empty phone book only; the process instance model just contains the main process. The object model changes whenever the list of contacts is modified by the mobile user. Figures 6.9a, 6.9d, and 6.9e show a non-empty object model as an instance of the data model (see Figure 6.4). Figures 6.9b and 6.9f show the runtime configurations of available processes, which result in the configuration seen in Figure 6.2a. In particular, Figure 6.9f shows the processes that are available at runtime for a certain user group. This set can be rearranged in any way. The dynamic processing of a *provider model* is a crucial feature to meet several requirements that arise with different contexts of use. □

6.3.5 Well-Formedness Rules

6.24 To get consistent app models, we also need a number of well-formedness rules. In particular, the consistency between model components has to be taken into account. The most important well-formedness rules are listed below, expressed in natural language. The complete list of rules formalized as OCL constraints can be found in Appendix A.

1. There is exactly one process with the 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* registered in a *ProcessSelector*, contains – potentially transitively – at least one task of type *InvokeGUI* or *CrudGui*.
4. When invoking a process, the list of arguments has to be consistent with the list of parameters defined for that process with respect to the number, ordering, and types.
5. With respect to the task *InvokeGUI*, the number, ordering, and types of input and output data, as well as output actions must be consistent with the type of page invoked. For example, to invoke a *MapPage*, two Double values are needed as output data; for a *LoginPage*, two strings are needed to show the user name and password, while a Boolean value as output data represents the result of a login trial.

6.25 The rules in Listings 6.2 and 6.3 show the corresponding OCL constraints for the page type *SelectableListPage* (instance of *ListablePage* and *SelectablePage*). A *SelectableListPage* provides the selection of one element (e.g., a person) from a list of the same type (e.g., persons). Figure 6.2d shows such a dialog for choosing a person who should be called. The constraint is divided into the output (Listing 6.2) and input (Listing 6.3) constraint. The output constraint describes the parameters to be passed *to* the page. In this case, the OCL constraint requires a single argument (`size()=1`), which has to be a list (`upperBound=-1`). The input constraint describes the return parameters *from* the page. As expected, the OCL constraint requires a single return value (`size()=1`), which is not a list element (`upperBound=1`). It conforms to the type of list elements (`eType`) shown by the page.

LISTING 6.2: Output constraints for *ListablePage* (Output)

```
context InvokeGUI inv IsListArgument:
self.page.oclIsTypeOf(gcore::ListablePage) implies
self.outputData -> size() = 1 and self.outputData ->
forAll(var:Variable|var.upperBound=-1)
```

LISTING 6.3: Input constraints for *SelectableListPage* (Input)

```
context InvokeGUI inv WellTypedSelectedElement:
self.page.oclIsTypeOf(gcore::SelectableListPage) implies
self.input -> size() = 1 and self.input ->
forAll(var:Variable|var.eType=self.outputData ->
at(1).eType and var.upperBound=1)
```

6.26 **Example** (Well-formedness of the simple phone book app model). Based on the excerpt of the process model, as shown in Figure 6.8, and the GUI model depicted in Figure 6.6, we show the well-formedness of the process *SearchPerson* in Figure 6.8d, particularly of the task *ChoosePersonFromResultList*, which is an *InvokeGUI* task. This task offers a list of results from prior search tasks. The mobile user can choose one result for a detailed view offered by a *SelectableListPage* (defined in the GUI model).

6.27 The mentioned constraints (Listings 6.2 and 6.3) check whether the input type of the

InvokeGUI task *ChoosePersonFromResultList* is a list (of *Persons*) and the output type is a single object of the same type *Person*. In this case, this constraint is satisfied and the code generator can generate a valid initialization of the *SelectableListPage*. □

6.4 Domain-Specific Modeling Language Implementation

The metamodel, shown in Figures 6.3, 6.5, and 6.7, now serves as input of the 6.28
code-generating components of EMF. First, the general application code of the metamodel was generated and results in two Eclipse plugins (second and third rows of Table 6.1). These plugins capture the domain-specific modeling language and additionally perform the (de)serialization and validation of model instances. These generated plugins serve as a language specification for every other plugin that is part of the model-driven development infrastructure for mobile applications.

TABLE 6.1: First set of plugins shaping the MDD infrastructure

Plugin project name	**Content**	**Type**
pimar.language	Metamodel, Model code	manual/generated
pimar.language.edit	Edit code	generated
pimar.language.editor	Editor code	generated
pimar.language.editor.extended	Editor code (custom part)	manual

Furthermore, EMF supports the generation of standard model editors. The plugin 6.29
pimar.language.editor provides facilities to create and modify model instances which conform to the metamodel of our domain-specific modeling language. Figure 6.10 shows generated standard model editors, editing a GUI model (*.*gcore*).

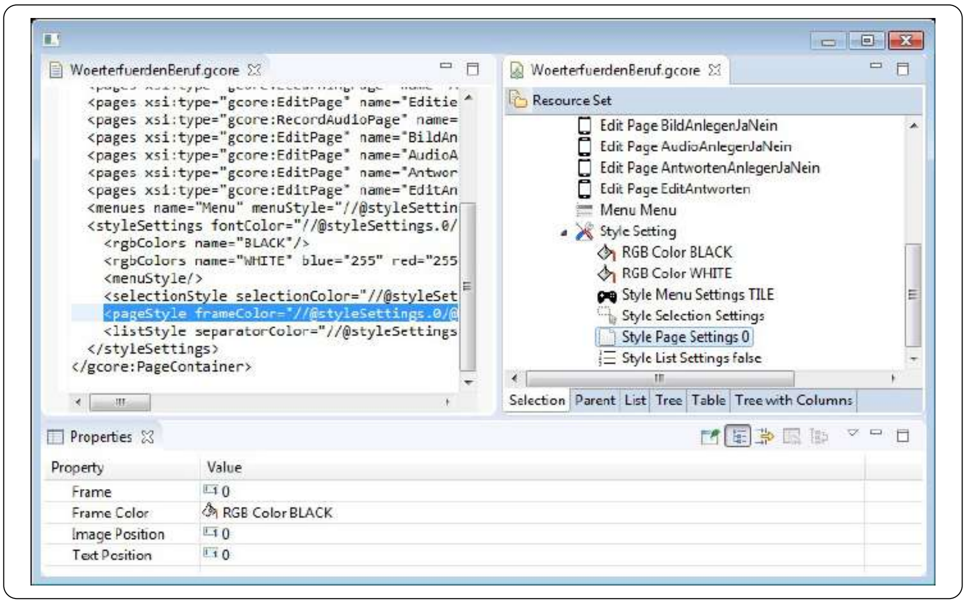


FIGURE 6.10: Standard model editors generated by EMF (showing the GUI model)

On the left-hand side of Figure 6.10, a plain-text model editor is shown to demon- 6.30
strate the XML Metadata Interchange (XMI) representation of a model instance. This model editor could be used for model modifications, but this way of access is only suitable for technical experts. In contrast, the tree-based model editor on the right-hand side of Figure 6.10 is a more convenient way to edit models. This model editor supports simple edit operations (e.g., adding/deleting an element, relocating

elements, setting attributes). The properties view, shown at the bottom of Figure 6.10, allows entering attribute values for selected model elements. The tree-based model editor already provides a validation of model instances.

6.31 Although EMF is a complex model-driven framework, it cannot provide all requirements, which is a general issue in model-driven development. In order to customize the tree-based model editor, an additional plugin, called *pimar.language.editor.extended*, uses the inheritance mechanism of EMF and extends the standard behavior by additional functions. These functions will be explained later in Section 6.7.

6.5 Graphical Modeling Framework (GMF)

6.32 While the presented domain-specific modeling language is based on EMF, the graphical model editor is based on Eclipse's Graphical Modeling Framework (GMF) [Gro09]. GMF is a powerful and widely-used framework for implementing and running graphical model editors for EMF-based domain-specific modeling languages. GMF provides a generative approach, similar to EMF. Beginning with an Ecore model (called domain model) that defines the abstract syntax of the domain-specific modeling language, we derive a set of more detailed models that describe the graphical concrete syntax and the mapping to the abstract syntax. These models can then be processed by the GMF code generators to build the graphical model editor. Figure 6.11 shows the set of involved models, the generation steps, and the resulting plugins that form the desired model-driven development infrastructure (i.e., the modeling functionality). Having been generated by GMF, the graphical model editors internally require the Graphical Editing Framework (GEF) [Rub+11] libraries and runtime. Hence, GMF bridges EMF and GEF. Although we cannot present GEF in detail here, it later becomes necessary to modify the generated plugins in order to provide custom functionality.

6.33 GMF is compatible with different meta-modeling paradigms such as graph-centric and declarative modeling approaches. For example, the Tiger (Transformation-based generation of modeling environments) [Ehr+05] framework applies GMF based on a graph-centric approach. However, the domain-specific modeling language we developed following a declarative approach and thus we apply GMF in the commonly used way.

6.5.1 Workflow

6.34 The general workflow of GMF requires the creation of four models (cf. Di Ruscio et al. [Rus+10, Sec. 1,2], Baetens [Bae11]): (i) the domain model (*.ecore), (ii) the graphical definition model (*.gmfgraph), (iii) the tool definition model (*.gmftool) and (iv) the mapping model (*.gmfmap). Additionally, the so-called generation model (*.gmfgen) is required for the code generation. It also containing platform-specific options. The generation model can be automatically derived from the previously mentioned models. It can also be customized if needed. The numbers inside Figure 6.11 denote the usual order of creation.

6.35 Since we already start with the definition of the domain-specific modeling language (cf. Section 6.3) that is captured in an Ecore model (*.ecore), we will reuse this model as our domain model for the graphical editor.

6.36 The graphical definition model (*.gmfgraph) provides *node*s and *connection*s as basic building blocks and limits the visual languages to a graph-like notation [Min02]. Both *nodes* and *connections* will be arranged on a *canvas*. *Node*s may contain other nodes through a *compartment* relation. *Node*s and *connection*s always refer to a *figure* descriptor that provides standard figures (e.g., rectangles, circles), customized

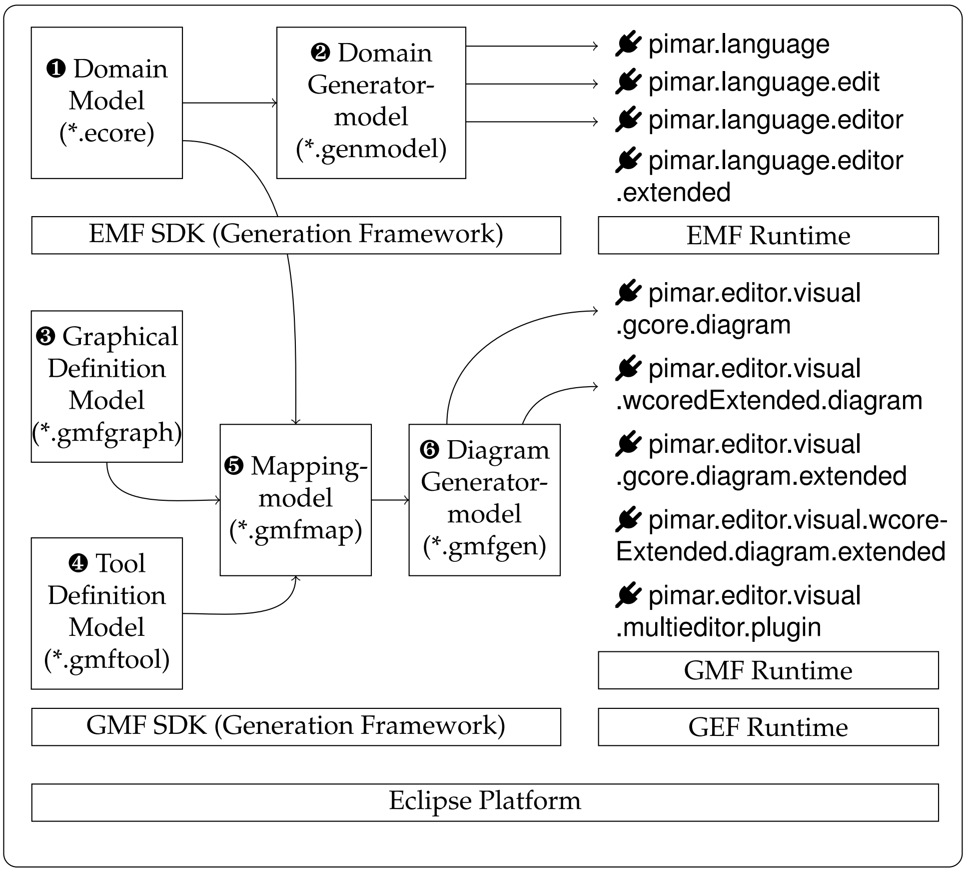


FIGURE 6.11: EMF/GMF process overview and involved models

figures (e.g., polylines), as well as external defined scalable vector graphics (SVG). The *figure* descriptor implements the graphical elements of the visual language. Additionally, the graphical definition model contains diagram labels that decorate *node*s and *connection*s. Usually, diagrams hide some attributes of model elements in order to increase the readability; these attributes can only be changed in a so-called property editor (cf. Figure 6.10). Both the *graphical* and the *tree-based* model editor use the properties editor to set non-visualized properties.

The graphical tool definition model (*.gmftool) defines which entries are available on the palette inside the graphical model editor. This model is constructed in a straightforward way and usually contains the main classes of the domain model. Categories will often divide elements into node types and connection types. **6.37**

The mapping model (*.gmfmap) combines the previously mentioned models. It maps the elements of the domain model, i.e., classes, to *node*s or *connection*s. Attributes will be mapped to labels of the graphical definition model. Besides, the mapping model maps the tooling items from the palette to visual elements. Thus, the tooling items enable the creation of visual elements inside the generated model editor. **6.38**

All of the mentioned models, except the generation model, must be created manually. Kolovos et al. [Kol+10] apply model transformation techniques to support the creation of such models and develop a tool support called EuGENia. However, we could not apply these techniques in our setting: having highly customized graphical language elements requires manual modeling for multiple reasons. **6.39**

Finally, the platform-specific generation model (*.gmfgen) must be generated and configured. Then, the program code of the plugins for the graphical model editor are generated from this diagram generation model. **6.40**

6.6 Graphical Concrete Syntax and Edit Operations

6.41 To provide a graphical model editor, at first we must define the graphical concrete syntax. The graphical concrete syntax is captured in the graphical definition model as shown by the presented workflow (cf. Figure 6.11).

6.42 **Example** (Specifying graphical concrete syntax elements). Figure 6.12 shows an excerpt from the presented models that specify the graphical concrete syntax of the domain-specific modeling language. More precisely, the element *ProcessContainer* is mapped to a canvas via the diagram element. A *Process* can be part (compartments) of the diagram and occurs as a *node* element. In turn, a *Process* contains a *Task* (e.g., *Create*). The *Create* element of the domain model is mapped to an identically named node element of the graphical definition model, which is linked to a figure descriptor. The figure descriptor defines the complete layout (e.g., geometric shape, labels, icons, decorator elements) of the resulting visual element. Single attributes (e.g., name) of domain elements are mapped through label mappings which refer to label providers in the graphical definition model. These label providers refer, in turn, to labels inside a figure descriptor.

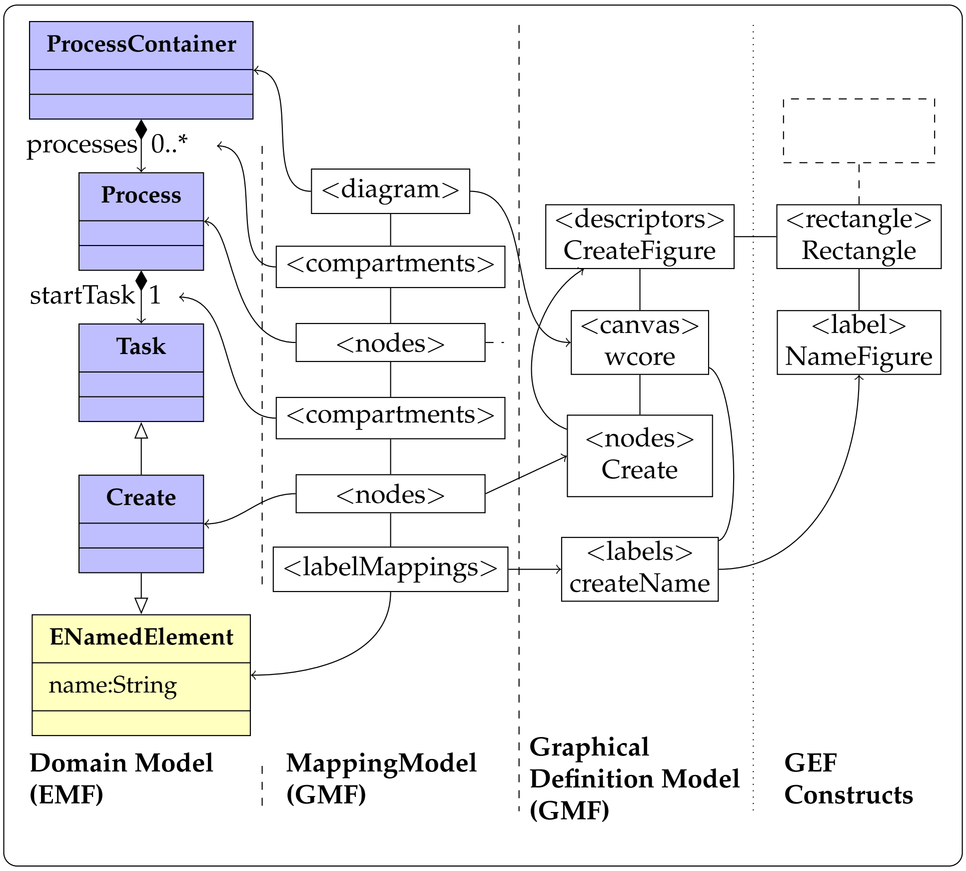


FIGURE 6.12: The mapping between EMF, GMF, and GEF

6.43 Since the notation shown in Figure 6.12 is not very intuitive to understand how the resulting visual elements look like, we will present the graphical concrete syntax based on concrete instances as shown in Figure 6.13.

6.44 The rendered figure descriptor shows the layout, icons, and the static and dynamic labels. The annotations on the right-hand side denote to which element of the domain model the node, label, or compartment refers. Some elements can be edited directly if they refer to simple data types (e.g., string, int, enumeration). If they refer to complex data types (e.g., Create.eClass), their values can only be set by the

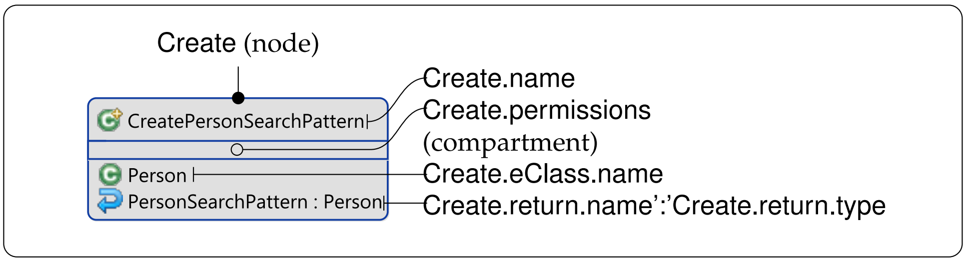


FIGURE 6.13: The graphical concrete syntax for the model element *Create*

properties view, although the values are displayed at the figure. Non-visualized attributes (e.g., *Create.anchor*) can only be set through the properties editor (not shown). □

The introduction of the graphical concrete syntax and edit operations inside the next paragraphs follows the tripartite structure of the domain-specific modeling language: 6.45

6.6.1 Graphical Concrete Syntax for the Data Model

Instance models (cf. Figure 6.4) of the data metamodel follow the same graphical notation as the Ecore metamodel (cf. Figure 6.3). This notation for class diagrams has proven to be very useful and widely accepted for modeling the structural properties of a domain. Hence, we will not present the details of the graphical concrete syntax for this part of app models. 6.46

6.6.2 Graphical Concrete Syntax for the GUI Model

The graphical concrete syntax of the GUI metamodel is very compact because the set of graphical concrete syntax elements is pretty small. Tables 6.2 to 6.12 show the mapping of the abstract syntax (AS) (cf. Figure 6.5) to the graphical concrete syntax (GCS). The GCS type denotes to which graphical syntax type the abstract syntax element is mapped, i.e., *node*, *node* with *compartment(s)*, or *node* with *label*. The GCS example row of the following tables shows an example of the respective graphical concrete syntax element. The context value indicates in which context and how often the graphical concrete syntax element may occur. Finally, the edit operations describe which operations can be performed on the diagram and which settings can be made in the properties view. 6.47

The abstract classes *SelectablePage*, *ListablePage*, and *MenuablePages* are not mapped to graphical concrete syntax elements. The same applies for the enumeration types. 6.48

TABLE 6.2: Mapping of the abstract syntax element *PageContainer*

AS	The abstract syntax element *PageContainer* shapes the diagram canvas and thereby directly or indirectly contains all other graphical concrete syntax elements.	
GCS (Type)	Canvas	
GCS (Example)	not applicable	
Context	none	
Edit operations	**Diagram operations**	The *Canvas* element provides the creation of *Page*, *StyleSettings*, and *Menu* elements.
	Property settings	The properties editor provides no further property settings.

TABLE 6.3: Mapping of the abstract syntax element *StyleSetting*

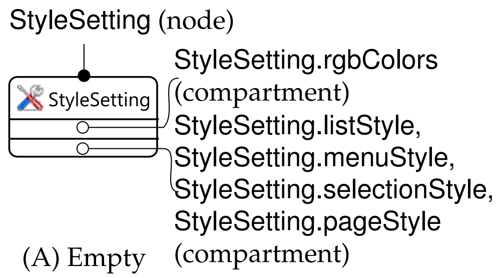
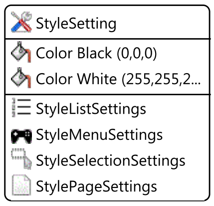
AS	The abstract syntax element *StyleSetting* and its contained elements like *PageStyleSettings*, *RGBColor*, *ListStyleSettings*, *MenuStyleSettings*, and *SelectionStyleSettings* shapes are mapped to the graphical concrete syntax element *StyleSetting*.	
GCS (Type)	Node with compartments	
GCS (Example)	 (A) Empty *StyleSetting*	 (B) Fully modelled *StyleSetting*
Context	Arbitrarily often inside a *PageContainer* compartment	
Edit operations	**Diagram operations**	The *StyleSetting* element provides the creation of all contained sub-elements according to the cardinality given by the abstract syntax model.
	Property settings	The properties editor supports the setting of the features *backgroundColor* and *fontColor*.

TABLE 6.4: Mapping of the abstract syntax element *RGBColor*

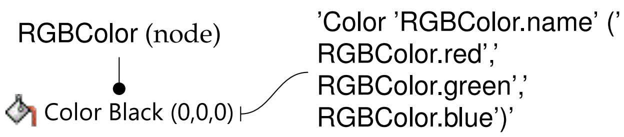
AS	The abstract syntax element *RGBColor* is mapped to the graphical concrete syntax element *RGBColor*.	
GCS (Type)	Node with label	
GCS (Example)		
Context	Arbitrarily often inside a *StyleSettings.rgbColors* compartment	
Edit operations	**Diagram operations**	The *RGBColor* element provides no further edit operations within the diagram.
	Property settings	The properties editor supports the setting of the features *red*, *green*, *blue*, and *name*.

TABLE 6.5: Mapping of the abstract syntax element *ListStyleSettings*

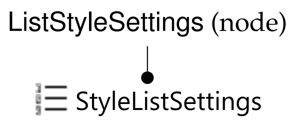
AS	The abstract syntax element *ListStyleSettings* is mapped to the graphical concrete syntax element *StyleListSettings*.	
GCS (Type)	Node	
GCS (Example)		
Context	At most once inside a *StyleSettings.listStyle* compartment	
Edit operations	**Diagram operations**	The *StyleListSettings* element provides no further edit operations within the diagram.
	Property settings	The properties editor supports the setting of the features *showSeparator*, *separatorColor*, and *listStyle*.

TABLE 6.6: Mapping of the abstract syntax element *MenuStyleSettings*

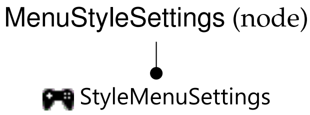
AS	The abstract syntax element *MenuStyleSettings* is mapped to the graphical concrete syntax element *StyleMenuSettings*.	
GCS (Type)	Node	
GCS (Example)		
Context	At most once inside a *StyleSettings.menuStyle* compartment	
Edit operations	**Diagram operations**	The *StyleMenuSettings* element provides no further edit operations within the diagram.
	Property settings	The properties editor supports the setting of the features *fixedMenu* and *menuStyle*.

TABLE 6.7: Mapping of the abstract syntax element *SelectionStyleSettings*

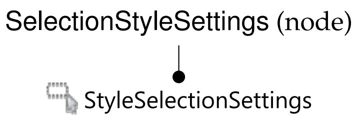
AS	The abstract syntax element *SelectionStyleSettings* shapes the graphical concrete syntax element *StyleSelectionSettings*.	
GCS (Type)	Node	
GCS (Example)		
Context	At most once inside a *StyleSettings.selectionStyle* compartment	
Edit operations	**Diagram operations**	The *StylePageSettings* element provides no further edit operations within the diagram.
	Property settings	The properties editor supports the setting of the feature *selectionColor*.

TABLE 6.8: Mapping of the abstract syntax element *PageStyleSettings*

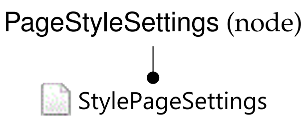
AS	The abstract syntax element *PageStyleSettings* shapes the graphical concrete syntax element *StylePageSettings*.	
GCS (Type)	Node	
GCS (Example)		
Context	Exactly once inside a *StyleSettings.selectionStyle* compartment	
Edit operations	**Diagram operations**	The *StyleSelectionSettings* element provides no further edit operations within the diagram.
	Property settings	The properties editor supports the setting of the features *frame*, *framecolor*, *imagePosition*, and *textPosition*.

TABLE 6.9: Mapping of the abstract syntax elements *Page*

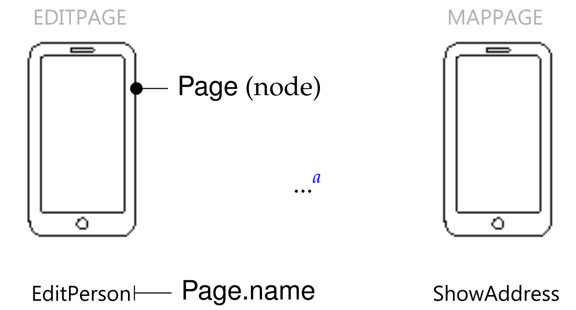
AS	All abstract syntax elements which inherit only from the element *Page* are mapped respectively to equally named elements of the graphical concrete syntax. The static label above the icons indicates the type of the page. The dynamic label below the icon denotes its object name.	
GCS (Type)	Node with label	
GCS (Example)	 (A) EditPage (B) MapPage [a] CustomPage, LoginPage, MediaPage, ARPage, TakePicturePage	
Context	Arbitrarily often inside a *PageContainer* compartment	
Edit operations	**Diagram operations**	The elements provide no further edit operations within the diagram.
	Property settings	The properties editor supports the setting of the *pageStyle* and the *name* property. By default, the first *pageStyle* element in a diagram will be assigned automatically.

TABLE 6.10: Mapping of the abstract syntax element *Menu*

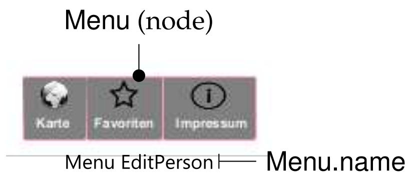
AS	The abstract syntax element *Menu* corresponds to the element *Menu* of the graphical concrete syntax. The dynamic label below the icon denotes its object name.	
GCS (Type)	Node with label	
GCS (Example)		
Context	Arbitrarily often inside a *PageContainer* compartment	
Edit operations	**Diagram operations**	The *Menu* element provides no further edit operations within the diagram.
	Property settings	The properties editor supports the setting of the features *menuStyle* and *name*.

TABLE 6.11: Mapping of the abstract syntax element *MenuablePage*

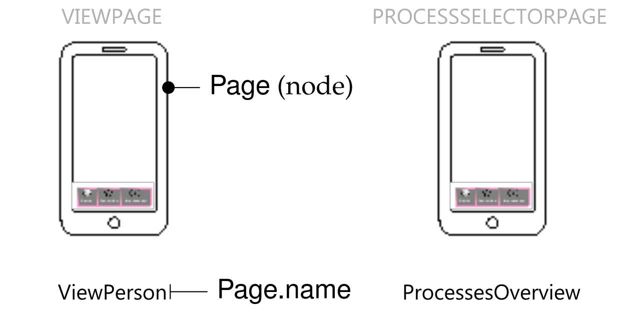
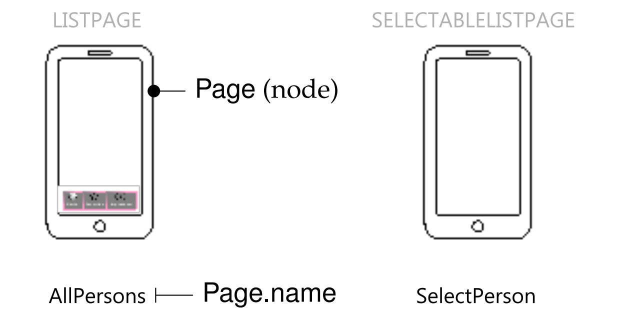
AS	The abstract syntax elements *ViewPage* and *ProcessSelectorPage* are mapped to equally named elements of the graphical concrete syntax.	
GCS (Type)	Node with label	
GCS (Example)	 (A) ViewPage (B) ProcessSelectorPage	
Context	Arbitrarily often inside a *PageContainer* compartment	
Edit operations	**Diagram operations**	The elements provide no further edit operations within the diagram.
	Property settings	The properties editor supports the setting of the features *pageStyle*, *menu*, and *name*. By default, the first *menu* element in a diagram will be assigned automatically.

TABLE 6.12: Mapping of the abstract syntax element *ListablePage*

AS	The abstract syntax elements *ListPage* and *SelectableListPage* are mapped to equally named elements of the graphical concrete syntax.	
GCS (Type)	Node with label	
GCS (Example)	 (A) ListPage (B) SelectableListPage	
Context	Arbitrarily often inside a *PageContainer* compartment	
Edit operations	**Diagram operations**	The elements provide no further edit operations within the diagram.
	Property settings	The properties editor supports the setting of the feature *listablePageStyle* for both page types. The features *selectablePageStyle* and *multiSelection* are only supported for the *SelectableListPage*, while the feature *menu* is only supported for the *ListPage*.

6.49 As apparent by the GCS types in Tables 6.2 to 6.12, the graphical concrete syntax provides only *nodes* and no *connection* types. This is because *connection* types (e.g., arrows between *nodes*) often imply an execution order or processing sequence. Since the GUI model defines neither the internal structure of pages nor their invocation order, we refrain from using graphical syntax elements that could imply this. Relations between objects can only be expressed by assignments in the properties editor (cf. Figure 6.6b and 6.6c) or by using the *compartment* mechanism of a *node*. The latter variant is used for the *StyleSetting* element.

6.6.3 Graphical Concrete Syntax for the Process Model

6.50 This section provides the graphical concrete syntax of the process model and its abstract syntax, respectively (cf. Figure 6.7). While the most of the elements of the abstract syntax can be mapped in a similar way as shown before, some elements can not be mapped to graphical concrete syntax elements in such a straight forward manner.

6.51 As stated by Ehrig et al. [Ehr+05, Sec. 2.2], the disadvantage of the graphical model editor generation based on GMF is that the underlying metamodel (i.e., the Ecore model) strongly influence the structure of the visual language alphabet. As indicated by the already shown mappings, a containment relation in the abstract syntax is always expressed as a compartment construct in the graphical concrete syntax. With respect to the abstract syntax of the *Process* element, mobile application modelers must follow a *containment* structure while modeling a *Process*. At first, they must create a *startTask*, which is usually a *Sequence*, and add *body* and *follower* tasks as compartment elements of the corresponding sequence. The reordering, insertion, and deletion of tasks are not supported by the GMF-generated graphical model editor, making the modeling process very inconvenient. Mobile application modelers usually prefer a *freehand* modeling method, being able to create all *nodes* (e.g., tasks) first and then combining them with *edges* (e.g., body and follower relations). In particular, edges should be flexibly assignable to modify the modeled process.

6.52 In order to circumvent the limitations of GMF, three solutions are conceivable:

6.53 First, editing operations could be implemented to support the reordering, insertion, and deletion of process elements. This could be implemented in a manual or a tool-supported way, e.g., by specifying model transformations [Tae+07]. However, in both cases, the editing operations change the existing layout of a diagram because rearranged model elements will also rearrange the diagram elements. Indeed, editing operations may affect several diagram elements (e.g., all tasks of a process) and thus might not be convenient for mobile application modelers because the layout will be lost.

6.54 Second, the metamodel, i.e., the domain-specific modeling language, could be changed in order to provide a more easy-to-use graphical model editor. This is the most frequently applied solution, especially if the metamodel is not used by other artifacts (e.g., code generators). However, a redesign of the abstract syntax is not favored during this work, because both the graphical model editor and the code generators should have no influence on the domain-specific modeling language and will be developed independently.

6.55 Third, an additional metamodel could be defined to provide a convenient graphical model editor while the domain-specific modeling language definition remains unchanged. This solution requires a model-to-model transformation that serves as a kind of a model parser (e.g., parse visual subsentences – cf. Costagliola et al. [Cos+05b] [Cos+05a]). The additional model is used only for editing and will be parsed to the initial process model. As a result, the graphical model editor and the code generators can be developed independently referring only to their common domain-specific modeling language. In practice, the additional metamodel consists of the existing metamodel and includes some extensions, hence it could called *extended process model*.

6.6.3.1 Extended Process Model

The extended process model, shown in Figure 6.14, contains the classes *Variable*, *Process*, and *Task*, which are derived from the equally named classes in the initial process model (cf. Figure 6.7). The first extension is the containment feature *allTasks* of the class *Process*. This feature provides the creation of multiple tasks inside a process compartment without using a *Sequence* task to combine the tasks. In order to combine tasks, the next added feature *RefTrue* can be used. The *RefTrue* class acts as a directed edge between two tasks and denotes the preceding and succeeding task respectively. The classes *If1*, *IfElse1*, and *While1* correspond to the similarly named classes in the initial process model but require no *body* (or *elseBody*). The *RefTrue* and *RefFalse* classes are used to model the accepting and rejecting path after such conditional tasks. Besides, the introduced conditional nodes and the *Assign1* element accept a textual condition, captured in the *condition* (or *rhs*) attribute. This will be used later to combine textual and graphical editing (cf. Section 6.7). **6.56**

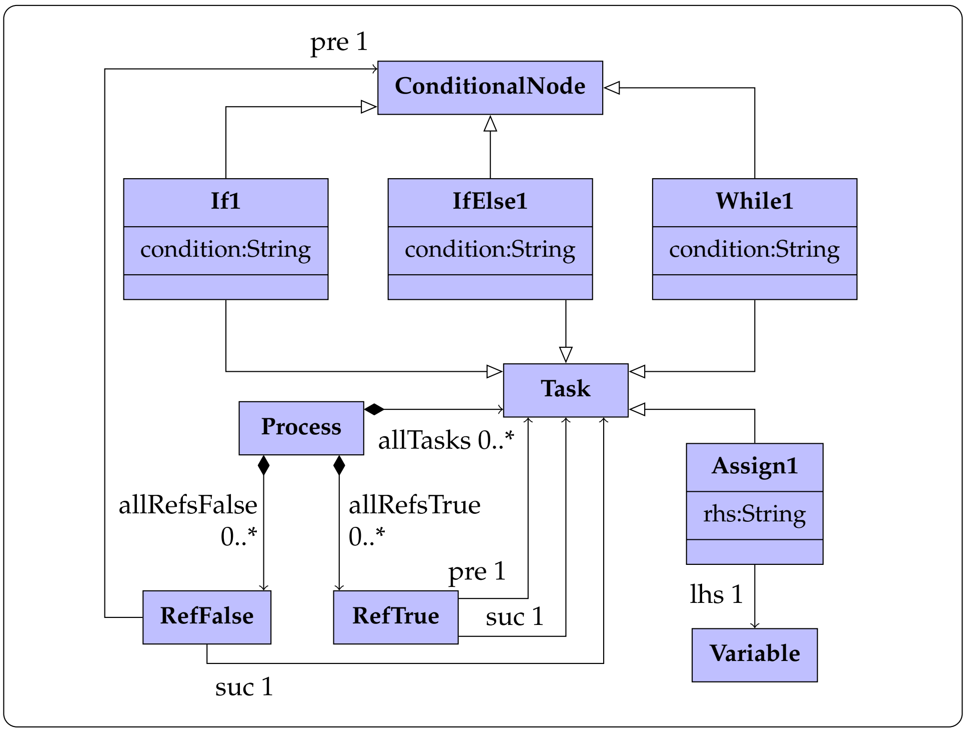


FIGURE 6.14: The extended process model (showing extension only)

To sum up, the extended process model provides a syntax structure that allows *graph-based* editing, rather than *tree-based* editing, inside the graphical model editor. The extended process model is parsed to the initial process model when the mobile application modeler saves a newly created extended process model. The parser removes the additionally introduced classes, i.e., the corresponding objects, thus creating a tree-based structure of tasks. However, this is not always possible if mobile application modelers create cycles or multiple edges. The parser provides trivial error-handling (e.g., ignoring multiple edges, multiple initial tasks) but does not yet provide a visual feedback or error location (cf. Tuovinen [Tuo00]). This **6.57** trade-off between freehand and syntax-directed editing is well discussed in the literature [KM00].

6.58 After extending the initial process model with these few elements, a more convenient model editor based on GMF can be generated. We will now introduce the graphical concrete syntax for the extended process model.

6.6.3.2 Graphical Concrete Syntax for the Extended Process Model

6.59 Tables 6.13 to 6.29 show the mapping of the abstract syntax (AS) (cf. Figure 6.7) to the graphical concrete syntax (GCS).

TABLE 6.13: Mapping of the abstract syntax element *ProcessContainer*

AS	The abstract syntax element *ProcessContainer* forms the diagram canvas and thereby directly or indirectly contains all other graphical concrete syntax elements.	
GCS (Type)	Canvas	
GCS (Example)	not applicable	
Context	none	
Edit operations	**Diagram operations**	The *Canvas* element provides the creation of *Process* elements.
	Property settings	The properties editor provides no further property settings.

TABLE 6.14: Mapping of the abstract syntax element *Process*

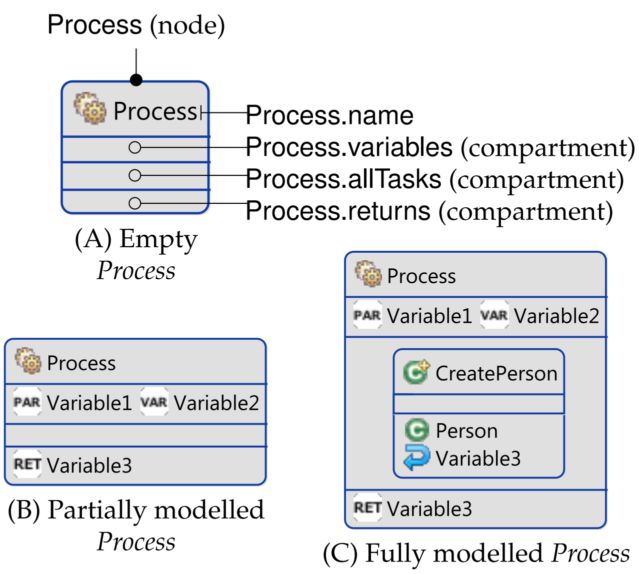
AS	The abstract syntax element *Process* and its contained elements like *Variables* and *Tasks* shape the graphical concrete syntax element *Process*.	
GCS (Type)	Node with compartments and label	
GCS (Example)	 (A) Empty *Process* (B) Partially modelled *Process* (C) Fully modelled *Process*	
Context	Arbitrarily often inside a *ProcessContainer* compartment	
Edit operations	**Diagram operations**	The *Process* element provides the creation of *Variables* and *Tasks*. *Variables* can be global parameter *variables*, local *variables*, and return *variables*.
	Property settings	The properties editor supports the setting of the feature *name*.

TABLE 6.15: Mapping of the abstract syntax element *Variable*

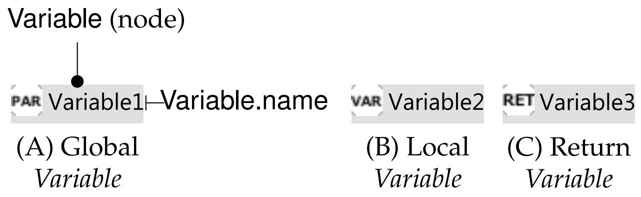
AS	The abstract syntax element *Variable* is mapped to the graphical concrete syntax element *Variable*.	
GCS (Type)	Node with label	
GCS (Example)		
Context	Arbitrarily often inside a *Process.variables* compartment or *Process.returns* compartment	
Edit operations	**Diagram operations**	The elements provide no further edit operations within the diagram.
	Property settings	The properties editor supports the setting of the features *name*, *changeable*, *defaultValue*, *scope*, *value*, and *eType* (inherited from *ETypedElement*).

TABLE 6.16: Mapping of the abstract syntax element *ProcessSelector*

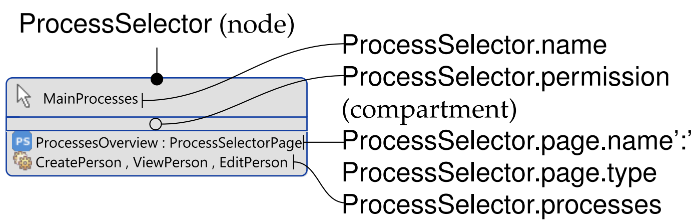
AS	The abstract syntax element *ProcessSelector* is mapped to the graphical concrete syntax element *ProcessSelector*.	
GCS (Type)	Node with compartment and label	
GCS (Example)		
Context	Arbitrarily often inside a *Process.allTasks* compartment	
Edit operations	**Diagram operations**	The element provides the creation of *Permissions*.
	Property settings	The properties editor supports the setting of the features *page*, *processes*, and *name*.

TABLE 6.17: Mapping of the abstract syntax element *Create*

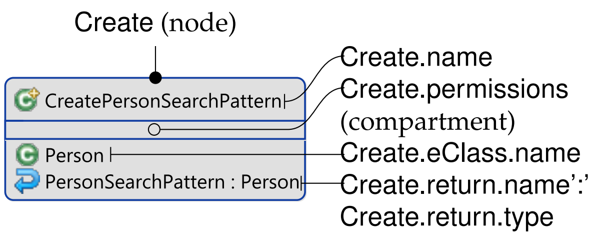
AS	The abstract syntax element *Create* is mapped to the graphical concrete syntax element *Create*.	
GCS (Type)	Node with compartment and label	
GCS (Example)		
Context	Arbitrarily often inside a *Process.allTasks* compartment	
Edit operations	**Diagram operations**	The element provides the creation of *Permissions*.
	Property settings	The properties editor supports the setting of the features *anchor*, *eClass*, *return*, and *name*.

TABLE 6.18: Mapping of the abstract syntax element *Read*

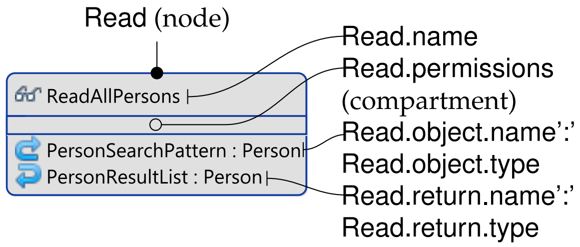
AS	The abstract syntax element *Read* is mapped to the graphical concrete syntax element *Read*.	
GCS (Type)	Node with compartment and label	
GCS (Example)		
Context	Arbitrarily often inside a *Process.allTasks* compartment	
Edit operations	**Diagram operations**	The element provides the creation of *Permission*s.
	Property settings	The properties editor supports the setting of the features *object*, *return*, and *name*.

TABLE 6.19: Mapping of the abstract syntax element *Delete*

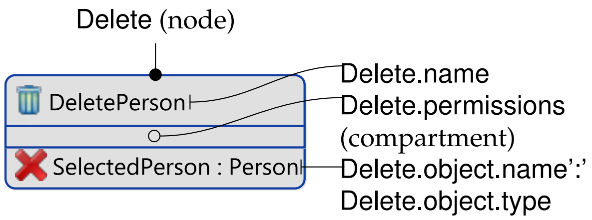
AS	The abstract syntax element *Delete* is mapped to the graphical concrete syntax element *Delete*.	
GCS (Type)	Node with compartment and label	
GCS (Example)		
Context	Arbitrarily often inside a *Process.allTasks* compartment	
Edit operations	**Diagram operations**	The element provides the creation of *Permission*s.
	Property settings	The properties editor supports the setting of the features *object* and *name*.

TABLE 6.20: Mapping of the abstract syntax element *Assign*

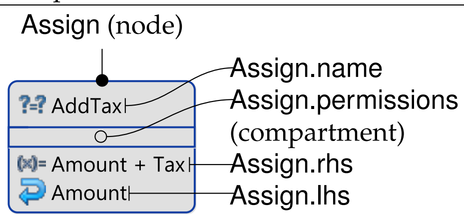
AS	The abstract syntax element *Assign* is mapped to the graphical concrete syntax element *Assign*.	
GCS (Type)	Node with compartment and label	
GCS (Example)		
Context	Arbitrarily often inside a *Process.allTasks* compartment	
Edit operations	**Diagram operations**	The element provides the creation of *Permission*s.
	Property settings	The properties editor supports the setting of the features *rhs*, *lhs*, and *name*.

TABLE 6.21: Mapping of the abstract syntax element *InvokeOperation*

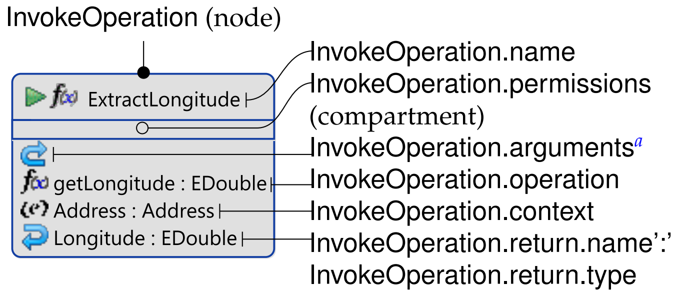
AS	The abstract syntax element *InvokeOperation* is mapped to the graphical concrete syntax element *InvokeOperation*.	
GCS (Type)	Node with compartment and label	
GCS (Example)	 [a] not used here	
Context	Arbitrarily often inside a *Process.allTasks* compartment	
Edit operations	**Diagram operations**	The element provides the creation of *Permissions*.
	Property settings	The properties editor supports the setting of the features *arguments*, *context*, *long*, *operation*, *return*, and *name*.

TABLE 6.22: Mapping of the abstract syntax element *InvokeGUI*

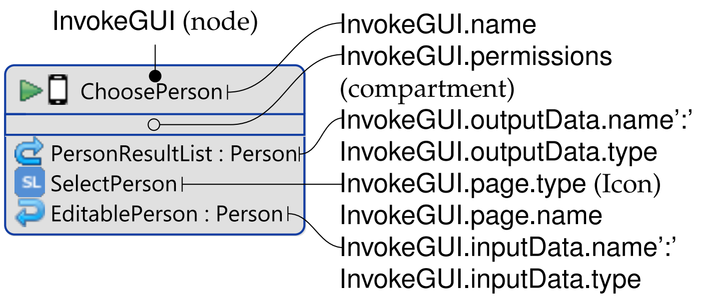
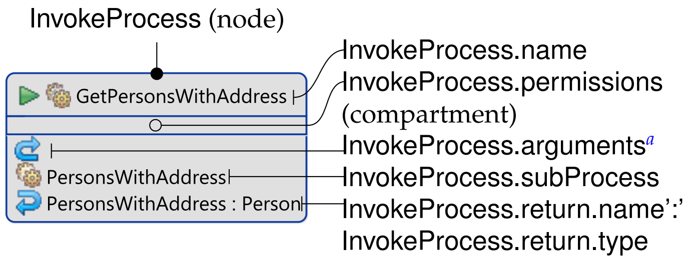
AS	The abstract syntax element *InvokeGUI* is mapped to the graphical concrete syntax element *InvokeGUI*. The second labeled icon in the lower compartment indicates which type of page is referred (e.g., *SL* for *SelectableList*).	
GCS (Type)	Node with compartment and label	
GCS (Example)		
Context	Arbitrarily often inside a *Process.allTasks* compartment	
Edit operations	**Diagram operations**	The element provides the creation of *Permissions*.
	Property settings	The properties editor supports the setting of the features *input*, *outputData*, *inputData*, *return*, and *name*.

TABLE 6.23: Mapping of the abstract syntax element *InvokeProcess*

<table>
<tr><td>AS</td><td colspan="2">The abstract syntax element InvokeProcess is mapped to the graphical concrete syntax element InvokeProcess.</td></tr>
<tr><td>GCS (Type)</td><td colspan="2">Node with compartment and label</td></tr>
<tr><td>GCS (Example)</td><td colspan="2">

[a] not used here</td></tr>
<tr><td>Context</td><td colspan="2">Arbitrarily often inside a Process.allTasks compartment</td></tr>
<tr><td rowspan="2">Edit operations</td><td>Diagram operations</td><td>The element provides the creation of Permissions.</td></tr>
<tr><td>Property settings</td><td>The properties editor supports the setting of the features arguments, return, subProcess, synchronized, and name.</td></tr>
</table>

TABLE 6.24: Mapping of the abstract syntax element *CRUDGui*

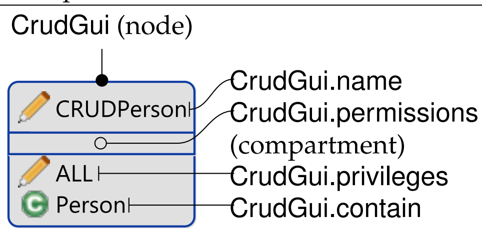
<table>
<tr><td>AS</td><td colspan="2">The abstract syntax element CRUDGui is mapped to the graphical concrete syntax element CRUDGui.</td></tr>
<tr><td>GCS (Type)</td><td colspan="2">Node with compartment and label</td></tr>
<tr><td>GCS (Example)</td><td colspan="2">
</td></tr>
<tr><td>Context</td><td colspan="2">Arbitrarily often inside a Process.allTasks compartment</td></tr>
<tr><td rowspan="2">Edit operations</td><td>Diagram operations</td><td>The element provides the creation of Permissions.</td></tr>
<tr><td>Property settings</td><td>The properties editor supports the setting of the features anchor, contain, privileges, and name.</td></tr>
</table>

TABLE 6.25: Mapping of the abstract syntax element *Permission*

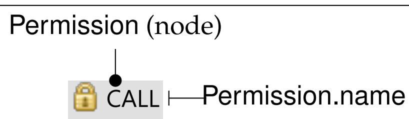
<table>
<tr><td>AS</td><td colspan="2">The abstract syntax element Permission is mapped to the graphical concrete syntax element Permission.</td></tr>
<tr><td>GCS (Type)</td><td colspan="2">Node with label</td></tr>
<tr><td>GCS (Example)</td><td colspan="2">
</td></tr>
<tr><td>Context</td><td colspan="2">Arbitrarily often inside a Task.permission compartment; except the task types If1, IfElse1, and While1</td></tr>
<tr><td rowspan="2">Edit operations</td><td>Diagram operations</td><td>The element provides no further edit operations within the diagram.</td></tr>
<tr><td>Property settings</td><td>The properties editor supports the setting of the features name and permission.</td></tr>
</table>

TABLE 6.26: Mapping of the abstract syntax elements *RefTrue* and *RefFalse*

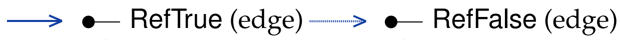
AS	The abstract syntax element *RefTrue* and *RefFalse* are mapped to equally named elements of the graphical concrete syntax.	
GCS (Type)	Edge	
GCS (Example)	 (A) *RefTrue* element (B) *RefFalse* element	
Context	Arbitrarily often inside a *Process.allTasks* compartment; The element *RefTrue* may start from any kind of task, while the *RefFalse* element only starts from *ConditionalNodes*.	
Edit operations	**Diagram operations**	The elements provide the creation of Tasks, which are selected as *suc* value.
	Property settings	The properties editor supports the setting of the features *pre*, *suc*, and *name*.

TABLE 6.27: Mapping of the abstract syntax elements *If1*

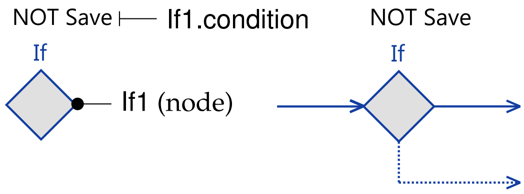
AS	The abstract syntax element *If1* is mapped to the graphical concrete syntax element *If1*.	
GCS (Type)	Node with label	
GCS (Example)	 (A) *If1* element (not connected) (B) *If1* element (fully connected)	
Context	Arbitrarily often inside a *Process.allTasks* compartment	
Edit operations	**Diagram operations**	The element provides the connection of *RefTrue* and *RefFalse* elements (edges).
	Property settings	The properties editor supports the setting of the features *condition* and *name*.

TABLE 6.28: Mapping of the abstract syntax elements *IfElse1*

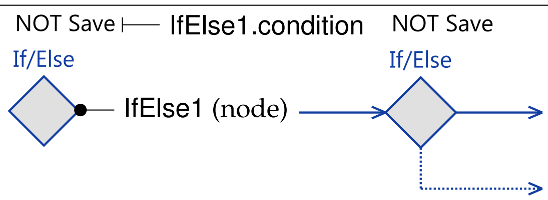
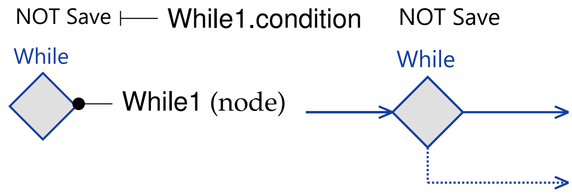
AS	The abstract syntax element *IfElse1* is mapped to the graphical concrete syntax element *IfElse1*.	
GCS (Type)	Node with label	
GCS (Example)	 (A) *IfElse1* element (not connected) (B) *IfElse1* element (fully connected)	
Context	Arbitrarily often inside a *Process.allTasks* compartment	
Edit operations	**Diagram operations**	The element provides the connection of *RefTrue* and *RefFalse* elements (edges).
	Property settings	The properties editor supports the setting of the features *condition* and *name*.

TABLE 6.29: Mapping of the abstract syntax elements *While1*

AS	The abstract syntax element *While1* is mapped to the graphical concrete syntax element *While1*.	
GCS (Type)	Node with label	
GCS (Example)	 (A) *While1* element (not connected) (B) *While1* element (fully connected)	
Context	Arbitrarily often inside a *Process.allTasks* compartment	
Edit operations	**Diagram operations**	The element provides the connection of *RefTrue* and *RefFalse* elements (edges).
	Property settings	The properties editor supports the setting of the features *condition* and *name*.

6.60 The abstract syntax element *Expression* is not mapped to a graphical syntax element: it is mapped to textual syntax as presented in the next paragraph.

6.7 Combining Textual and Graphical Editing

6.61 The tooling of our approach has points of similarity with *AToM*^{3} (de Lara et al. [DL+04]), which combines different editing paradigms for visual model editors. We follow a multi-paradigm approach, thereby extending the functionality of GMF (cf. Völter [Völ09, Sec. 5]). While *AToM*^{3} combines a meta-modeling approach with a graph-grammar approach, we combine the meta-modeling approach with textual parsing techniques developed for string languages. We use the ANTLR (Another Tool for Language Recognition) [Par12] framework to generate a parser to recognize expression statements of the domain-specific modeling language. These expressions are part of the presented *conditional* elements such as *If*, *If/Else*, and *While*. The *Assign* element also refers to an expression (cf. Figure 6.7).

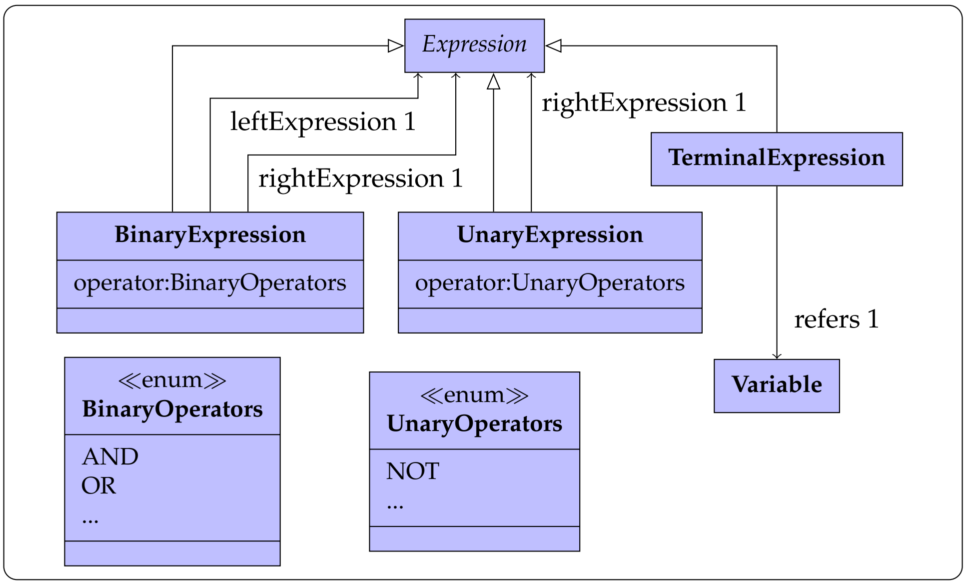


FIGURE 6.15: Ecore metamodel for defining the behavior of mobile applications (Pt. II/II)

Figure 6.15 shows how these expressions are composed. Based on the requirement 6.62
to create logical expressions in a textual way, the extended process metamodel (cf. Figure 6.14) accepts at first string values inside the attributes *condition* and *rhs* of the relevant elements.

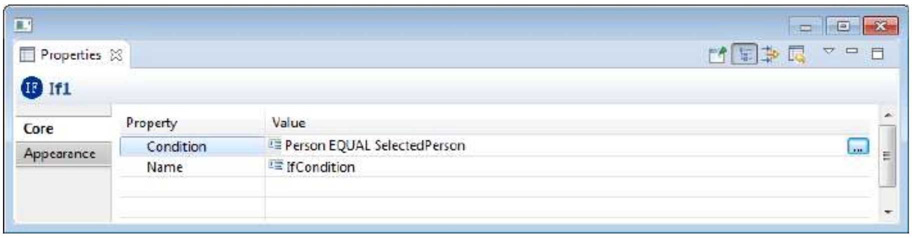


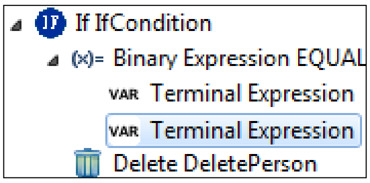
FIGURE 6.16: A textual entered condition of an *IF* element

Figure 6.16 shows a textual expression (`Person EQUAL SelectedPerson`) cap- 6.63
tured by the *condition* attribute of an *If1* element. When saving the model, the generated parser becomes active and recognizes the structure based on the grammar rules given in Listing 6.4.

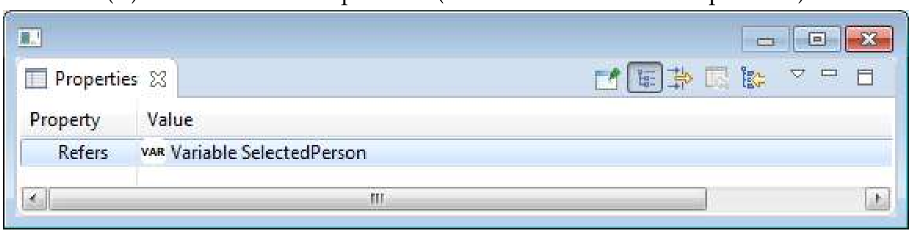
LISTING 6.4: Parser rules for an expression

```
grammar Expression;

start: expression ;

expression : '(' expression ')'  | op=unaryOperator expression
      | left=expression op=binaryOperator right=expression |
    variable ;

unaryOperator  : (NOT | ... )  ;

binaryOperator : (AND | OR | EQUAL | ... ) ;

variable: IDENTIFIER;
```

Finally, the parser delivers the abstract representation of the textual expression 6.64
entered before as shown in Figure 6.17.



(A) The conditional expression (with selected terminal expression)



(B) Properties of the selected terminal expression

FIGURE 6.17: The parsed conditional expression

During the numerous experiments with undergraduate students, we came to learn 6.65

that the textual input option is generally valuable, but it remains unclear for the mobile application modelers which variables are in scope of the textual expression to be written. The scope of variables for the conditional (or assignment) statements is difficult to figure out. Henceforth, an expression editor supports mobile application modelers while creating textual expressions. The expression editor shown in Figure 6.18 proposes only variables that are in scope of the currently edited element.

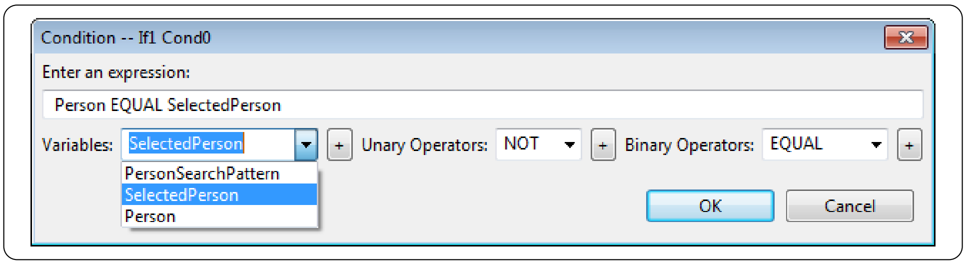


FIGURE 6.18: Expression editor for conditions and assignments (rhs)

6.8 Graphical Model Editor Implementation

6.66 Table 6.30 shows the second set of plugins forming the graphical model editor. The graphical model editor for app models is designed as a graphical model editor comprising three different views. There is a view for data modeling, one for process modeling (see Figure 6.19), and one for graphical user interface modeling. Since this multi-editor functionality is not provided by GMF, the first plugin project (first row of Table 6.30) realizes this multi-editor functionality and provides three views for the different models. As expected, changes in one view are immediately propagated to the other ones. The existing Ecore diagram editor has been integrated for data modeling (second row of Table 6.30).

6.67 The GUI model editor and the extended process model editor are created by the presented workflow (cf. Section 6.5.1) resulting in one plugin for each model (third and fourth rows of Table 6.30). However, the generated graphical model editors do not cover all required features and must be adapted. For example, we want to use custom labels and icons. To this end, we apply the well-known generation gap pattern [Vli98, pp. 85–101], which provides a mechanism to inject custom functionality. Thus, the graphical model editors for the extended process model and the GUI model are extended (fifth and sixth rows of Table 6.30). Figure 6.19 shows the resulting graphical model editor for an app model while editing a process model.

TABLE 6.30: Second set of plugins shaping the MDD infrastructure

Plugin project name	**Content**	**Type**
pimar.editor.visual.multieditor.plugin	Multi-editor framework code	manual
org.eclipse.gmf.ecore.editor	Data model editor	reused
pimar.editor.visual.gcore.diagram	GUI model editor	generated
pimar.editor.visual.wcoreExtended.diagram	Extended process model editor	generated
pimar.editor.visual.gcore.diagram.extended	GUI model editor (custom part)	manual
pimar.editor.visual.wcoreExtended.diagram.extended	Extended process model editor (custom part)	manual

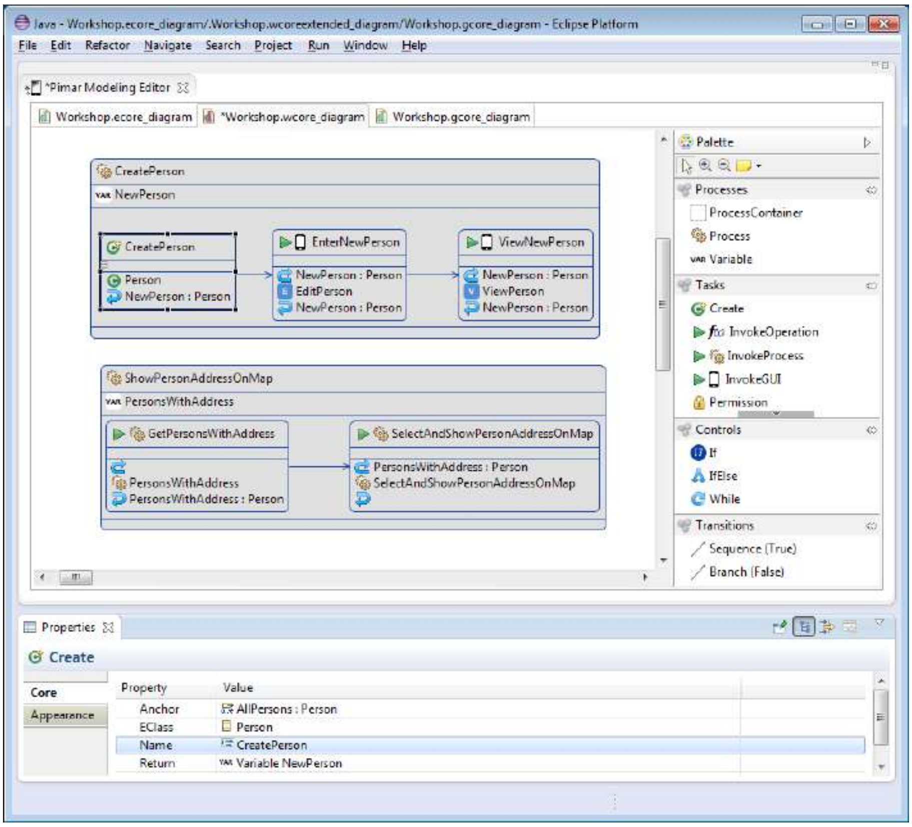


FIGURE 6.19: Graphical model editor for app models (Process model being edited)

6.9 Further EMF-Based Tools and Frameworks

Since EMF is a widely-used modeling framework implemented on top of the Eclipse **6.68**
software development platform, we could use other EMF-related tools to analyze and modify the created app models. Two of the most important applications are model quality assurance and model transformation.

6.9.1 Tooling for Model Quality Assurance

Arendt and Taentzer provide [AT13] a tool environment for quality assurance **6.69**
based on the Eclipse modeling framework. This tool environment supports metrics computation, smell detection, and refactoring for models that are based on EMF. Different specification methods such as Java, OCL [35], and Henshin [Are+10] can be used. The model quality approach can be adapted to project-specific and domain-specific needs. This feature facilitates the creation of metrics functions, smell definitions, and refactoring operations, which are tailored to the designed domain-specific modeling language.

We apply the quality assurance process to our domain-specific modeling language. **6.70**
We define 41 metrics functions and 58 smells. Refactorings are currently not implemented.

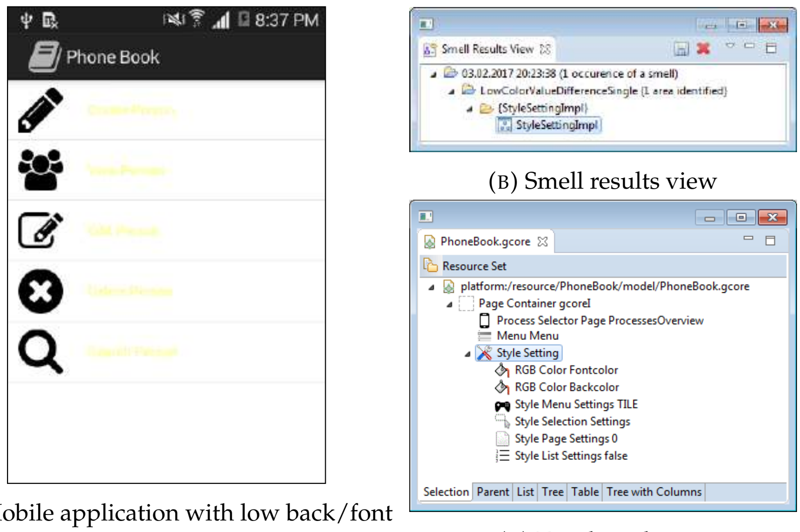
Example (Color metric and color smell). While creating a mobile application, the **6.71**
mobile application modelers have to set at least one style setting inside the GUI model. In turn, this style setting requires at least two colors that serve as background

and font colors of the mobile application. They may choose inappropriate color values, resulting in bad legibility of the mobile application as shown in Figure 6.20a.

6.72 Hence, one of the realized metric functions, `CVD_min` (color value distance minimum), calculates at first the minimal difference between two color objects. Listing 6.5 shows the implementation of this function.

LISTING 6.5: Metric function `CVD_min`

```
public final class CVD_min implements IMetricCalculator {
...
   @Override
   public double calculate() {
   ret = minDifference(color1, color2);
   ...
   return ret;
   }

   private double minDifference(RGBColor color1, RGBColor
       color2) {
      double ret = Math.abs(color1.getRed()-color2.getRed());
      double diff = Math.abs(color1.getGreen()-color2.getGreen
          ());
      if(diff<ret){ret=diff;}
      ...
      return ret;
   }
}
```



(A) Mobile application with low back/font color difference

(B) Smell results view

(C) Tree-based view

FIGURE 6.20: Detection of model smells

6.73 Based on this metric function we define the smell *LowColorValueDifferenceSingle*,. This smell occurs if a color value distance (i.e., red, green, or blue) of two colors (e.g., font or background color) is below a certain defined threshold (e.g., 100). Consequently, quality assurance can detect smells for models having inappropriate color values (e.g., *Fontcolor (153, 255, 255)* and *Backcolor (255, 255, 255)*) as shown in Figure 6.20b. Additionally, the quality assurance tool points to the occurrence of the smell (cf. Figure 6.20c). □

6.74 Table 6.31 shows the third set of plugins forming the quality assurance of the model editor.

TABLE 6.31: Third set of plugins shaping the MDD infrastructure

Plugin project name	Content	Type
pimar.model.analysis.metrics	Metrics definitions	manual
pimar.model.analysis.smells	Smell definitions	manual

6.9.2 Tooling for Model Transformation

The Henshin framework [Are+10] provides an in-place transformation of EMF models. Pattern-based transformation rules can be expressed using a visual syntax. Henshin is suitable for implementing model refactorings and metamodel evolution. In the course of this work, we use Henshin to refactor app models at design time. For example, an in-place transformation of an app model can be used to meet the requirements of a particular platform or device. **6.75**

6.10 Evaluation

The preceding sections presented the main features of our domain-specific modeling language for mobile applications and the corresponding model editors. We discuss two questions: first, did our designed artifacts cover the modeling language requirements provided in Chapter 4 (Question 1)?, and second, is the design of the domain-specific modeling language appropriate (Question 2)? **6.76**

6.10.1 Setup

To show that the artifacts described in this chapter cover the requirements, we map the artifacts to the corresponding requirements. Moreover, since the requirements already contain acceptance criteria, we check whether or not the artifacts fulfills the particular acceptance criteria for each of the targeted requirements. **6.77**

The requirements, especially the modeling language requirements, are not expressed in a final manner through the use of an iterative development process. Consequently the acceptance criteria are also expressed on a very high level. Hence, we will discuss the final artifacts, i.e., the domain-specific modeling language for mobile applications and the corresponding model editors, with respect to generally accepted guidelines for the design of domain-specific modeling languages. To this, we use design guidelines for domain-specific modeling languages stated in [Kar+14] (cf. [Mer12, Chap. 14]) and the design guidelines for user interface description languages stated in [SV03] [Nav+09]. These generally accepted guidelines serve as additional acceptance criteria. The evaluation with respect to the feature model is postponed to the next chapter in order to cover the generator components of the model-driven development infrastructure. Similar, the user studies and user experiences are presented therein. **6.78**

6.10.2 Requirement Coverage (Question 1)

The parts of the overall implementation of the model-driven development infrastructure which are provided by the plugins denoted in the tables 6.1, 6.30, and 6.31 cover the requirements 4.1.1 to 4.1.5 and 4.3.1 (cf. Table 4.1). However, the semantics of the domain-specific modeling language is still not defined. Only the code generators define the behavior of the model elements by transforming them to program code with well-defined syntax and semantics. **6.79**

The plugin *pimar.language* contains the metamodel which defines the domain-specific modeling language. By reusing the Ecore metamodel (cf. Section 6.3.1) for **6.80** data modeling purposes, the data modeling part allows an appropriate modeling of various domains where the mobile application should be used. The behavior modeling part (cf. Section 6.3.3) provides abstract modeling (e.g., by the *CrudGui* element) as well as more specific model elements for individual process definitions. The discussion of the runtime configurability based on the process instance model will be postponed to the next chapter. The third part of the plugin implements the language elements for graphical user interface modeling (cf. Section 6.3.2). In accordance with the acceptance criteria, this part provides abstract modeling elements. Again ,the runtime configurability based on the style model will be discussed in the next chapter.

6.81 Finally, well-formedness rules for app models and the model quality assurance plugins (*pimar.model.analysis.**) were implemented to cover the modeling requirements which are not directly focused on the design of domain-specific modeling languages, but are highly relevant in order to avoid compile errors or inappropriate program code during the code generation process. In accordance with the acceptance criteria of the mentioned requirements, the well-formedness rules prohibit many cases in which the models may lead to compile errors. Moreover, the model quality assurance process enables infrastructure developers to define metrics and smells that should be identified and avoided during the modeling process.

6.82 The second set of plugins (cf. Table 6.30) covers the tooling requirement of a graphical model editor. In accordance with its acceptance criteria, the graphical model editor provides freehand editing and the evaluation of the created app models.

6.10.3 Language Adequacy (Question 2)

6.83 In order to demonstrate that the designed domain-specific modeling language has an appropriate design, we discuss it with respect to the following design guidelines:

6.10.3.1 Design Guidelines for Domain-Specific Modeling Languages

6.84 The main purpose of our domain-specific modeling language is code generation. It will be used mainly by mobile application developers, possibly also by domain experts and content-providing users. The language is designed to be platform-independent, i.e., independent of Android, iOS, or other mobile platforms.

6.85 A decision whether to use a textual or graphical concrete syntax does not have to be made since we design the language with EMF and therefore have the possibility to add a textual concrete syntax with e.g., Xtext [Bet13] or a graphical one with e.g., the Graphical Modeling Framework (GMF) [Gro09] [Rub+11], as shown before. The development of a textual syntax is less work and will be added in the near future. We decided to reuse EMF for data modeling, as it is very mature. Since we define our language with EMF, the Ecore metamodel can also be reused, along with its type system.

6.86 Next, we discuss the choice of language elements. Since all generated mobile applications share the same architectural design (as detailed in the next chapter), the domain-specific modeling language does not need to reflect the architecture. However, data structures, behavior, and graphical user interface design are covered. Since we want to raise the abstraction level of the domain-specific modeling language as high as possible, we have discussed each specific feature of mobile applications carefully to decide whether it can be set automatically by the code generator or the mobile application 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 mobile application modeler since these are based on the operations modeled. The authors of [Kar+14] emphasize the simplicity of a language in order to be useful. Our domain-specific modeling 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 graphical concrete syntax has to be chosen carefully: 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 shapes and icons to increase their descriptiveness and make them easily distinguishable. Models are organized in three separate sub-models with respect to 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. **6.87**

There is one part in particular 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 operators such as while loops and if clauses. This allows for an easier handling of operations for code generation. However, they are unhandy during the modeling process. The chosen layout does not have any effect 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 process models. **6.88**

6.10.3.2 Design Guidelines for User Interface Description Languages

Looking at the design and comparison criteria of user interface description languages given in [SV03] [Nav+09], (i.e., component model, methodology, tools, supported languages, platforms, targets, interaction coverage, expressiveness), we can say that our graphical user interface definition reflects most of the stated criteria. **6.89**

In order to position our GUI model with respect to the existing work, we discuss the following criteria: first, the component model criteria require a separation of the user interface description language into sub-models (or aspects) such as a task model, domain model, presentation model, and dialog model. Our app model is structured in this way per design. The task model describes different tasks to be accomplished by the user. Within our approach, we describe these tasks (e.g., viewing data, filling out a form, taking a picture) by corresponding page types (such as *ViewPageEditPage*, and *TakePicturePage*). Each page has a predefined, generic structure of graphical user interface components and follows a specific purpose. For example, the purpose of the *ARPage* is to compare the current camera image with a predefined pattern and to augment it with additional information such as text and images. The *ARPage* hides the technical details of the AR functionality from the mobile application modeler and reuses already existing AR frameworks (e.g., MetaioSDK [26]). **6.90**

The methodology criteria distinguish between (i) the specification of user interfaces for each of the different contexts of use and (ii) the specification of a generic (or abstract) user interface description for all the different contexts of use. Our approach provides both variants. A graphical user interface (here *Page*) can be reused in different contexts by using style models that act as runtime instances of GUI models. Alternatively, graphical user interfaces (i.e., *Pages*) can be created for each context of use to, e.g., support an individual layout or modify generated code. **6.91**

The tool criteria describe the existence of a translation tool. Such a tool translates a user interface description into a specific language or platform [Pue+94]. Our graphical user interface description is automatically translated by code generators. In this respect, the criteria of platforms and supported languages are also met by **6.92** our code generators supporting different platforms (Android, iOS) and different languages (XML, Java, Objective-C).

6.93 The target criteria describe the ability of the user interface description to express variations according to the desired platform, user group, and environment. Our modeling approach supports user-specific targeting of processes and related user interfaces specified by provider models [Vau+14]. Thus, our GUI model has multi-user capability.

6.94 Finally, we discuss the expressiveness of our GUI model: according to Navarre et al. [Nav+09], the expressiveness includes data description, state representation, event representation, representation of time, concurrent behavior, and dynamic instantiation. As many other user interface description languages, our GUI meta-model covers this variety of expressiveness aspects only partly. As mentioned earlier, data description is part of our modeling concept, namely the data model. The state representation of our modeled graphical user interface (pages) cannot be modeled, but it is provided at the code level. Events can be modeled. A usual case is the invocation of and return from graphical user interfaces with certain return events (e.g., Save and Cancel). Most of Navarre's expressivness criteria are either provided by the data or process sub-model or at the code level. Some of them have not been in the focus of our design and are thus not provided. Our GUI model is admittedly minimalistic, describing a graphical user interface that is focused on a specific purpose.

6.10.4 Threats to Validity

6.95 Considering external validity, it is not ensured that our designed domain-specific modeling language is, in general, appropriate regarding different potential user groups, although it respects several design guidelines. For example, mobile application modelers who are more technically oriented might prefer a textual concrete syntax and a low abstraction level because this is more similar to traditional code-writing. In turn, mobile application modelers with a background in business-oriented domain-specific modeling languages might prefer a high-level abstraction and a graphical concrete syntax. An empirical study might confirm this hypothesis. In addition to the adaptation of the domain-specific modeling language described in Section 6.7, we will report on the experiences during the practical use of the designed domain-specific modeling language in the next chapter.

6.96 A threat to internal validity is that the iteratively designed domain-specific modeling language was not discussed after each of the iterations. Earlier design variants are ignored since only the final design was evaluated with respect to the design guidelines. Moreover, the well-formedness rules for app models can only be completed as soon as the code generators are available, because inappropriate app models which lead to compile errors can be identified easier by systematic code generator tests.

Chapter 7

Reference Applications, Code Generators, and Prototypes

This chapter presents the development of code generators as a second essential component of the model-driven development infrastructure. However, before starting with the construction of the code generators, representative reference applications must be created according to our agile bottom-up development approach (cf. Figure 3.1). **7.1**

Thus, we start with a qualitative analysis of reference applications, a step that we call *reverse engineering*. These reference applications were initially provided by our industrial research project partners. Our feature model can express different kinds of mobile applications. Therefore, the analysis of only one reference application may not be sufficient. Based on the focused features (cf. Section 5.4), we consider *information systems* and *transaction systems* as well as *standalone* mobile applications in this work. Therefore, we start the qualitative analysis with two different mobile applications that represent the respective kinds of mobile applications. **7.2**

The qualitative analysis of reference applications is useful for identifying domain concepts, but the codebase of the reference applications is usually not directly suitable for code generator template extraction: the reference applications may contain anti-patterns, code smells, or bugs, which are not to be included in the code generators. Additionally, recurring and schematic boilerplate code may be obfuscated by slightly different manifestations and cannot be identified easily. In both cases, a refactoring toward design patterns and code quality may be required, but a complete refactoring of real-world mobile applications often seems ineffective and is too time-consuming. **7.3**

A less complex approach is the re-engineering of these reference applications, as *forward engineering*. Forward-engineered applications may be limited in certain aspects, meaning that forward-engineered applications contain only application-specific functionality, i.e., will not implement recurring or schematically similar structures as they may occur in the initially analyzed reference applications. Additionally, forward engineering allows unifying several mobile applications on a conceptual level. Finally, forward-engineered applications contain the application concepts in a condensed manner. To guarantee that the re-engineered applications are *representative* with respect to the original reference applications, several techniques can be applied (e.g., back-end compatibility, functional tests, and mobile application developer's interviews). **7.4**

Another aspect of forward engineering is the harmonization of different implementation variants. As presumed, every analyzed implementation (iOS, Android) was developed and maintained by different teams. Sociological and technical factors influence the design of the platform-specific implementations, which is why a second goal of the forward engineering is to align the platform-specific aspects of each implementation as far as possible. **7.5**

Finally, forward engineering can be used to introduce new features that will be **7.6** provided by the generated mobile application later. In our case, we add augmented reality (AR) technology to forward-engineered reference applications although the analyzed applications do not have this feature.

7.7 After an evaluation of the forward-engineered applications, the second part of this chapter will describe how the code generators are designed. The term code generator in its broadest sense describes not only native program code generation but rather the generation of all artifacts of a mobile application. Hence, we subsume the IDE preparation for a mobile platform-specific development project and the architecture of the generated mobile applications as relevant while constructing a code generator. These aspects are often ignored in the model-driven development literature. The native program code generation in the narrow sense, is explained along the processing of the app model. Subsequently, the processing of the provider model is explained and some hints how generated native program code could be customized are given.

7.8 The concluding evaluation presents a discussion with respect to the requirements and the feature model criteria. We demonstrate, using three case examples, how generated prototypes cover the focused features. The applicability of the model-driven development infrastructure is demonstrated through a user-study evaluation. Finally, we discuss the structure and similarity of the code generators and the generated mobile applications and evaluate additional more technical aspects.

7.1 Reverse Engineering of Reference Applications

7.9 Reverse engineering is mainly a qualitative analysis of reference applications and gives a deeper insight into the kinds of mobile applications we want to generate. Their architecture is of particular interest. Hence, instead of focusing on the detailed features of the considered mobile applications, we focus on their concepts. The selection of reference applications excludes applications which have little to no application data (e.g., games and music players). Such *non-interoperable* and *non-data-oriented* mobile applications are not our focus. The process step involving reverse engineering does not contain any kind of implementation task. This will be carried out by the forward engineering task in the following stage. Finally, at the end of this section, we will provide a short discussion of how the selected mobile applications reflect the focused features.

7.1.1 Information System

7.10 The analysis of reference applications was started with the *key2guide* application (cf. Section D.2), which can be classified as an offline-capable *information system*.

7.11 *Information systems* generally have a unidirectional data flow between the back end and the front end. Typical mobile applications that realize an information system are passenger/visitor information systems, weather information systems, and financial information systems. These systems have one thing in common: the provided data is transient and is distributed over a hub (e.g., a back-end system). In contrast, an electronic dictionary or an encyclopedia generally uses stable records, which is why their data distribution may also be different (e.g., provided by a factory setting).

7.12 Two systems can be further distinguished based on the degree of transience of data:

7.13 First, as observed by *key2guide*, the number of data records will be often extended while the existing records remain unchanged. In such cases, a native implementation of the mobile application is recommended. Mobile applications of this kind can replicate the data locally, which means they can operate in a disconnected situation unless some data is added at the back end.

Second, other mobile applications like weather information systems are very often realized by a web-based mobile application or the mobile version of a website. The data of such an application is naturally transient, and thus, a replication makes no sense in this case. The response of the mobile end-user request is cached only temporally for a short time. These kinds of information systems do not support being restarted, longer states of inactivity, or network disconnection without the need to get current data again. **7.14**

Another aspect is the specificity of a request/response from a back-end system and the amount of replication. For example, *key2guide* provides only two replication modes (full replication/full replication without media files). Full replication is used because a mobile end user might use the information at any place or any time later. The time and place of such a replication request cannot be used beneficially to reduce the amount of replicated data. In turn, an online request within a web-based mobile application reveals the current position and time of information usage (immediately). For example, a mobile end user requesting a flight connection at a certain airport at a certain time might generate a very specific response e.g., only the next connections. Outdated data or data extending too far into the future can be ignored and thus excluded from a replication of data. **7.15**

As analyzed by *key2guide*, the only issue with replication in information systems is the obsolescence of data due to additions or modifications at the back end, which is called *read inconsistency*. However, in practice, mobile end users accept *read inconsistency* on replicated data while they are disconnected because even potentially inconsistent data might be more beneficial than no data access at all. **7.16**

To sum up, many existing mobile applications that realize an *information system* are still web-based and lack offline capability. One of the reasons for this is the fact that offline-capable implementations such as *key2guide* require native implementation for each of the desired platforms. **7.17**

7.1.2 Transaction System

The second analyzed reference application was *key2operate*, which can be classified as an offline-capable *transaction system*. **7.18**

In most cases, *transaction systems* are realized as online systems that follow the well-known online transaction processing (OLTP) paradigm. This paradigm describes that transactions are performed in real time and online on a centralized system, i.e., database management system (DBMS). Conflicts between different transactions (or their users) follow well-known conflict definitions (e.g., taken from serializability theory), and can be detected and resolved easily in such a system. However, when it comes to the domain of mobile applications, the mentioned paradigm needs to be rethought. As stated by Bernstein [Ber+87, Chap. 8.8] transaction processing on replicated data does not tolerate communication failures. As a result, mobile applications always have to work online to meet the assumptions of the mentioned paradigm. This does not seem to be feasible in many mobile applications and usage scenarios. **7.19**

Key2operate realizes an offline-capable *transaction system* by employing replication and synchronization. This *transaction system* has a bidirectional data flow that bridges back end and front end mutually. Transactions can be performed online as well as offline within certain limits. **7.20**

One ongoing problem, which is also not resolved in *key2operate*, is the conflict analysis in online- and offline-capable transaction systems. At first, online- and offline-capable transaction systems have the same *read inconsistency* problem as **7.21** information systems while operating offline on replicated data. Additionally, transactions are usually not limited to reading data, but they also write data. Thus, *write inconsistency* might occur. For example, while synchronizing the mobile front end (mobile client) with the back end (server), a locally performed transaction must be reset or reevaluated because another mobile client has changed the value in the meantime.

7.22 The analyzed mobile reference application requires no *write consistency*, a fact that may lead to conflicts at the synchronization, because two offline mobile users may write to the same (replicated) value. Users of *key2operate* generally have to accept conflicts while operating offline. This is not uncommon in other mobile applications as well.

7.23 To provide conflict-free transaction processing inside mobile applications while being offline, the implementation of more sophisticated concepts from literature (e.g., mobile transaction models) is required.

7.24 To sum up, mobile applications that realize *transaction systems* are at present largely incapable of working both online and offline and ensuring, at the same time, conflict-free operation.

7.1.3 Standalone System

7.25 The analysis of the reference applications has shown which kind of mobile applications should be covered by the model-driven development infrastructure and what the challenges are. The analysis of the analyzed mobile applications' architectures shows that both can temporarily operate offline. Thus, the question arises whether these designs also cover the design of a standalone application that works only offline. As part of the analysis, we notice that the analyzed reference applications are potentially able to work only offline. The reason for this is that the back end provides no core functionality (e.g., application logic) which is required by the mobile device permanently.

7.26 Hence, the model-driven development infrastructure should also generate standalone applications. Besides, the generated default back end should provide as little functionality as possible, because we focus not on model-driven development infrastructure for web-applications.

7.1.4 Coverage of Focused Features

7.27 Based on the focused features given in Section 5.4, we will discuss which of these focused features are covered by the selected mobile applications and which of these features will additionally be covered by the model-driven development infrastructure and the mobile applications it can generate. As shown in Table 7.1, the selected mobile applications cover most of the features, thus we consider them as *representative* reference applications. However, the remaining features deal mostly with software engineering concerns (e.g., model-driven development of mobile applications). Hence, our goal in this part of the thesis is to create mobile applications in a model-driven way which provide at least the same functionality as the respective selected reference applications.

7.28 key2guide (cf. Section D.2) is a mobile multimedia guide (*Information system*) available for the *native* platforms *Android* and *iOS*. It uses several sensors/actors and interfaces of the devices' hardware, as well as internal memory. It can interpret data that are obtained from a back-end system, which is why it has a *hybrid* architecture. Finally, it is designed for permanent offline operation (i.e., *local data and transaction management*), except for its initial startup it needs a network connection.

7.29 key2operate (cf. Section D.3) is a manufacturing and production information system

TABLE 7.1: Focused features and mobile application features

Feature group ▷ ... ▷ Feature/s	**Covered by application**
Application type ▷ Standalone	not covered
Application type ▷ Information system	key2guide
Application type ▷ Transaction system	key2operate
Software platform ▷ Android	key2guide
Software platform ▷ iOS	key2guide
Hardware platform ▷ Screen ▷ Resolution	not covered
Hardware platform ▷ (Actors/Sensors/Interfaces/Memory)	key2guide, key2operate
Application architecture ▷ Native application	key2guide, key2operate
Application architecture ▷ Interpreter application	key2operate
Application architecture ▷ Hybrid application	key2guide
Application development ▷ Model-driven	not covered
Model-driven ▷ Design model (Abstract/Detailed)	not covered
Model-driven ▷ Runtime model	key2operate
Data and transaction management ▷ Local	key2guide, key2operate
Data and transaction management ▷ Central	key2operate
Graphical User Interface ▷ Dynamic	key2operate
Context-awareness ▷ Platform context-awareness	not covered

(MPIS). Information can be retrieved in a manner similar to key2guide from a backend system, but *transactions* can also be made (e.g., supplies requests). key2operate is a native mobile application that uses industry-typical sensors, i.e., barcode scanner. Moreover, local memory is required to store data during offline situations. However, key2operate synchronizes transactions made offline using a centralized backend system. Hence, the data and transaction management are both local and centralized. key2operate is designed as an interpreter, because manufacturing and production processes are company-specific. It applies a runtime model that specifies the behavior of the mobile application and the graphical user interface.

To sum up, the two main contributions of this thesis (cf. Sections 1.5.1 and 1.5.2) cover all focused features. The mobile applications generated by the model-driven development preserve the existing features of the prototypes but will additionally cover the features which are not yet covered. At the end of this chapter, we will show which features are covered by the model-driven development infrastructure (cf. Table 7.6). The remaining, uncovered features are discussed in the second part of this thesis. **7.30**

7.2 Forward Engineering of Reference Applications

Forward engineering follows the goal to create a *representative* reference application that, firstly, combines the identified architectural features of the analyzed mobile applications, and secondly, provides native program code of good quality which can be adopted by the code generators. **7.31**

7.2.1 Front End

At first, forward engineering of the front end delivers a mobile application for each of the desired software platforms (Android and iOS). The process of forward engineering was mostly identical for the Android and iOS reference applications, but there were some differences, because some tools were only available for Eclipse **7.32** rather than XCode. Forward engineering focuses on the product *key2guide*, but aspects of *key2operate* have also been realized.

7.33 At the beginning of the forward engineering, the mobile applications use the original back-end system. Thus, the first implementation step was to provide mobile applications that can login to the back end and download the project files and, if applicable, the related media files.

7.34 Since *key2operate* provides a dynamic data model in the opposite direction of the static data model of *key2guide*, forward engineering adopts dynamic modeling. In order to provide a flexible data model, the corresponding classes of a domain model (Ecore model) were already generated in a model-driven way by EMF. The forward engineers for the iOS front-end variant must additionally implement a corresponding code generator that generates data access objects (DAO) because EMF was not able to generate the Objective-C code. Moreover, the iOS application variant uses a lightweight database instead of the file-based XMI serialization used for the Android application variant. These data-oriented parts of the re-engineered reference applications could be reused later while creating the code generators of the model-driven development infrastructure, because they were already developed in a model-driven way.

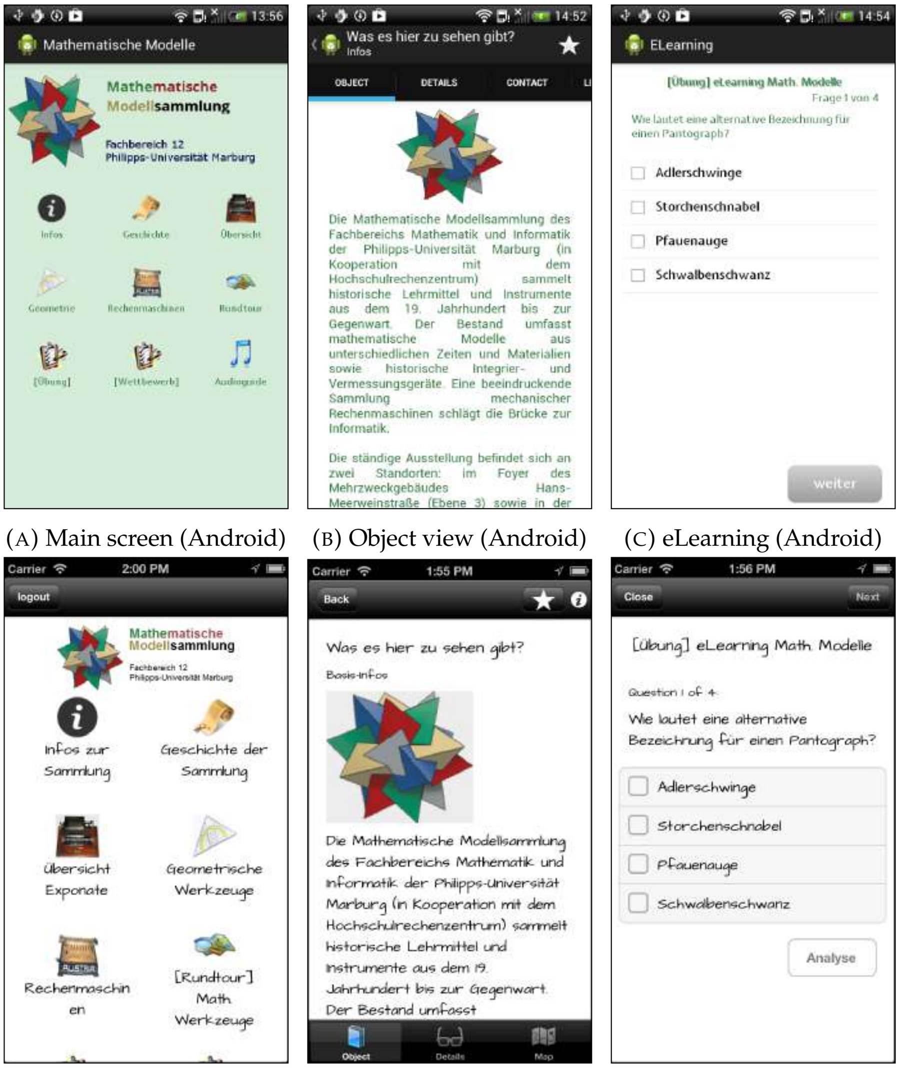
7.35 Due to the requirement of using the original back end, the forward engineers have to write parsers that parse the vendor-specific project file obtained by the back end (e.g., the content management system – CMS) and convert them to the internally used data model. This part of the reference applications will not be used later while constructing the code generators. Instead, it enables carrying out a compatibility test of the forward-engineered front-end applications using the already existing back end. The mentioned parts (i.e. the parsers) can be removed when the forward engineering of the back end is completed.

7.36 Finally, the forward engineers realize the graphical user interface of the application that presents the parsed and converted data. As the mentioned mobile applications provide extensive configuration capabilities of the graphical user interface at runtime, the forward-engineered applications must provide the same configurations.

7.37 Consequently, the forward engineering of the front end provides an Android and an iOS application that reflect several features of *key2guide* and *key2operate*. Figure 7.1 illustrates the forward-engineered reference applications showing data from an university's mathematical collection that are created in the existing back-end system as a test project.

7.2.2 Back End

7.38 The information provided in the mobile applications (Figure 7.1) was obtained from the original back-end system. In the original setting, the analyzed mobile applications need such a content management system because it provides the only facility to create application data and configure the mobile applications. Using EMF as the underlying technology in the forward-engineered mobile applications facilitates the creation of standard model editors for all relevant models, as shown in Figure 6.10. Henceforth, models can be created and edited locally by rich-client model editors and uploaded directly to the mobile devices. Besides, the mobile applications to be generated will provide CRUD functions, which provide self-service functionality to create and modify runtime models directly on the mobile device. Hence, it is at first appropriate to provide a platform just for runtime model distribution rather than creation and editing. A *simple* back end was created to cope with this task. Moreover, to provide web-based editing of runtime models, a *web-based back-end model editor* was developed, similar to the content management system used in *key2guide*.



(A) Main screen (Android) (B) Object view (Android) (C) eLearning (Android)

(D) Main screen (iOS) (E) Object view (iOS) (F) eLearning (iOS)

FIGURE 7.1: Forward-engineered mobile application variants (Android/iOS)

7.2.2.1 Simple Back End

The simple back end (called *SeVVerl* – Serveranwendung zur Verwaltung und 7.39
Verteilung von Laufzeitmodellen; engl. Server application for administration and distribution of runtime models) supports the following features:

- User-roles and groups: the back end implements user management. Different users can be grouped together with their different levels of authorization. Since the back end can administer several runtime models of different types of mobile applications, every user is allocated to a particular application.
- Login and logout of users: corresponding to the mobile front ends, the back end supports the login and logout of mobile devices, i.e., their users.
- Delivery of metadata: following the login, the back end delivers a list of the available runtime models for the particular user. This list of data may vary depending on the user authorization level.
- Download of a selected runtime model: after selecting one runtime model, the mobile application retrieves the runtime model and applicable additional media files, if any.

- Upload of a selected runtime model: users can also upload models to the back end. Here it does not matter whether the runtime model was locally created or downloaded before and modified.
- Deletion of runtime models: users can delete models. This function is only available in the web-based front end of the back end.
- Sharing of runtime models: users can share models. Using this function, the runtime models can be made available to other users or groups.

7.40 The simple back end does not provide any editing functionality apart from the runtime editing in the mobile applications. Besides, concurrently modified runtime models (e.g., modified by two independently working mobile users) cannot be joined together when both are uploaded. In such cases the simple back end provides two update strategies: (i) the existing runtime model will be overwritten completely by a modified and uploaded runtime model, regardless of the loss of already performed modifications of another mobile user; (ii) Modified and uploaded runtime models are stored separately with a new revision number. However, modifications of other users cannot be integrated into this model.

7.2.2.2 Web-Based Back-End Model Editor

7.41 The web-based back-end model editor (*PIMARWebEdit*) was developed to provide a platform-independent model editor for provider models (cf. Section 6.1). This application adopts most of the features of the simple back end, but extends the functionality considerably in terms of model-editing. Figure 7.2 shows the design of the editor's graphical user interface.

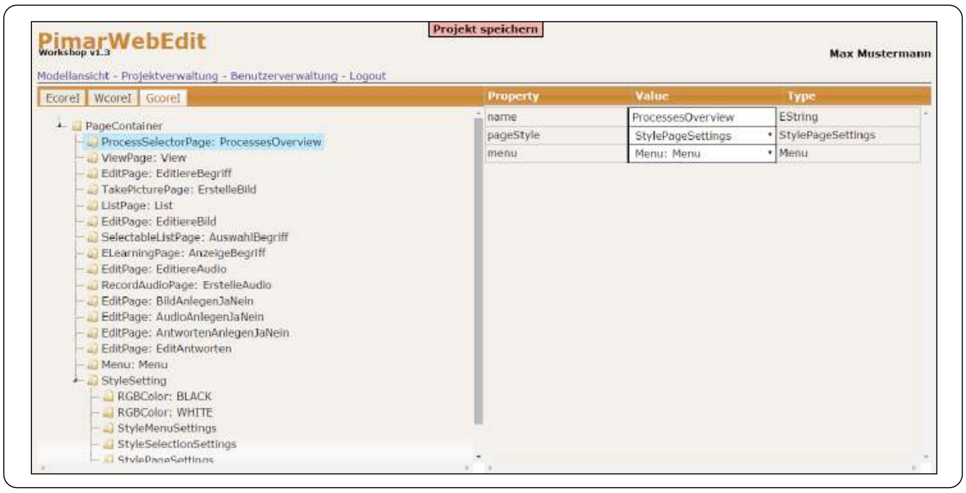


FIGURE 7.2: Web-based model editor for runtime models (editing a GUI model)

7.42 Similar to the simple back end, the web-based back-end model editor provides no features for collaboration in terms of joining the modified models. As shown later, different frameworks such as CDO [59], Teneo/Hibernate [71] [LM10] or EMFStore [KH10] [64] can be used to provide concurrent access to the runtime models.

7.2.3 Evaluation of the Forward Engineered Reference Applications

7.43 To argue that the forward-engineered reference applications are *representative*, two goals must be pursued:

7.44 The first goal concerns the *functionality* of the reference applications. The forward-engineered reference application should fulfill the same specification as the analyzed reference applications. In practice, this can be ensured by a compatibility test. The forward-engineered components were tested against unmodified components, i.e., the forward-engineered front end with the unmodified back end. Another test exploits the already existing test cases for the forward-engineered reference applications in order to demonstrate an equal behavior or correct in- and outputs.

The second goal focuses on the internal structure of the forward-engineered reference applications. The mobile application developers of the original reference application review the forward-engineered mobile applications and point out differences, if any. **7.45**

7.2.3.1 Compatibility Test

The forward-engineered reference applications initially use the unmodified backend to obtain their application data (cf. Figure 7.1). The unmodified back end provides several already existing projects (runtime models), which can be used as representative test data for the forward-engineered applications. Apart from the existing projects, a test record was systematically created by the forward engineers in order to test different configurations of the front end as accurately as possible. **7.46**

7.2.3.2 Functional Test

The mobile application developers of the original reference application test the forward-engineered reference applications based on their existing test cases. However, they are unable to apply test cases of a low test stage (e.g., unit tests). Hence, the already existing *user acceptance tests* (UAT) are reused. Moreover, this test covers device heterogeneity (in the case of Android), because the tester carries out several tests on different device types. All in all, about 60 single user acceptance tests were run for the forward-engineered reference applications. In total, the functional test demonstrates that the *functionality* of the forward-engineered mobile versions is generally equal to the functionality of the initial reference applications. **7.47**

7.2.3.3 Qualitative Review

The qualitative review of the forward-engineered reference applications is particularly important because all quality defects and anti-patterns, as well as architectural issues, will be adopted to the deduced code generators. Therefore, the domain experts review the forward-engineered applications in a final step and fix minor issues. This last step results in the forward-engineered reference application which serves as a basis for the construction of good code generators: **7.48**

7.3 Construction of Code Generators

The construction of code generators includes code generation, the initialization of platform-specific projects of the particular IDEs, and the app model and provider model processing. Finally, the constructed code generators provide functionality to inject custom code into the generated mobile applications. **7.49**

7.3.1 Initialization of the IDEs

Although the model-driven development infrastructure itself is developed and executed on the Eclipse platform, the code generator does not necessarily target **7.50** the same platform for artifact generation. Most existing IDEs for mobile application development provide their own IDE-specific project structure. Moreover, the mobile platforms need different project artifacts (often called manifest files), which contain metadata and further essential information (e.g., API keys). Hence, the code generator must also initialize these IDE-specific project structures and their platform-specific artifacts correctly, in order to subsequently apply the build mechanisms (program-code compiler) of the desired IDEs.

7.3.1.1 Platform-Specific Project Initialization (Android)

7.51 The Eclipse platform is the targeted platform for the Android code generation. The initialization routine at first initiates a new project, adds the required project natures (e.g., Android, EMF), and creates standard sub-folders. After that, the required libraries (e.g., EMF runtime library), the default icons, and an empty Android manifest file are copied to the project. Finally, the Android-specific settings (e.g., targeted Android version) are set. The initialized Android project is then ready for code generation. However, the code generator must be aware of these artifacts and maintain the data in it. For example, a generated Android activity must be registered in the Android manifest file; otherwise, the application may fail during execution.

7.3.1.2 Platform-Specific Project Initialization (iOS)

7.52 The target platform for iOS code compilation and project building is XCode [06]. However, the corresponding code generator runs in Eclipse and writes native program code in a local project folder which is exported from Eclipse and then imported into the XCode IDE. The initialization routine for the iOS project inside Eclipse uses an empty XCode project that serves as a template project. Since Eclipse cannot programmatically access XCode project-specific settings (e.g., add library dependencies), all required artifacts are already created inside the project template. The XCode project is ready for code generation directly after its creation. Similar to the Android project, several files must be maintained during the code generation stage. XCode and Eclipse work together seamlessly because the XCode project may be active in XCode and in Eclipse at the same time. Hence, code and project modifications will be immediately recognized by XCode.

7.3.2 Preprocessing an App Model

7.53 The app model undergoes some preprocessing steps before it is compiled. Apart from an obligatory validation of the app model, the data model will be decorated with some additional methods. Regarding the abstract modeling of standard behavior (e.g., CRUD elements), the preprocessor concretizes the abstract modeling elements automatically. The preprocessing follows a *visitor-based* approach because the preprocessor modifications are done at the modeling level and a visitor mechanism is used to iterate over the model elements.

7.3.2.1 App Model Decoration

7.54 The generated mobile applications require different operations for reading and viewing data objects. The read task performs pattern-based reading, thus requiring the *equals* method to find pattern-matching objects. The *ListablePages* requires the *comparable* method to sort the objects that are listed in a particular order. The *toString* method is used to format item names as part of list elements.

7.55 The mobile application modeler is not bound to model these methods manually, since the preprocessor will add them to the data model in the model decoration step. Nevertheless, the mobile application modeler may add the mentioned methods to create a customized solution instead of using standard behavior. In this case, the preprocessor will not overwrite the manually added methods.

7.3.2.2 App Model Concretization

In the model concretization step, any occurrence of an abstract modeling element is substituted by an appropriate structure consisting of concrete modeling elements. For example, the preprocessor decomposes *CRUD* tasks into an *InvokeGUI* task, which in turn invokes an *EditPage*. Thus, the update functionality can be provided (cf. Figure 6.2a left-hand site) as part of the abstract CRUD process. Create, Read and Delete functionalities are provided in the same manner. **7.56**

Our employed automatic concretization inside the same language level, i.e., metamodel (endogenous transformation), is unusual, since usually, the model-driven development approach seeks to lower the abstraction level only between different language levels, i.e., metamodels (e.g., CIM, PIM, and PSM). However, using both abstract and concrete modeling elements inside a modeling level will ease the modeling process (more compact elements means less modeling effort) and can be efficiently handled by the code generators by reusing concrete transformation rules. **7.57**

7.3.3 Processing an App Model

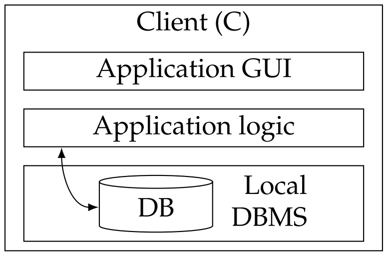
The core functionality of the code generator (model compiler) is the processing of a software model. However, the code generator will not only generate simple and unstructured program code artifacts, but also a runnable mobile application with a defined architecture. Hence, the code generator establishes the architecture of the generated mobile applications. Due to the absence of any architectural modeling construct inside the developed domain-specific modeling language, the code generator implicitly incorporates the resulting architecture of the mobile applications. The general architecture of the mobile applications to be generated is explained below. Afterward, the mapping between the modeling elements and the platform-specific program constructs is presented. Finally, the code generation for Android and iOS is shown in more detail. **7.58**

7.3.3.1 Architecture of Generated Mobile Applications

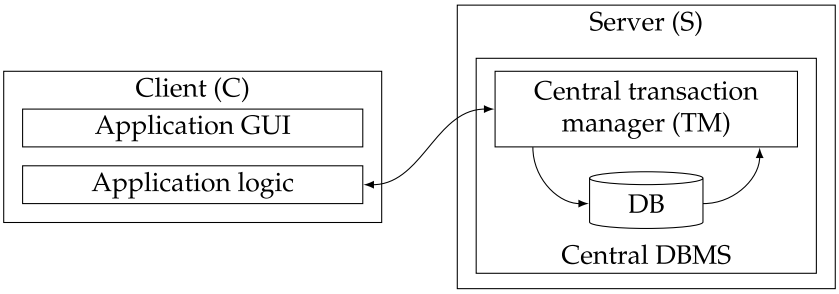
Before giving a detailed presentation of the mapping of the modeling elements to the constructs of the different target platforms and the code generation to Android and iOS, we describe the overall platform-independent architecture of the generated mobile applications. **7.59**

The architecture of generated mobile applications reflects the separation of data, process, and graphical user interface aspects in our app model (as shown in Figure 6.1). Since we generate data-oriented mobile applications [Fra+14, Section 2.1], the architecture of each generated mobile application has a data access layer. This layer contains the modeled data entities (e.g., persons, addresses) and provides functionality to serialize and deserialize these objects. The data layer forms the *model* of the application. **7.60**

The *controller* layer implements the behavior specified by the process model, e.g., 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 produced by user dialogs. Since it is possible to take the process instance model into **7.61**



(A) Local data and transaction management



(B) Centralized data and transaction management

FIGURE 7.3: System architectures of the generated mobile applications

account at runtime, the controller layer contains an interpreter part for the process instance models.

7.62 The presentation or *view* layer provides all interactive dialogs and the graphical user interface. This is the most platform-specific layer because it uses the graphical components provided by the different software platforms. Additionally, the graphical user interface must be dynamically configurable (e.g., according to device-specific features) at runtime. The dialogs in the view layer do not contain any computation or navigation logic (except for input validation) and forward all events to the controller.

7.63 Finally, the architecture of a generated mobile application implements a transaction concept. A process invoked by the initial process selector dialog opens a transaction on the object model. If the user returns to this initial process selector dialog by confirming all the steps, this transaction is completed successfully. Otherwise, the user can cancel the transaction at any dialog, and the changes made are consequently lost.

7.64 Besides the internal structure of the generated mobile applications (clients), the mobile devices can be again part of external system architecture, if connected with a server. Since we are focused on data-driven mobile applications the considered servers are lightweight and only provide a database management system. Figure 7.3 shows the architectural variants that are possible with respect to the features *local* and *central*ized data and transaction management.

7.65 These variants can be instantiated at runtime by the mobile users of the generated mobile applications. The class `DataAccessLayer` is called at the start of the mobile application. The constructor of this class (see Listing 7.1) tries to determine whether a central database was initialized before. In this case, the mobile application will run with centralized data and transaction management (see Figure 7.3b). Otherwise, the constructor checks if the mobile application is preset with local data. In this case, the mobile application will be operated as a standalone application with local data and transaction management (see Figure 7.3a). In case that both cases are not applicable (no mobile application of this type was ever started) the mode is set to an undefined state and the user is asked whether the database should be initialized at the server (centralized) or the client (locally). This case will only occur for the first startup of the mobile application, which may be performed by the administrator of the system.

LISTING 7.1: Determination of data and transaction mode

```
private DataAccessLayer(...) {
  ...
  masterdatabase = new Masterdatabase(getContext());
  if (isOnlineDataAvailable()) {
    setModeOnline();
    masterdatabase.login();
  } else if (isOfflineDataAvailable()) {
    setModeOffline();
  } else {
    setModeUndefined();
  }
}
```

7.3.3.2 Mapping of Model Elements to Platform-Specific Types

7.66 Prior to the presentation of the code generators, we explain our mapping between the platform-independent modeling elements and the platform-specific types and technologies.

Mapping of the Data Model

7.67 The mapping of the data model is not platform-specific, since multi-platform technologies (e.g., platform-independent relational databases) are available. We may use different technologies such as a file-based system (storing data in XMI format) or a database management system (using SQLite or MySQL) to map the data model. Consequently, the mapping of the object-oriented data model follows well-known concepts (e.g., the object-relation mapping [Amb12, Chapter 14]). It is not further regarded in this discussion.

Mapping of the Process Model

7.68 Table 7.2 shows the important model elements of the process model (see Figure 6.7) and the corresponding counterparts in the targeted platforms. We mapped the *Process* element to *Service*s. Services have no graphical user interface and can be started and stopped. An *InvokeGUI* task calls a graphical user interface. Thus, it is mapped to platform-specific graphical user interface constructs (e.g., *Activity* or *UIViewController*). The *Create-*, *Delete-*, and *Read*-Tasks are mapped to simple classes of the platform-specific programming language. An *InvokeOperation* task calls a method and does not differ from a normal method access. The mapping of the *InvokeProcess* task is slightly different for each of the supported platforms. While Android provides the *Intent*-construct to call Services (or other mobile applications), iOS has no counterpart. Thus, in iOS, the *InvokeProcess* task is mapped to an instantiation of the corresponding service class. This mapping reflects the main design decisions with respect to code generation based on the domain experience of mobile application developers and suggestions provided in the relevant literature. As we have used a set of reference applications (and forward engineered reference applications) to develop our domain-specific modeling language, as well as getting appropriate code snippets for the code templates, we reuse this bottom-up mapping from code to language elements in reverse.

TABLE 7.2: Mapping the model elements to platform-specific constructs

Modeling element	Android	iOS
Process	android.app.Service	Service (NSObject[1])
InvokeGUI task	android.app.Activity /FragmentActivity	NSObject.UIResponder. UIViewController
Create-, Delete-, Read-Tasks	java.lang.Class	NSObject
Variable	Declaration of field	Declaration of field
InvokeOperation task	Invocation of a method	Invocation of a method
InvokeProcess task	android.content.Intent	Invocation of Service

Mapping of the GUI Model

7.69 In contrast to the clear mapping of the main process model elements, GUI model elements (see Figure 6.5) cannot be mapped to platform-specific types in a straightforward way. There are two reasons for this:

7.70 First, the GUI model and the style model (if available) are mainly interpreted by the mobile application at runtime. Thus, a mapping of modeling elements and platform-specific constructs is hardly possible. The generated mobile applications contain generic code to interpret the model information, and as a result, neither the generated declarative descriptions of the graphical user interfaces nor the generated code contain hard-coded information about the graphical user interface modeled earlier.

7.71 Second, the generic code relates to several components of the graphical user interface (e.g., labels, text views). The amount of graphical user interface components depends on the type of *Page* (e.g., *ViewPage*, *EditPage*) and the data model (see Figure 7.5). Thus, the amount and location of generic graphical user interface interpreting code are indirectly affected by the app model.

7.3.3.3 Code Generation for Android

7.72 The code generation process begins automatically after changing and saving an app model (auto generation). To avoid the processing of temporarily invalid models while editing them, the mobile application modeler can deactivate the auto generation for the time being. The code generator produces at least two projects (see Figure 7.4) – an Android project <Project>.Android (e.g., Phonebook.Android) containing the Android application program code, and an Android library project <Project>.Lib (e.g., Phonebook.Lib), which contains the data layer program code. The mandatory Android library project serves two purposes: (i) it makes entity classes of the data model available for use by the Android project, primarily by Android activities. Hence, provider models can be easily handled by Android activities using objects instead of raw parameter lists. (ii) The second purpose of the Android library project is the permanent storage of provider models. For this purpose, the Android library project refers to further libraries to (de)serialize and modify the provider models at runtime. The Android library project is created by reusing the existing EMF generator that generates code for the EMF runtime. The generated code and the EMF runtime are directly applicable on the Android platform. The EMF generator becomes a sub-generator of the complete code generator and processes the Ecore data model separately. Alternatively, it is possible to use SQLite instead of EMF. The process model and GUI model are translated by separate sub-generators written in Xtend, which is a contribution of this work.

7.73 The main Android project follows the usual Model-View-Controller [Lia14, p. 47] [Med+12, 171 ff.] architecture of Android applications. View components are mostly

[1] NSObject is the root class of most Objective-C class hierarchies.

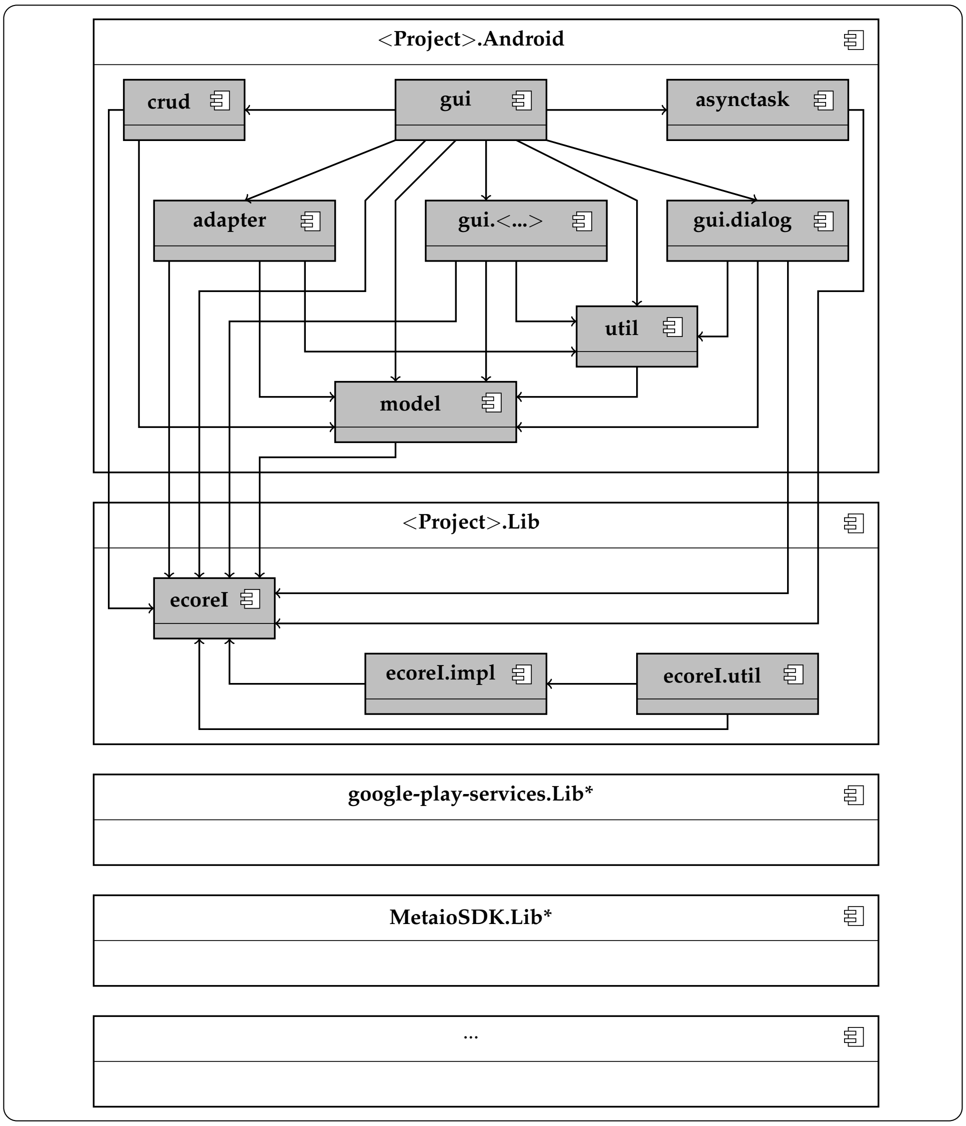


FIGURE 7.4: Architecture of generated Android applications

generated as mobile application resources. The controllers contain the modeled application logic and occur in the form of activities (*gui*, *gui.dialog*), fragments (*gui.<...>*), adapters (*adapter*), services (*gui*), asynchronous tasks (*asynctask*), and simple Java classes (*crud*). While entity class interfaces are widely dispersed in these controllers, access to runtime models is done exclusively via the *model* package. The *model* package acts as a data access layer, ensuring that Android activities do not access runtime models directly. Therefore, the generated application architecture can be easily adapted to other technologies (i.e., relational databases, web services) for (de)serialization of provider models by just changing this layer (cf. code generation for iOS).

Example (Automatic generation of the Graphical User Interface). Figure 7.5 shows the generated activity layout for the process *CRUDPerson*. As mentioned earlier, the mobile application modeler does not define the hierarchical structure of the graphical user interface components in an explicit manner. This information is deduced from the data model. The mobile application developer only specifies that the process *CRUDPerson* displays an object of type *Person* (see Figure 6.3) to modify. The code generator produces a standard layout for this task (cf. Tran [Tra+12] and **7.74**

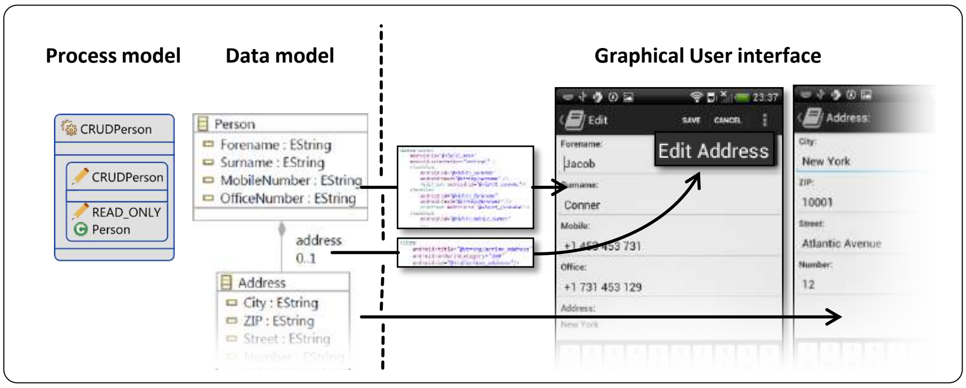


FIGURE 7.5: The generated graphical user interface of the process *CRUDPerson*

Raneburger [Ran+12]). □

7.75 Further library projects (whose use is optional) encapsulate the utility of map services (e.g., Google Play Service [12]) and AR functionality (e.g., MetaioSDK [26]).

7.76 All these projects are immediately compiled and are then ready to run. By default, the memory card of the mobile device contains an initial provider model containing an empty object model, an initial style setting, and an initial process instance model containing the main process with all those processes assigned to the main process in the app model. This provider model may be extended during runtime. After app model changes (which result in code regeneration), it might become partly invalid, depending on the kind of changes. If, for example, 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 left up to future research to support automated migration of provider models.

7.3.3.4 Code Generation for iOS

7.77 The workflow to generate iOS code is nearly the same as for Android. The code generator produces one project (see Figure 7.6) – an iOS project <Project>.iOS) (e.g., Phonebook.iOS) containing the iOS application program code. A slight difference, however, is that the generated project must be exported from Eclipse and imported into the XCode-IDE in order to build an iOS application.

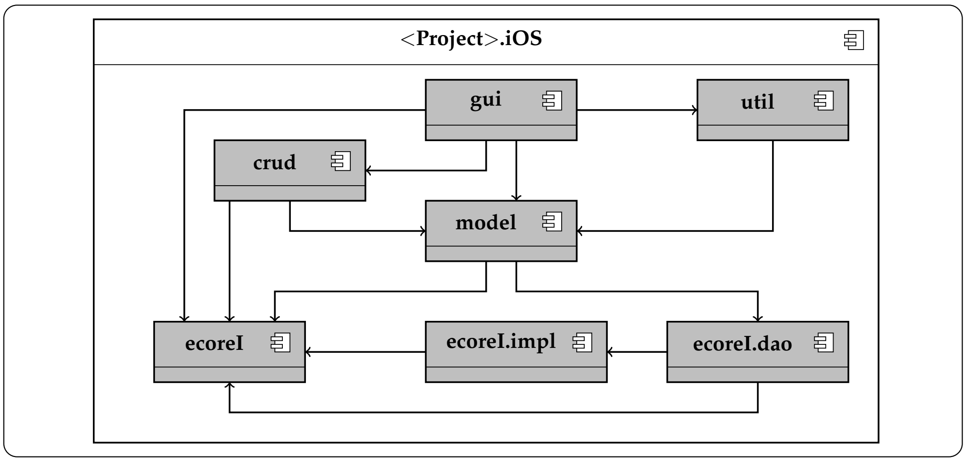


FIGURE 7.6: Architecture of generated iOS applications

7.78 In contrast to the Android code generator, the iOS code generator currently creates only one project. The code generator for iOS cannot reuse the EMF generator to process the Ecore data model since EMF-generated code cannot be run on the iOS platform. This functionality must be covered by the code generator for iOS in addition. Hence, the EMF-equivalent code for iOS comprises entity classes deduced from the data model (*ecoreI*, *ecoreI.impl*) and corresponding data access objects (DAOs [Yen+14, p. 154]) to (de)serialize objects. As the *model* package indicates, the generated iOS applications use a relational database (SQLite [17]) to store runtime models. An initial database is created based on the data model using Teneo (which is based on Hibernate). This database contains an initial provider model. Similar to the Android platform, the database might become partly invalid after app model changes and regeneration of code. The remaining packages follow the same architectural design as presented for an Android application. In this sense, they are platform-specific equivalents to previously described Android packages.

To show geographical maps, the generated iOS applications use the built-in Apple Maps Service. A library project such as google-play-services.Lib is not necessary, but possible, for the iOS platform. Since Apple provides an increasing number of similar services to Google (e.g., iCloud/Google Drive, iMessage/Google Messenger) additional third-party libraries are less important to integrate a certain service. The only exceptions to this are third-party libraries that provide specialized functionality (e.g., AR functionality). **7.79**

7.3.4 Processing a Provider Model

In addition to the high-level description of the provider model and its capabilities to configure several options of generated mobile application given in the introduction, we will now describe in detail which model elements will be evaluated at runtime. **7.80**

With regard to the object model as an instance of the data model (cf. Figure 6.4), all objects will be evaluated, i.e., processed, at runtime. The generated mobile application can modify the object model during runtime. The object model is generally many times larger, in terms of the number of elements, than the data model. **7.81**

Similar to the object model, all elements of the style model will be evaluated at runtime. A GUI model may contain one general style setting for all pages. A style model may contain one style setting element per page. Therefore, a style model may have twice as many elements as the GUI model. A style model will not be modified by the generated mobile application, in its current usage. However, it might be useful that mobile users are allowed to change the style model to set their personal preferences. **7.82**

Finally, the process instance model is the most limited sub-model of the provider model. For several reasons, the process instance model allows only the redefinition of the *ProcessSelector* tasks. Hence, the process instance model never contains as many elements as the process model itself. The process instance model may be changed during runtime due to contextual changes, but the mobile end users are not allowed to modify the process instance model. **7.83**

The provider model and its sub-models are loaded at the start-up of the mobile application and configure it accordingly (cf. Section 5.3.2). Modifications of the object model are committed at the end of a process execution or rolled back in case of process cancellation. **7.84**

To sum up, a provider model manages the data, style, and the available processes in a generated mobile application at runtime. **7.85**

7.3.5 Injection of Custom Code

7.86 Automatic code generation often limits the possibility of modifying or adding program code manually. Four possible strategies deal with this problem:

7.87 A trivial strategy involves *changing the paradigms*: at first, mobile application developers use a model-driven development approach to generate a mobile application. They make several iterations and try to cover as many requirements as possible. At some point, they stop using the model-driven development approach and change to a traditional development approach but keep the generated mobile application as a basis for further implementations.

7.88 The *protected region* strategy is often used by model-driven development infrastructures that generate only parts of a system, i.e., structural elements or mock-ups. Subsequently, mobile application developers may fill these gaps by adding program code inside certain regions. These regions are respected during regeneration, and the manually inserted code is kept. Thus, code generation and manual coding can be coexistent. However, when the model is changed in such a way that the structure of the resulting implementation cannot be kept, this approach will turn out to be useless or involve at least, user interaction.

7.89 The *generation gap pattern* has already been explained in Section 6.8 since we applied it to extend the generated parts of the model editor with custom program code.

7.90 Finally, our strategy to enable custom code writing and adding is based on *model annotations*. These model annotations are also used to introduce external libraries.

7.3.5.1 Platform-Specific Model Annotation

7.91 Since custom program code usually has the lowest abstraction level, any annotation of custom code changes a computation-independent model (CIM) or platform-independent model (PIM) to a platform-specific model (PSM). To avoid this abstraction level switch, annotations should contain only cross-platform language program code. However, we use only platform-specific model annotations, which change the originating CIM/PIM app model to a PSM app model.

Customizing the Behavior

7.92 Custom behavior can be realized by the annotation mechanism for class methods, as shown in Listing 6.1, which is already supported by EMF. The annotation is a string-typed field comprising Java (Android) or Objective-C (iOS) program code. The class methods currently accept only one annotation, i.e., Java or Objective-C program code. The code generators (iOS, Android) adopt the annotated code automatically while creating the corresponding classes. Concerning the construction of code generators, it would be valuable to use different annotation keys to map annotations to a particular platform. For example, the Android code generator should accept only annotations that have a corresponding key for this platform and ignore other annotations. Consequently, the annotations are platform-specific, whereas the model might be platform-independent further on due to multiple platform-specific annotations covering all relevant platforms.

Customizing the Graphical User Interface

7.93 Just like they can use the injection method to realize custom behavior, mobile application modelers might want to customize the graphical user interface. To this end, the modeling element *CustomPage* provides annotations for custom layouts. In accordance to the Model-View-Controller pattern, the code generators generate default controllers for this user-defined *CustomPage*. The generated default controller depends on the particular usage of the *CustomPage*, which can be detected by its invocation context. We identify four different applications of a *CustomPage*:

First, if the *CustomPage* provides an input object (e.g., a Person), the mobile appli- **7.94**
cation modeler intends to view this object. Hence, the controller will provide the values of the object. In turn, the annotated custom layout must provide the same data structure as the input object.

Second, if the *CustomPage* provides an output object (e.g., an Address), the mobile **7.95**
application modeler intends to edit this object after creation. Hence, the controller will write the values gained from the graphical user interface to the object. Accordingly, the annotated custom layout must provide the same data structure as the output to the input behavior. This also applies to the types of the graphical user interface elements (e.g., setting a Boolean attribute of the output object requires a checkbox and setting a string attribute requires an editable text field).

Third, if the *CustomPage* provides both an input and an output object, the mobile **7.96**
application modeler intends to modify an existing data object. Hence, the controller outputs already defined object values and gets modified ones.

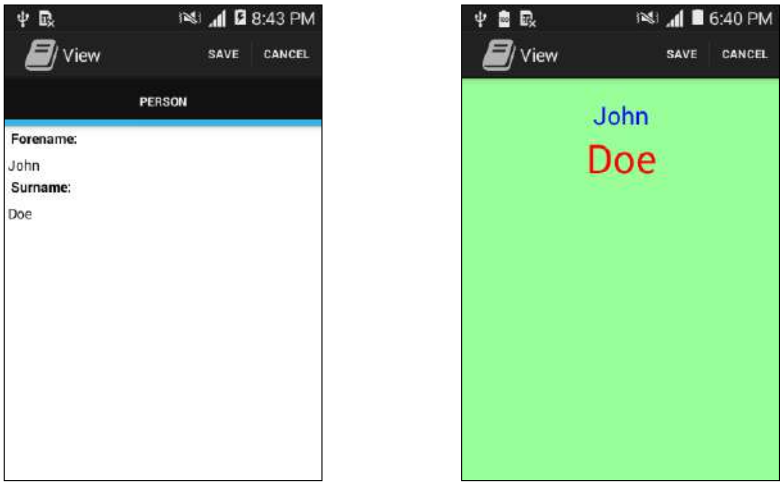
Thus, the first three modes of the *CustomPage* application provide customized **7.97**
graphical user interfaces while dealing with the view and acquisition of data.

Finally, the fourth scenario applies if no input or output objects are modeled during **7.98**
the invocation of the *CustomPage*. The corresponding controller does not provide or require any data. It loads only the annotated layout defined by the mobile application modeler. Due to being independent of data, the layout can be designed without restrictions.

Example (Customizing a graphical user interface). Based on the existing simple **7.99**
phone book application, the process *AllPersons* (cf. Appendix B.1.3.6) provides the selection of a *Person* object in order to subsequently show this object in a detailed view (see Figure 7.7a).

LISTING 7.2: Layout annotation of a custom page (excerpt)

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="#99FF99"
    ... >
    <TextView
        android:id="@+id/tv_forename_data"
        android1:layout_marginTop="20dp"
        android:gravity="center"
        android:textColor="#00f"
        android:textSize="25sp" />
    <TextView
        android:id="@+id/tv_surname_data"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="#f00"
        android:textSize="40sp" />
    <TextView
        android:id="@+id/tv_mobileNumber_data"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:visibility="invisible" />
    <TextView
        android:id="@+id/tv_officeNumber_data"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:visibility="invisible" />
</LinearLayout>
```



(A) Generated GUI for viewing a *Person* object

(B) Customized GUI for viewing a *Person* object

FIGURE 7.7: Different graphical user interfaces for viewing a *Person* object

This refers to the first variant of *CustomPage* usage. To customize the generated graphical user interface for viewing a *Person* object, mobile application modelers change the invoked page type from a *ViewPage* to a *CustomPage* inside the task *InvokeGUI* (CustomViewSelectedPerson) of the process *ViewPerson* (cf. Appendix B.1.3.2). The introduced *CustomPage* contains the layout annotation shown in Listing 7.2, which results in the graphical user interface shown in Figure 7.7b. □

7.3.5.2 External Library Inclusion

7.100 The inclusion of external libraries facilitates the access to external functionality that constitutes merely an extension, rather than a customization, of the generated mobile applications. However, external libraries may contain custom code written by mobile application modelers. As shown in Figure 7.4, the generated mobile application can be bundled with other libraries. Combined with annotated code that invokes these libraries (called *hooks*), the generated mobile application can be extended considerably with external functionality.

7.3.6 Code Generator Implementation

7.101 Finally, the Eclipse plugins, namely *pimar.generator.frontend.ios* and *pimar.generator.-frontend.android*, implement the code generators for the desired platforms (cf. Table 7.3). A separate plugin for the back end generation of each platform is provided inside the code generator projects respectively.

TABLE 7.3: Fourth set of plugins shaping the MDD infrastructure

Plugin project name	**Content**	**Type**
pimar.generator.frontend.ios	Code generator (iOS)	manual
pimar.generator.frontend.android	Code generator (Android)	manual
pimar.generator.frontend.android.test	Test cases (Android)	manual

7.102 The code generators implement a builder mechanism and can be registered in an EMF-based project that contains app models (cf. Figure 6.19). The code generators automatically track each change in the model files and regenerate the application code. However, they might deactivate temporarily until the mobile application modeler completes a part of the model.

The plugin *pimar.generator.frontend.android.test* implements test cases for the Android code generator. Currently, a test-suite for the iOS code generator has not been implemented yet. However, manual test cases can be reused because the structure of the resulting mobile applications (Android and iOS) is similar and their behavior should be identical. **7.103**

7.3.6.1 Similarity Between the Code Generators

A review of the code generators shows that both code generators have a similar structure but also some differences which are not reasoned only by the different implementers. Since the Android generator was implemented before the iOS generator, the iOS implementer tried to adopt as many architectural design decisions as possible. Both generators provide similar packages for the interaction of the code generator plugin with the IDE and particular the common used graphical model editor. The mechanisms to setup the project folder structure of the IDEs are also similar. As shown in the Tables 7.4 and 7.5, the Android code generator is more structured, i.e., provides more packages. In turn, not all packages and the corresponding content are necessary for the iOS code generator. For example, the iOS platform does not require external layouts, menus, and strings (cf. A.9) and adapter or dialog (A.2 and A.3) generators. Thus, the iOS generator has no directly corresponding packages, but rather uses implicit equivalents inside the denoted packages (I.2; I.3; I.4). **7.104**

TABLE 7.4: Packages and content of the iOS code generator

No.	Packages (pimar.*)	Description	Maps to
I.1	.generator	Setup of the project folder structure and meta-files	A.11
I.2	.generator.controller	Creates the process selector, the sequence of task invocations and the predefined CRUD element	A.1
I.3	.generator.controller.crud	Creates the individual CRUD operations like Create, Read, and Delete	A.6
I.4	.generator.controller.-invokegui	Creates the invocations of GUIs and the different pages (e.g., EditPage and ViewPage)	A.4; A.5
I.5	.generator.infrastructure	Plugin code for adding the IOS generator to the Eclipse IDE	A.8
I.6	.generator.model	Generates the DAO classes	A.11
I.7	.generator.util	Utility code used by the Plugin	A.12

The alignment and shared use of similar and especially non-generating parts of the code generators might reduce the effort for maintenance of several code generator implementations. Nonetheless, we recommend that platform-specific code generators should reflect the targeted platform and intended artifacts (e.g., declarative layouts). Based on our experience it is very useful to generate human-readable code and respect the intended project structure of the desired software platform. This ensures an easy adoption of prototypical code which is written and tested in a generated prototype to the code templates. **7.105**

The similarity between the resulting mobile applications is discussed at the end of this chapter as part of the evaluation. **7.106**

TABLE 7.5: Packages and content of the Android code generator

No.	Packages (pimar.*)	Description	Maps to
A.1	.generator.controller	Creates the process selector, the sequence of task invocations and the predefined CRUD element	I.2
A.2	.generator.controller.adapter	Creates List adapter	I.2; I.3; I.4
A.3	.generator.controller.dialog	Creates Dialogs	I.2; I.3; I.4
A.4	.generator.controller.-invokegui	Creates the invocations of GUIs	I.4
A.5	.generator.controller.-invokegui.pagegenerator	Creates different pages (e.g., Edit-Page and ViewPage)	I.4
A.6	.generator.crud	Creates the individual CRUD operations like Create, Read, and Delete	I.3
A.7	.generator.helper	Helper classes used by the code generator templates to create code snippets	I.7
A.8	.generator.model	Generates the DataAccessLayer which imports the EMF Library project (<project>.Lib)	I.6
A.9	.generator.resources	Generator dependent Android resources (e.g., Menus, Layouts, Strings)	I.2; I.3; I.4
A.10	.generator.util	Utility classes to copy static code to the generated project	I.1
A.11	.infrastructure	Plugin code for adding the IOS generator to the Eclipse IDE and setup of the project folder structure and meta-files	I.1; I.6
A.12	.infrastructure.util	Utility code used by the Plugin	I.7

7.4 Evaluation

7.107 The preceding sections presented the facilities for code generation. Together with the infrastructure components presented in Chapter 6 the model-driven development infrastructure can be used by mobile application developers for creating mobile applications in a platform-independent way. Hence, we will now discuss whether or not the requirements given in Chapter 4 are now satisfied completely (Question 1). Moreover, since our model-driven development infrastructure is able to generate mobile applications, we come up with the following question: do the generated mobile applications reflect the focused features and the features of the initially provided reference applications (Question 2)?

7.108 As mentioned during the evaluation of the domain-specific modeling language, the appropriateness of a developed domain-specific modeling language depends not just on the compliance to generally accepted design guidelines, but also on the acceptance of the desired user group, i.e., the mobile application developers that use this domain-specific modeling language. Hence, we ask: how easily can the domain-specific modeling language and the model-driven development infrastructure be adopted by users of different skill levels (Question 3)?

7.109 Finally, we evaluate more technical aspects of the designed model-driven development infrastructure by discussing questions such as: how similar are the code generators and the generated mobile applications for the particular platforms, i.e., Android/iOS (Question 4)? Do the code generators work correctly (Question 5)? Do the code generators scale (Question 6)?

7.4.1 Setup

To answer the first set of questions (Question 1 and Question 2), we again use the acceptance criteria of the requirements to demonstrate that the created artifacts within this chapter cover and comply with the requirements. To show that the generated mobile applications reflect the focused features and the features of the initially used reference applications, we discuss three case examples. These case examples, i.e., the mobile applications and their features, cover most of the features given in Table 7.1. **7.110**

In order to answer the question of how easily the model-driven development infrastructure could be adopted by users of different skill levels (Question 3), we conducted several hands-on tutorials. We will report on the qualitative observations. **7.111**

The set of technical questions (Question 4 – Question 6) is discussed by presenting the mapping of the generated Android and iOS applications (Question 4), providing a test suite for functionality tests of the code generator (Question 5) and conducting scalability tests to test the runtime behavior of the code generators (Question 6). **7.112**

7.4.2 Requirement Coverage (Question 1)

The parts of the overall implementation of the model-driven development infrastructure which are shipped by the plugins denoted in Table 7.3 cover the requirements 4.2.1, 4.2.2, and 4.3.2 (cf. Table 4.1). **7.113**

The plugins *pimar.generator.frontend.ios/android* (cf. Table 7.3) implement the aforementioned requirements completely: first, they form the code generator component as part of the tooling (cf. Section 4.3.2). Second, they determine the architecture of the generated mobile applications. Within this part of the thesis, we focus on data-driven mobile applications (cf. Section 4.2.1) which form a single-user standalone system with back-end access (cf. Section 4.2.2). **7.114**

The data-driven acceptance criterion posits that the generated mobile applications (Android and iOS) must be able to serialize the acquired data. Using the functionality described in Section 7.3.3.2 (Mapping of the data model), the designed model-driven development infrastructure fulfills this criterion for the desired platforms. **7.115**

The designed model-driven development infrastructure generates mobile applications that are able to work as a standalone system with back-end access as described in the Sections 7.2.2.1 and 7.2.2.2. Finally, the acceptance criteria for the code generator requirement are fulfilled if a representative set of models can be used for the generation of runnable mobile applications for different platforms. This holds true to the extent as the case examples and the functionality test demonstrate that the code generators are able to create runnable mobile applications for many different app models and, in turn, the test models reach a high coverage of the code generator. **7.116**

7.4.3 Feature Coverage (Question 2)

The designed model-driven development infrastructure primarily allows the *model-driven* development of mobile applications, which is a key contribution of the first part of this thesis. The model-driven development approach employs *abstract* and **7.117**

TABLE 7.6: Focused features and coverage by the model-driven development infrastructure

Feature group ▷ ... ▷ Feature/s	Case example 1	Case example 2	Case example 3	**Covered by the MDD infrastructure**
Application type ▷ Standalone	✓	✓	×	✓
Application type ▷ Information system	×	×	✓	✓
Application type ▷ Transaction system	×	×	×	(✓)
Software platform ▷ Android	✓	✓	✓	✓
Software platform ▷ iOS	×	✓	×	✓
Hardware platform ▷ Screen ▷ Resolution	×	×	×	(✓)
Hardware platform ▷ (Actors/Sensors/Interfaces/Memory)	✓	✓	×	✓
Application architecture ▷ Native application	✓	✓	✓	✓
Application architecture ▷ Interpreter application	×	×	✓	✓
Application architecture ▷ Hybrid application	×	×	✓	✓
Application development ▷ Model-driven	✓	✓	✓	✓
Model-driven ▷ Design model (Abstract/Detailed)	✓	✓	✓	✓
Model-driven ▷ Runtime model	×	×	✓	✓
Data and transaction management ▷ Local	✓	×	×	✓
Data and transaction management ▷ Central	×	×	✓	✓
Graphical User Interface ▷ Dynamic	×	×	✓	✓
Context-awareness ▷ Platform context-awareness	×	✓	×	✓

*detailed design model*s and *runtime model*s. Since the domain-specific modeling language is platform-independent, the available code generators compile the models to different *software platforms* (currently Android and iOS).

7.118 According to the model-driven development approach, which does not focus on technical details, the code generator conceals the technical details and the architecture of the mobile applications. The implicit architecture of the generated mobile applications is *native* but contains also *interpreter* functionality (used during the implementation of the context-awareness in the second part of this thesis). Hence, we call the application architecture of the generated mobile applications *hybrid application*. Due to the nativeness of the generated mobile applications, different *actors*, *sensors*, *interfaces*, and *memory* of the hardware platforms can be accessed. The type of the generated mobile applications can be *standalone* (with preset data) or an *information system* if a back end is used. This requires *local* data and transaction management. Additionally, transaction systems with central data and transaction management can be realized.

7.119 Table 7.6 shows the covered features. Compared with the features that are covered by the initial reference applications (cf. Table 7.1) now all features are covered. Please note that the features *transaction system* and device-specific *resolution*, as well as corresponding case examples, will be covered/introduced in the second part of this thesis. Moreover, we will introduce and discuss additional features in the second part.

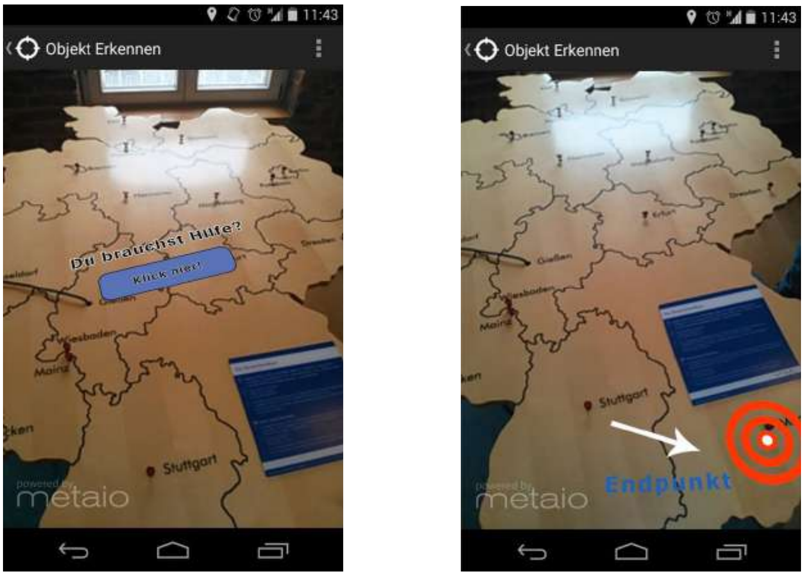
7.120 To demonstrate the applicability and usefulness of the model-driven development infrastructure, we consider three case examples for the first part of this thesis. Please note that the following prototypes do not yet demonstrate the two-level modeling approach using provider models or the context-aware behavior. Such prototypes will be shown later in the second part of this thesis.

7.4.3.1 Case Example 1 (Mathematikum)

Our first case example is a mobile application designed for a local science museum, 7.121
called *Mathematikum*[1]. The Mathematikum is a museum with a huge collection of exhibits that explain complex mathematical topics in a playful manner, making them accessible for laypeople and children. To follow this philosophy, the mobile application requires a very intuitive and highly interactive graphical user interface and should not only present mere text-based information statically.

FIGURE 7.8: Caesar's ciphered text "Knack den Code" (Break the code)

To meet this requirement, two game-like activities are implemented. One of them 7.122
is concerned with an exhibit where a ciphered text has to be decoded by clever guessing (cf. Figure 7.8). Successful guesses are shown to the mobile user. Moreover, they see the text that still has to be deciphered. The second game-like activity uses Caesar's cipher for ciphering short texts and sending them by SMS or a messaging application like WhatsApp.

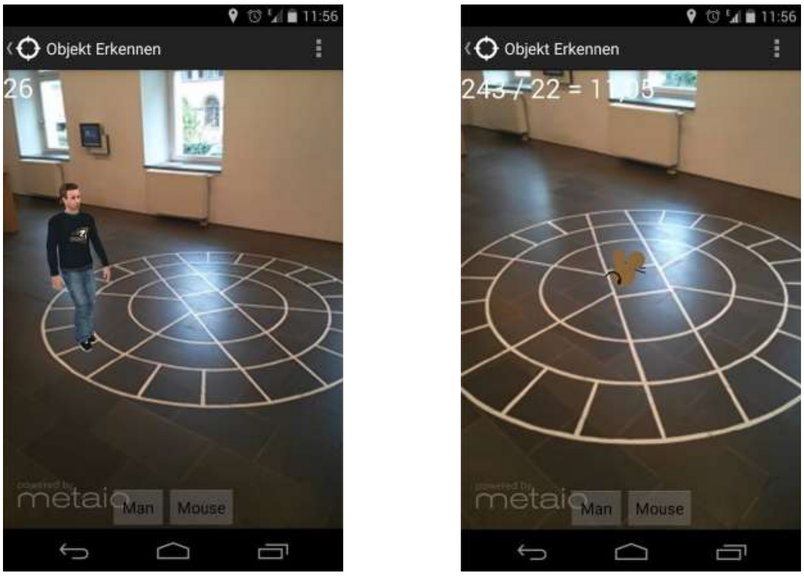


(A) Object detection and support offering (B) Overlayed solution hint

FIGURE 7.9: Traveling Salesman Problem "Deutschlandtour" (Germany Tour)

Using augmented reality, users can get interactive guidance to understand exhibits. 7.123

[1] http://www.mathematikum.de/



(A) Overlayed animation (Man)   (B) Overlayed animation (Mouse)

FIGURE 7.10: π Approximation “PI-Kreis” (PI-Circle)

This functionality is used here to interactively explain the Traveling Salesman Problem (TSP). When visitors try to solve this problem hands-on by building a tour through Germany, the mobile application can give them hints on finding the right solution (see Figure 7.9). To demonstrate how π can be approximated (dividing the sum of different-sized footsteps around a circle by the diameter of this circle), augmented reality features are used to create animated explanation scenarios (see Figure 7.10). The mobile application is available at [57].

7.124 The Mathematikum application is a standalone application that operates permanently offline. Hence, the data and transaction management are local. It is first generated for the Android platform, i.e., a native implementation is generated. A native iOS version could also be generated, but customized code (annotated in the app model) must be written again for iOS. The model-driven development process of this application requires only a design model, as a start- or runtime adaptation of the graphical user interface or the behavior was not required. The application makes use of the device hardware (e.g., the built-in camera). The corresponding features of this mobile application are marked in Table 7.6 (Case example 1).

7.4.3.2 Case Example 2 (SmartPlug)

7.125 The second case example demonstrates how a generated mobile application can deal with external hardware devices. The SmartPlug application provides wireless remote control for home appliances and electronics. Users can turn off and on electronic devices that are attached to a manageable power distribution unit (e.g., NETIO-230B [22]) controlled by SmartPlug. SmartPlug logs the electricity usage and can forecast the costs for electrical energy based on the logged switching intervals. Figure 7.11 shows the main use cases of the mobile application.

7.126 The main menu (Figure 7.11a) contains the use cases *Configure device*, *Switch devices*, *Show protocol*, and *Power consumption*. The use case *Switch devices* (Figure 7.11b) shows the registered devices and current states (in brackets). The screen *Power consumption* (Figure 7.11c) shows the calculated energy consumption broken down to the devices. The mobile application uses manually written code (extending *EOperations*) and existing libraries (e.g., java.net.Socket) to establish the connection

(A) Main menu (B) Switch devices (C) Power consumption

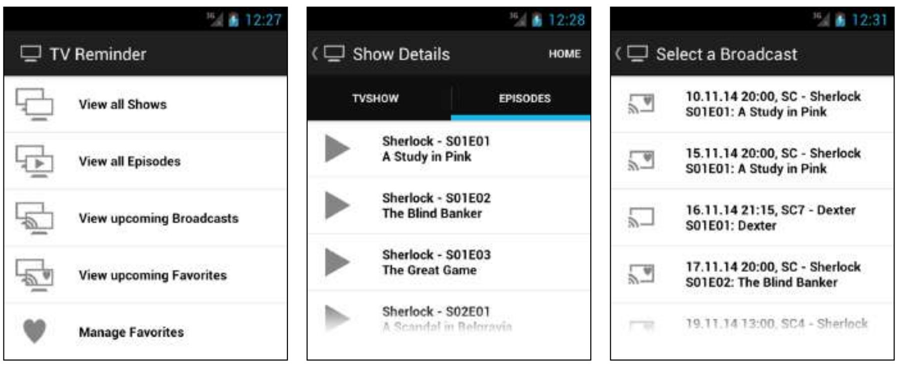
FIGURE 7.11: The mobile application *SmartPlug*

to the external device. After an adaptation of the manually written platform-specific code (i.e., *EOperations*), the app model could be used to generate a mobile application version for iOS.

The SmartPlug application is a standalone application that operates permanently **7.127**
offline (except for the connection to local devices through a Wi-Fi, NFC, or Bluetooth connection). The app model of the SmartPlug application does not contain a lot of customized code, and can thus be easily used for generating native implementations for Android and iOS without much effort. Consequently, the development process is platform-aware, as different software platforms can be operated. The SmartPlug application can interact with external hardware (e.g., sensors/actors) through the interfaces of the mobile device. Table 7.6 (Case example 2) shows the features of this mobile application.

7.4.3.3 Case Example 3 (TV Reminder)

The third case example shows a data-oriented mobile application that provides **7.128**
mostly standard processes and tasks (e.g., CRUD functionality). The TV Reminder mobile application organizes the broadcast times of the mobile user's favorite TV shows. By browsing the list of upcoming broadcasts, the mobile user can view and select favorites. The selected favorites are shown in a separate list in ascending order to get a quick overview of the upcoming broadcasts. Additionally, the user may export the selected favorites to the calendar to get a notification when a TV show begins. The mobile application creates a calendar entry based on the broadcast time and duration by using the calendar provider. Contrary to the mobile applications



(A) Main menu (B) View upcoming broadcasts (including favorites) (C) Detail view (Tv show and episodes)

FIGURE 7.12: The mobile application *TV Reminder*

presented before, the TV Reminder does not provide any means to create, update, or delete broadcasting elements. Moreover, the detail view of TV shows/episodes can be customized at runtime by a runtime model. Figure 7.12 shows the main use cases of the mobile application.

7.129 The TV Reminder application is an information system that relies on a central database, i.e., its data management is centralized. It is generated for the Android platform, but can be generated for iOS as well. Its native implementation allows the interpretation of data and style models at start- and runtime. This requires a model-driven development approach that contains both design- and runtime models. Thus, we call its application architecture hybrid. The features of this mobile application are summarized in Table 7.6 (Case example 3).

7.4.3.4 Potentials and Limits

7.130 During the development of the shown example applications, we learned that our domain-specific modeling language can be applied best if the mobile application focuses on data management. Data structures, their graphical user interface representation, and the CRUD functionality can be modeled on a high abstraction level. Our case examples also reveal that individual behavior may be added, sensors may be used, and external devices may be controlled. Relatively small numbers of model elements compared to huge amounts of generated code boost productivity for this kind of mobile applications. The situation may differ for mobile applications that focus less on data management. The Mathematikum application, for example, has a larger amount of game-like behavior that is not generated, but manually coded. In particular, this application shows that code generation and manual coding can be integrated seamlessly. It also shows the limitations of our model-driven development approach in its current form. Although some high-level behavior can be specified by abstract modeling elements (e.g., CRUD processes), and simple logic can be modeled using the control structures (e.g., If-Else, While, Assign), it is preferable to hand-code more complex behavior. The reason for this does not necessarily lie in the limited expressiveness of the domain-specific modeling language. Rather, it is a matter of convenience to code complex algorithms manually and in a text-based way instead of modeling them on a similar abstraction level in a graphical model editor using a visual domain-specific modeling language.

7.4.4 User Experience Evaluation (Question 3)

7.131 In addition to the rather technical evaluation of the domain-specific modeling language along the design guidelines presented before, we also validate the infrastructure (i.e., the graphical model editor and the code generators) with respect to socio-technical aspects. In fact, one requirement is to support mobile application modelers coming from different areas, e.g., domain experts, technical experts. Hence, we test the model-driven development infrastructure during different stages of development with different groups of testers.

7.132 During a period of 16 months (02/14-06/15), 10 workshops were organized and about 75 participants in total used the model-driven development infrastructure. The workshop participants implemented the phone book application shown in Appendix B by following the guided tutorial which is shown in Appendix C. Apart from these guided workshops, the model-driven development infrastructure was tested during a master's course. The students of this course could freely realize a self-developed mobile application idea after a short introduction. Similarly, a group of 5 experts from industry realized a mobile application for a safety briefing. Finally, the model-driven development infrastructure was used within several advanced learners' laboratory courses. In the following, we report about the experiences and cluster them to the different user groups:

7.4.4.1 Undergraduate Testers

Since undergraduate testers are generally skilled neither in mobile application development nor in model-driven development, this group of testers had naturally the most problems. While creating app models, they generally go by the instructed modeling approach, but a variety of modeling mistakes such as wrong cardinalities, incomplete models, wrong, or untyped attributes could be recognized by the instructors. These issues are not atypical of modeling novices, regardless of the technique (e.g., UML, EMF, and BPMN) or tooling used. In turn, prepared examples and sample solutions help most of the participants to master the generation and build process and come up with a runnable mobile application in the end. Thus, the majority of the undergraduate testers are able to generate mobile applications, realize minor changes to app models, and perceive the effects of these changes. **7.133**

7.4.4.2 Graduate Testers

The group of graduate testers had generally fewer problems while modeling the phone book application. Although they, like the previously mentioned group, make modeling mistakes, the graduate testers use the validation facilities of the graphical model editor very exhaustively. Hence, they are able to fix most of the errors without the help of the instructors. After passing the generation and build process of the model-driven development infrastructure a couple of times, they are able to follow the tutorial without further help from the instructors. Some participants interact independently to the tutorial and experiment freely with the model-driven development infrastructure. However, the generated mobile application may crash due to incorrect modeling. The testers were usually incapable of mapping runtime exceptions to the error-causing part of the app model. **7.134**

7.4.4.3 Expert Testers

The group of expert testers exhibits the same problems as the group of graduate testers. Based on their higher skills of mobile application development, they use more code annotations even if modeling elements would provide the same functionality. Due to their high technical skills, the experts tend to circumvent the model-driven development infrastructure because they feel limited. Despite this, the participants like the automatic generation of the mobile application for the platform which they are not familiar with (e.g., Android or iOS). **7.135**

7.4.4.4 General Observations and Results

According to our question, how easy the proposed approach could be adapted by different user groups (Question 3); we are convicted that it could be applied easily by potential users of any user group. Although the effectiveness and accuracy of modeling show a wide range due to different skill levels of the participants, mostly every participant was able to model and generate at least one mobile application. This gives evidence that the designed model-driven development infrastructure could be applied in general. **7.136**

When we take a closer look at the individual problems of the participants, we identify that technical problems (e.g., how to model a cardinality) are more dominant than conceptual issues. The participants adopted the domain-specific modeling language very quickly due to the example-driven presentation, but had several problems with the tooling and the setup of the development environment (e.g., installing the appropriate software development kit and configuring the emulator). **7.137**

The most benefit a mobile application developer can take when using a model- **7.138** driven development infrastructure is achieved when s/he knows how the input (i.e., the app model) must look like to reach the goal, i.e., a mobile application that satisfies certain functional requirements. Mobile application developers who use (a new) model-driven development infrastructure for the first time often build a mental model that reflects modeling actions and the resulting code structure. More precisely, they internalize the transformations that are performed by the model-driven development infrastructure, i.e., the code generators.

7.139 Our observation is that participants with a low skill level in both areas (modeling and code writing) poorly perceive what the model-driven development infrastructure can do and thus cannot use the tool well. In turn, highly skilled mobile application developers and domain experts are able to reconstruct the transformations made by the model-driven development infrastructure very quickly and know what is to be done in order to complete an assignment. Further on, we recognize that non-technical experts (i.e., domain experts) depend more on the features of the domain-specific modeling language and an up-front description. In contrast, technical experts dislike paying attention to documentation or verbal instruction about the domain-specific modeling language and prefer sample models, even if they do not understand them fully. They adopt the domain-specific modeling language by investigating the generated mobile applications, i.e., the generated native program code. Therefore, the participants could be divided into groups of *language-oriented* learners and *transformation-oriented* learners (cf. Table 7.7).

TABLE 7.7: Observation made during the user experience evaluation

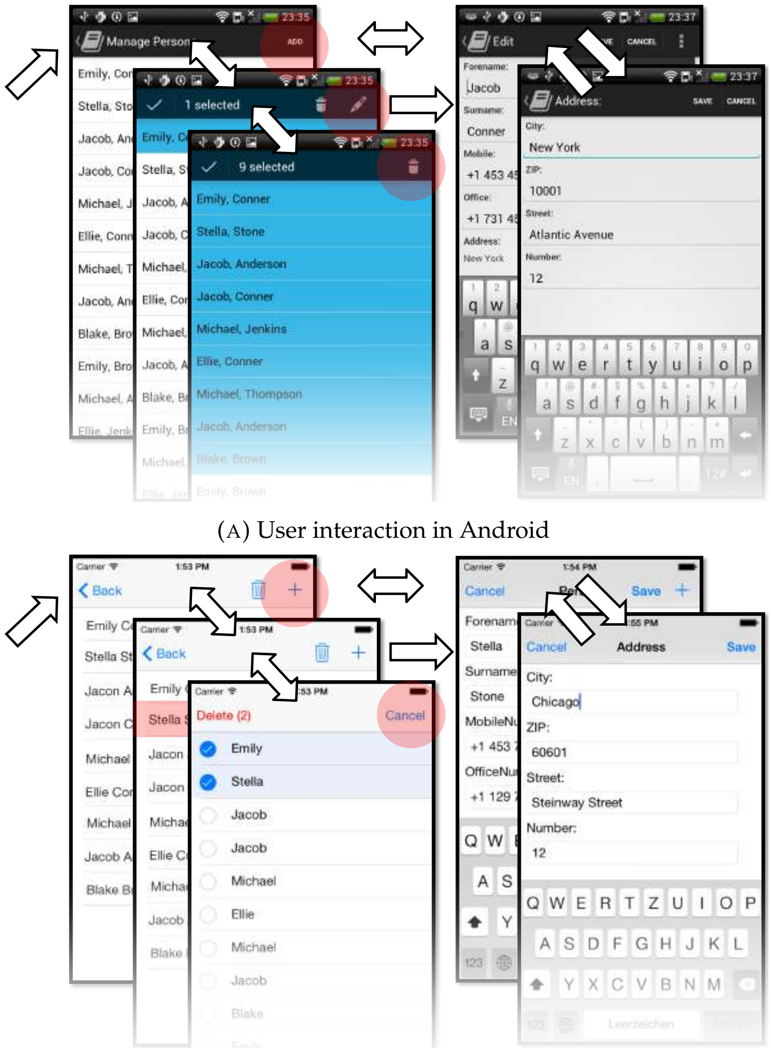
Modeling skills	Coding skills	Profit of MDD	Learning approach
Low	Low	Low	Unstructured
High	Low	High	Language-oriented
Low	High	High	Transformation-oriented
High	High	High	Language- and/or Transformation-oriented

7.4.5 Similarity Between Applications (Question 4)

7.140 In addition to the presentation of the internal architecture of the generated mobile application presented in the Sections 7.3.3.3 and 7.3.3.4, we will also discuss and compare the structure of the generated mobile applications from a mobile user perspective.

7.141 The review of the generated prototypes (iOS and Android application variants) is based on the used app model for the phone book example. Both the Android and the iOS generators process the same app model and generate the projects mentioned in Sections 7.3.3.3 and 7.3.3.4. The process model includes 12 processes and 47 tasks. The data model contains three classes (see Figure 6.4). Finally, the GUI model given in Figure 6.6 contains five pages of different types and appropriate style settings. From the structural point of view, we can confirm that both generated mobile applications follow the presented architecture based on a review of the selected app model. The only difference is that different technologies are used inside the architectural layers (model, controller, and view). Besides, we always find technical and logical counterparts in both architectures. From a behavioral point of view, we compare the site map (order and structure of appearing screens) of the prototypes, the runtime artifacts (i.e., object-, process instance-, and style-model), and the transactional behaviors. Figure 7.13 shows an excerpt of the site map of the prototypes.

7.142 For the 10 user-interacting processes, we can find equal site maps. As an example, we show this for the process *CRUDPerson* in Figure 7.13. The transactional behavior



(A) User interaction in Android

(B) User interaction in iOS

FIGURE 7.13: Sitemaps of the process *CRUDPerson*

of both prototypes is nearly the same: the only difference is the underlying persistence framework. The Android prototype uses EMF libraries to load and store runtime models (XMI), while the iOS prototype uses a SQLite database to store and load data.

7.4.6 Code Generator Testing (Question 5 and 6)

The code generators are tested from a functional perspective. Additionally, their execution times and their scalability are tested. **7.143**

7.4.6.1 Functional Tests

Testing a code generator differs from a traditional software test in many ways. **7.144** Model-driven development infrastructures handle abstract software models and generate complex software systems, i.e., native program code. During the development of the model-driven development infrastructure, the phone book app model (cf. Appendix A) is used as a reference model for both code generators (iOS, Android). Testers (being at the same time infrastructure developers) check the resulting program code for the syntactical correctness and if runnable, correct runtime behavior.

7.145 Additionally, several automated functional tests (unit tests) were defined and applied to the implementation of the Android code generator. In order to implement functional tests, the testers specified (parts of) test models as an input and parts of native program code as an expected output of a code generator. A test framework (e.g., JUnit) executes these tests automatically and reports the results.

7.146 Testers usually follow a structured approach to create representative test models (e.g., the *category-partition method* proposed by Ostrand and Balcer [OB88]).

7.147 Indeed, the specification of the expected output represents a challenge: comparing the output – native program code – as a byte or character stream makes the test quickly unusable because even semantically equivalent changes inside the code template cause a test fail because the syntax may differ.

7.148 Thus another approach applies techniques of model transformation testing [Bau+06]. Before a test, the generated native program code must be parsed in order to create another model that conforms to the programming language metamodel. Subsequently, the expected output can be defined also in this programming language metamodel and compared with the parsed model.

7.149 A very practical approach is the random test. Several test models are generated based on the given metamodel [Bro+06] of the domain-specific modeling language and tested during a test run. The expected outcome need not be specified. The only criterion is the acceptance of the generated native program code by the program-code compiler. Hence, the program-code compiler (called *oracle*) identifies app models that lead to errors. This approach is very helpful in the early stages of code generator development.

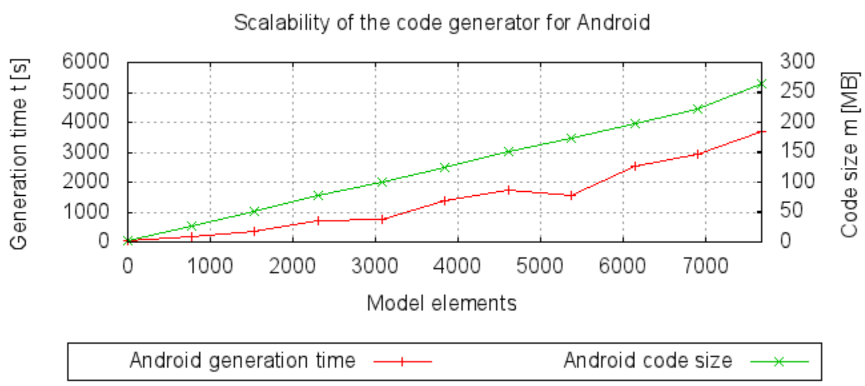
7.150 To sum up, the Android generator can be tested in an automatic way with a code-coverage of about 40%. The test suite contains 87 test cases for automatic tests (i.e., unit tests). The code coverage is not higher since our main focus is on the parts of the code generator which are related to the generation of native program code (e.g., the packages A.2 till A.4 and A.9 from Table 7.5). Other packages contain code that is related to the plugin functionality or do not produce native program code of the resulting mobile applications. This plugin code might be tested with other tools such as Jubula [68] or the RCP Testing Tool [69]. Moreover, five test models (modeling real mobile applications) are used for manual testing of the Android and iOS code generators.

7.4.6.2 Scalability Tests

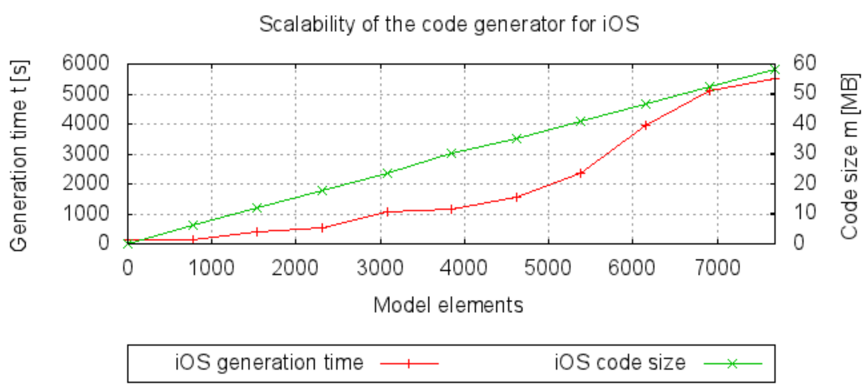
7.151 The model-driven development infrastructure was tested with design models of different sizes until the generated mobile applications reached the technical limitations of the desired platforms.

7.152 The Android platform below Version 5.0, i.e., the underlying Dalvik machine, supports mobile applications with less than or equal to 64k methods. A mobile application that exceeds the 64k limit cannot be compiled directly in the selected IDE (Eclipse ADT). The Android platform above version 5.0 supports mobile applications beyond 64k methods. The model-driven development infrastructure supports such applications too, but they must be built with another IDE (Android Studio). The iOS platform has no program-code compiler limit but requires that the resulting mobile applications will not exceed 4GB (02/2015).

7.153 To test scalability, we tested the model-driven development infrastructure with design models that meet the mentioned limits, i.e., the generated mobile applications follow the mentioned sizes. These test models were synthetic app models. These synthetic app models contain different classes and relation types (aggregation, association), all task types, and all types of pages in the respective sub-model parts. This representative set of model elements is then duplicated to meet the number of model elements shown in the Figure 7.14. Assuming that the mentioned limits will increase over time, we are interested in the general runtime behavior of the model-driven development infrastructure. Thus, we carry out scalability tests with different sizes of the design models. Figure 7.14[2] shows the result of the test in terms of generation duration and code size.



(A) Scalability of the code generator for Android



(B) Scalability of the code generator for iOS

FIGURE 7.14: Scalability of the code generators

Figure 7.14a shows how the Android code generator scales with models that range from a few model elements up to about 7,500 model elements. The left y-axis (associated with the red plot) denotes the generation time in seconds [s]. The right y-axis (associated with the green plot) denotes the code size of the generated native program code in megabyte [MB]. Figure 7.14b shows the same data for the iOS code generator. **7.154**

We remark on two observations: (i) Figure 7.14 shows the worst cases of the code size growth because each model element causes a native program code fragment in this case. By naming equal parts (e.g., Read tasks that read objects of the same type) identically, the output of code and thus the project size could be reduced considerably because the code generators reuse generated code. In terms of memory complexity, the model-driven development infrastructure provides a sub-linear behavior in the average case. (ii) The generated native program code must be subsequently compiled by the project-specific environments (e.g., Eclipse, Android Studio, or XCode). The generation and build durations are nearly equal. Hence, the **7.155**

[2] Tested on an Intel Core CPU M520 i5 2.4 GHz; 8GB RAM

generation of native program code will not dominate the overall build process of mobile applications.

7.156 Additionally, the scalability of the runtime models, i.e., the provider models, was also tested. While the process instance and the style models cannot scale up to any size due to the limitations presented in Section 7.3.4, the data model may scale up to any size. To demonstrate the scalability of the generated mobile applications, we load data models of different sizes and log the load-time duration and the allocated memory size for the Android platform. Figure 7.15 shows the results of these measurements[3].

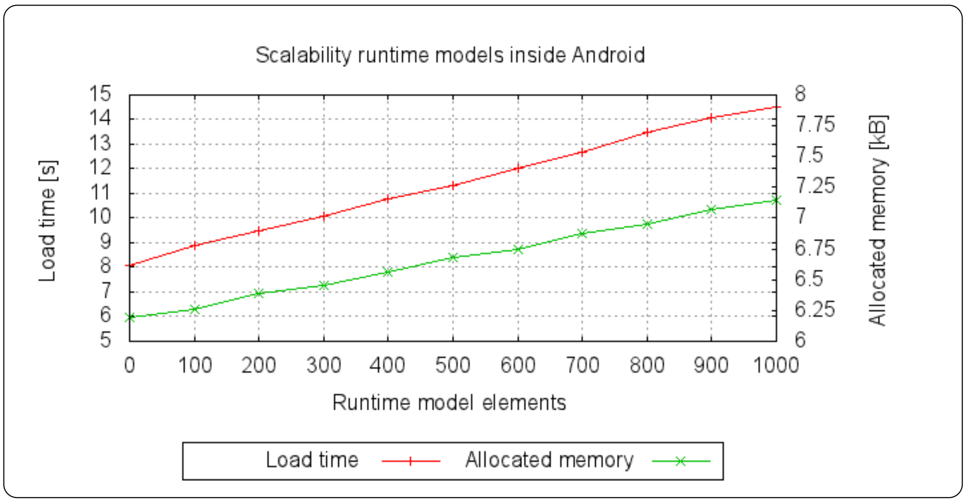


FIGURE 7.15: Scalability of Android runtime models

7.157 The diagram in Figure 7.15 shows how the Android applications scale with different sizes of a runtime model. The sizes range from a few model elements to about 1,000 elements. The left y-axis (associated with the red plot) denotes the load-time needed from the start of the application until the data can be seen on the screen in seconds [s]. The right y-axis (associated with the green plot) denotes the allocated main memory of the device in kilobyte [kB].

7.158 These experiments show that the model and code size scales linearly, no matter which code generator is considered. The generation time for iOS is higher than for Android, but the compilation time dominates the building process on both IDEs (Eclipse, XCode). Moreover, the runtime model also scales very well with respect to the loading time and allocated memory size. In summary, our experiments show that the model-driven development process with the developed model-driven development infrastructure is not critical in terms of scalability.

7.4.7 Threats to Validity

7.159 Although we discussed several questions during the evaluation of this Chapter, we will now focus on the question of learnability and user adoption of the model-driven development infrastructure (Question 3), since this question is subject to many risks in terms of result validity.

7.160 Considering external validity, the reactive effects of the experimental arrangement may influence the learnability of the designed model-driven development infrastructure in a positive way. In turn, users which are not embedded in an experimental arrangement (e.g., a workshop with an instructor) may show a lower performance because a personal instruction will be more effective as written documentation or

[3] Tested on a Samsung SM-G130HM Single Core 1GHz; Android 4.4.2

another kind of static support material. Another threat to the external validity is the selection of users. Since the most experiments (i.e., workshops) were conducted with students in an academic environment, the participants were generally open-minded to learn and test new concepts and tools. Hence, personal motivation might be a noteworthy bias.

A threat to internal validity is the unknown *experimental mortality*. It describes **7.161**
that (mostly underperforming) participants will not continue a given task in an experimental setting. In our case, the participants could use the prepared solutions for each part of the workshop to keep pace with the group. To deal with this circumstance, we applied the workshop in a slightly changed setup and asked participants to model an individual mobile application.

Chapter 8

Related Work: Model-Driven Development of Mobile Applications

In the following sections, the model-driven development approach presented in this thesis is compared with other approaches for the cross-platform development of mobile applications. First, we discuss the advantages and disadvantages of a model-driven development approach in comparison non-model-driven cross-platform approaches. Since this work focuses on model-driven development approaches, the discussion of related work is limited to approaches and frameworks which are model-driven. **8.1**

8.1 MDD vs. Non-MDD Cross-Platform Approaches

Although a model-driven development approach is also a cross-platform approach by nature, there are cross-platform approaches (e.g., Apache Cordova [03], Mobile Together [01], SAP Mobile Platform 3.0 [47]), which provide no design model, i.e., they cannot be classified as model-driven development approaches. According to the classification given by Heitkötter et al. [Hei+12], these cross-platform approaches require interpreters (runtime employing approaches) for each desired platform or cross compilers which compile a platform-independent codebase to a particular platform., i.e., native program code. **8.2**

Mobile application developers must often spend more effort to learn a new cross-platform programming language within a non-model-driven development cross-platform approach, than they spend learning a new domain-specific modeling language within a model-driven development approach. This is firstly because the cross-platform programming languages are less abstract as the domain-specific modeling languages. Secondly, the use of a domain-specific model during the development process provides a better distinction between technical aspects and business concerns. Domain experts are not confronted with too many new concepts or technical details. Hence, an advantage of the model-driven development approach is a better learnability of the domain-specific modeling language and the reduction of technical details, which are left to the code generators. This advantage does not necessarily hold for high-skilled technical experts. **8.3**

An abstract domain-specific modeling language is also beneficial regarding the platform evolution. While the low abstraction level of the cross-platform programming language may cause considerable effort for platform migration due to manual modification of technical details, the platform evolution could be handled more easily by the model-driven development approach. In the best case, a platform migration only cause changes in the code-generating part of the infrastructure. The existing models can be reused for code generation without any modification, and the migrated version of the mobile application will be the result. In less favorable cases, the domain-specific modeling language must be modified (e.g., extended by new language elements) and existing models must be transformed. However, depending **8.4** on the changes of the domain-specific modeling language, the transformation of the model instances is tool-supported.

8.5 A wealth of meta-tools (e.g., model quality tools, model transformation tools, model differencing, and model analysis) provide additional functionality which can be used beneficially to create mobile applications of high quality. For example, a codebase written in a platform-independent cross-platform programming language could not be easily transformed according certain aspects (e.g., device contexts). The use of models is advantageous when the implementation is to be changed and analyzed.

8.6 A general disadvantage of model-driven development approaches is the limitation to the intended function and architecture of the generated (mobile) applications. Mobile application developers are often annoyed that model-driven development infrastructures are limited beyond a certain point and the injection of custom code is often experienced as cumbersome compared to traditional coding. Non-model-driven development cross-platform approaches often provide the ability to use custom libraries and plugins written in platform-specific, native program code (cf. Figure 2.7). This enables handling non-standard requirements flexibly and individually. In turn, the easy introduction of libraries and custom plugins inside non-model-driven development cross-platform tools will not enforce the reuse of existing or general functionality which may lead to quality issues. Hence, based on a quality-oriented viewpoint an agile bottom-up development process of model-driven development infrastructures could ensure quality goals as wells a flexible extension of the model-driven development infrastructure.

8.7 A further advantage of the model-driven development approach is the capability to generate native implementations. Since some non-model-driven development cross-platform approaches follow an interpreter architecture, these mobile applications might be limited in several aspects, i.e., hardware access or security concerns.

8.8 To sum up, we demonstrate that the model-driven development approach has several advantages, such as a concise abstract domain-specific modeling language which are well-suited for non-technical mobile application developers, a separation of technical and domain concerns, a powerful platform migration capability, and a toolkit of meta-tools for different purposes. A disadvantage almost every model-driven development approach is the insufficient flexibility with respect to individual requirements which have not been considered by the infrastructure developers.

8.2 Comparison Criteria for MDD Approaches

8.9 Prior to the discussion to the model-driven development approaches, some criteria should be introduced in order to make the different approaches more comparable:

8.10 *Modeling aspects*: The modeling aspects describe the expressiveness of the domain-specific modeling language. Due to the lack of a standard mobile modeling language, a lot of model-driven development approaches reuse existing modeling languages from the domain of software-system modeling (e.g., UML). Hence, we evaluate how the approaches cover the modeling aspects of data, behavior, and graphical user interface by their (adopted) general-purpose modeling languages.

8.11 *Modeling techniques*: From a mobile application modeler's perspective, modeling techniques are important. First, a mobile application modeler is interested to see how mobile applications models can be expressed, e.g., by a graphical concrete syntax or a concrete textual syntax and how they look like. According to the state-of-the art process of model-driven development, these approaches differ in model usage (cf. Figure 2.2). For example, some of the approaches generate directly from the CIM to native program code, instead of using intermediate models such as PIM or PSM. Since most of the model-driven development approaches provide code generation from a design model and support no model execution or interpretation, we will discuss the latter in the chapter on related work in the second part of this thesis (cf. Chapter 15).

Mobile application implementation: A major property of a model-driven development approach is the program code generation, i.e., the generation of native or cross-platform program code. While identifying model-driven development approaches in the literature, we have to distinguish carefully between these technologies. For example, in their categorization of cross-platforms and languages, Ribeiro et al. [RS12, Table 2] exclude all approaches that lack a domain-specific modeling language (cf. Palmieri et al. [Pal+12, Table 1]) from the category of model-driven development approaches, because model-driven development requires a domain-specific modeling language in order to increase the level of abstraction. **8.12**

Mobile application architectures: The architecture of a mobile application is strongly related to the kind of application implementation. Since cross-platform implementations mostly tend to be web-based client-server architectures, the native application often realizes a rich-client architecture. **8.13**

Unfortunately, the literature on the different model-driven development approaches for mobile applications generally contains limited or no information as to how the model-driven development approaches and frameworks were developed, particularly which software development process model was used for the model-driven development infrastructures. Thus, we cannot position our agile bottom-up development approach clearly regarding this aspect. Additionally, the literature provides only insufficient information about user evaluation and practical experiences when using the designed model-driven development infrastructures in practice. **8.14**

8.3 Comparison with Related Approaches

We will consider the following approaches as the most relevant work related to our contribution: **8.15**

MD^2: MD^2 [Hei+13] [Hei13] is a model-driven development framework for cross-platform (Android, iOS) development of data-oriented mobile applications. The used domain-specific modeling language is organized in three parts with respect to Model-View-Controller pattern. Thus, the first part of the modeling language that describes the format of data entities is called *data model*. The *view model* specifies the graphical user interface of the mobile application at a detailed level (e.g., defines widgets). Finally, the *controller model* links the *view model* and the *data model*, and describes the behavior of the resulting application. The behavior of a mobile application is described in an event-based manner. Different kinds of events can be handled by modeled *actions*. Based on the modeling techniques, the MD^2 approach provides a concrete textual syntax (similar to the Human-Usable Textual Notation – HUTN [32]), corresponding model editors, and code generators that compile models to the mobile applications for different platforms. Model execution is not provided. The generated mobile applications are implemented in the native language of the desired platform. The generated mobile application has a client-server architecture. The model-driven development framework additionally generates a back end (JEE application) used by mobile applications of both platforms (Android, iOS). **8.16**

MD^2 shows a high similarity to our approach and has a lot of well thought-out solutions. The authors point to many well-known issues (e.g., concept mismatches between Android and iOS and alignment). There are differences in the *top-down* approach of MD^2. In contrast to our agile bottom-up development approach using reference applications, the authors of MD^2 decide on the features of their model-driven development infrastructure in a top-down manner by prioritizing the frequently found requirements. Further limitations with respect to our approach include the limitation of modeling business logic inside the *controller model* and the **8.17** customization of generated mobile applications. Mobile applications that are generated with the MD^2 model-driven development infrastructure follow a client-server architecture. This limits the offline capability of the generated mobile applications because the data is stored on the server and cannot be reached in disconnected situations.

8.18 Mobl: Mobl [HV11a] [HV11b] is a model-driven development framework for the rapid development of data-driven mobile web applications. The used domain-specific modeling language is organized in two parts with respect to the components of the Model-View pattern (a modified version of the Model-View-Controller pattern). Controller functionality is not modeled by the user, but added by the code generator. The data model provides facilities for defining classes with typed attributes. Moreover, the *data model* contains an object-oriented sub-language, thereby allowing adding program code to the data model. The *view model* provides screens as main elements. Screens comprise different kinds of widgets (e.g., buttons, labels, and text fields). Additionally, control elements (e.g., buttons) can be used to link different screens, i.e., to invoke other screens. Besides, the *view model* contains data bindings to establish a direct connection to the *data model*. Based on the modeling techniques, the Mobl approach provides a concrete textual syntax, corresponding model editors, and code generators that compile models to deployable server applications coded in HTML/CSS/JavaScript. The resulting web application provides different web services for the exchange of data. Direct model execution is not provided even if the generated artifacts are interpreted locally by the web browsers of the mobile clients. As any other web-based approach, the application implementation is a cross-platform implementation. The overall architecture of the system is a client-server architecture.

8.19 With respect to our approach and the majority of other approaches, Mobl shows a lot of differences. The framework is strongly focused on data-oriented mobile web applications with standard behavior only. Hence, custom behavior and business logic cannot be modeled, nor can the generated standard behavior be customized. Although the authors claim that the generated mobile application can store data locally and thus the interruption of the network connection may not affect the mobile user negatively, a restart of the browser or the system might cause a problem in terms of data availability.

8.20 JUSE4Android: JUSE4Android [SA14b], [SA14a], [Sil+14] is a model-driven development framework for single-platform (Android) development. The approach uses no domain-specific modeling language in the narrow sense. A common UML class diagram serves as the domain model. This model must be equipped with annotations that focus on graphical user interface definition, persistence, and declare which classes of the diagram are domain classes as part of a PIM. According to the modeling techniques, the JUSE4Android approach provides different concrete syntaxes (via UML class diagram definition), but no corresponding model editor to add the mentioned annotations to an app model. The code generator compiles the annotated model to a native implementation (currently only Android). However, model execution is not provided. The generated mobile application has a client-server architecture. The additionally generated back end provides the functionality of storing and distributing data objects to and from different front ends (mobile clients).

8.21 JUSE4Android is very similar to our approach, especially with respect to the contributions (e.g., online- and offline capability) made in the second part of this thesis. The JUSE4Android approach realizes extended client-server architecture (cf. Figure 1.4). Due to a replication mechanism, the generated mobile applications can access the data even if they are offline. After reconnecting to the server, they can synchronize locally modified data. Since JUSE4Android demonstrates a good architectural design that matches up well with business information systems (BIS), the modeling concerns are not very elaborated. Apart from the domain model, only little information can be added and the generated behavior is strongly limited to CRUD functionality. A key difference between JUSE4Android and our approach is the modeling technique. JUSE4Android is based on UML and uses annotations. Hence, a proper definition of application-specific behavior is not possible. As we see in the second part of this thesis, such a behavior model is an essential requirement to generate conflict-free, offline-capable mobile applications.

Modagile: Modagile [Küs+13] is a model-driven development framework for cross-platform (Android, iOS, Windows Phone) development of mobile applications. The used domain-specific modeling language is divided into four parts: a mobile application model consisting of *DomainEntities*, *DomainAdapters*, *UIContainers*, and *Events*. Similar to the MD^2 approach, the behavior of the resulting mobile applications is defined in an event-based manner. The domain entities are typed by an Ecore model. The graphical user interface layout can be modeled granularly using abstract, platform-independent widget types. Based on the modeling techniques, the Modagile approach provides a graphical concrete syntax, corresponding model editors, and code generators that compile models to the mobile applications for the different platforms. Similar to the other approaches, models are not executed at runtime. The generated mobile applications are implemented in the native language of the desired platform. The generated mobile application has a rich-client architecture. **8.22**

Compared to our approach, we do not follow such a detailed modeling approach regarding the graphical user interface. Additionally, we do not employ an event-based behavior modeling approach and follow a more process-oriented modeling approach. Nonetheless, the Modagile [Küs+13] approach is technologically quite similar to our approach, as it is also based on EMF. **8.23**

The model-driven development frameworks developed by Serral et al. [Ser+10] [Muñ+06], Ceri et al. [Cer+07], Escolar et al. [Esc+14], and Kapitsaki et al. [Kap+09] support additional different contextual dimensions. We include these approaches and frameworks in our discussion of related work herein, but the detailed description of these approaches follows in the second part of this thesis because the focus of these approaches is on context-awareness. **8.24**

Other frameworks and tools such as AppInventor [23] [Wol+14], Canappi mdsl [20], and arctis [Kra11] could not be evaluated, because internals about the domain-specific modeling language, the application architecture, or used target languages were not (or no longer) available at the time of evaluation. **8.25**

More comprehensive reviews of the existing approaches — not just limited to model-driven development approaches — are provided by Umuhoza and Brambilla [UB16] (results are partly shown in the middle part of Table 8.1), Le Goaer and Waltham [GW13], Gaouar et al. [Gao+15], El-Kassas et al. [EK+15], and Charkaoui et al. [Cha+14]. **8.26**

Table 8.1 shows the model-driven development approaches for the development of mobile applications and the classification of their main characteristics. A ✓-symbol indicates that the corresponding feature is supported, whereas an ×-symbol indicates the opposite. An "-" indicates that this feature has not been evaluated. **8.27**

Most of the model-driven development approaches focus on model compilation and thus generate native program code. Only one of the evaluated approaches generates no program code and follows a model interpreter approach. Besides, client-server architectures are more often present. Unfortunately, most frameworks come with their own domain-specific modeling language. The lack of a standard modeling language for mobile application development makes it difficult in general to compare the different approaches. **8.28**

As seen in Table 8.1, the criteria *annotation* and *textual* of the criteria group *modeling techniques*, and the criterion *cross-platform* of the criteria group *application implementation* are not supported by our approach. Since the *annotation* criterion is usually used only in combination with general-purpose modeling languages (GPML), this **8.29**

TABLE 8.1: Model-driven approaches to the development of mobile applications (middle part of the table adopted from [UB16])

Name	**Modeling Aspects**			**Modeling Techniques**					**Application implementation**			**Application architecture**	
									Model compiler				
	Data	**Behavior**	**GUI**	**Annotation**	**Textual**	**Graphical**	**PIM**	**PSM**	**Native**	**Cross-Platform**	**Model interpreter**	**Client-Server**	**Rich-Client**
MD²	✓	✓	✓	×	✓	×	✓	×	✓	×	×	✓	×
Mobl	✓	×	✓	×	✓	×	✓	×	×	✓	×	✓	×
JUSE4Android	✓	×	✓	✓	×	×	✓	✓	✓	×	×	✓	×
Modagile	✓	×	✓	×	×	✓	✓	×	✓	×	×	×	✓
Serral et al. [Ser+10] [Muñ+06]	✓	✓	×	×	×	✓	✓	×	×	✓	×	✓	×
Ceri et al. [Cer+07]	✓	✓	×	×	×	✓	✓	×	×	✓	×	✓	×
Escolar et al. [Esc+14]	✓	✓	✓	×	✓	×	✓	×	×	✓	×	✓	✓
Kapitsaki et al. [Kap+09]	✓	✓	×	✓	×	×	✓	×	×	✓	×	✓	×
MobML [Fra+14]	✓	✓	✓	×	✓	×	✓	×	✓	×	×	-	-
MIMIC [Elo+14]	×	×	✓	×	×	✓	✓	×	✓	×	×	-	-
Applause [Beh10]	✓	✓	✓	×	✓	×	✓	×	✓	×	×	✓	✓
Francese et al. [Fra+15]	×	×	✓	×	✓	✓	✓	×	✓	×	×	-	-
MAG [Usm+14]	×	✓	✓	✓	×	×	✓	×	✓	×	×	-	-
RUMO [SF13]	×	×	✓	×	✓	✓	✓	×	✓	×	×	-	-
WL++ [Str+15]	✓	✓	✓	×	✓	✓	✓	×	✓	×	×	-	-
AXIOM [JJ14]	✓	✓	✓	×	✓	✓	✓	✓	✓	×	×	-	-
Mendix App Platform [25]	✓	✓	✓	×	×	✓	✓	×	×	×	✓	-	-
IBM Rational Rhapsody [19]	✓	✓	✓	×	✓	✓	✓	×	✓	×	×	-	-
WebRatio Mobile Platform [77]	✓	✓	✓	×	×	✓	✓	×	✓	×	×	-	-
Appian [04]	✓	✓	✓	×	×	✓	✓	×	✓	×	×	-	-
Our Framework	✓	✓	✓	×	×	✓	✓	✓	✓	×	✓	✓	✓

criterion is not relevant if a domain-specific modeling language is employed. A *textual* concrete syntax is not yet realized, but based on the used framework (EMF) such a syntax could be additionally added as future work. Similarly, the criterion of *cross-platform* code generation is currently not realized. Since the model-driven development approach is not limited to native code generation, this criterion could be fulfilled in the future by writing an additional code generator which generates cross-platform program code.

8.30 Our framework for the model-driven development of mobile applications supports most of the mentioned features. Moreover, the ability to compile and interpret models and the support of different mobile application architectures lead to the contribution made in the second part of this thesis: the *context support*.

Part II

Context Support

Chapter 9

Context Support – Foundations and Definitions

The following sections clarify the terms *context*, *context-awareness*, and *mobility* which are relevant in the second part of this thesis. Regarding the state of the art, we will shortly summarize the existing solutions and concepts with respect to platform-, device-, and user context support. We will discuss the support for system contexts in more detail. To this, we recall prior work of our own [Vau+16a] to show how we adopted the state-of-the-art extended client-server architecture and created a generic version which is used as the architecture of our generated mobile applications (cf. Figure 1.4). This includes an introduction to the essential mobile transaction models. **9.1**

9.1 Context and Context-Awareness

A significant difference between mobile applications and traditional applications is the influence and evaluation of different contexts. In this work, we define a *context* as anything a mobile application can sense (cf. Abowd et al. [Abo+99, Sec. 2.2] and Dey [Dey01]). Moving on, we can distinguish between *simple contexts* (e.g., geographical location, temperature, and brightness) and *complex contexts* (e.g., walking outside, waiting at the check-out line at the supermarket), as Hofer et al. [Hof+03][1] proposed. If a mobile application could sense and react to such contexts, we would call it context-aware, situation-aware, or self-adaptive (cf. [Abo+99, Sec. 3.2]). For instance, a context can be the time and the location of a mobile user. A context-aware application may use this information to provide the mobile user with context-specific data. Different dimensions of the context have been proposed by Abowd et al. and Dey [Abo+99, Sec. 2.3], Schilit et al. [Sch+94], and Schmidt [Sch+99]. As a summary of these, we use the following four dimensions (cf. Section 1.2) in the course of this work: platform-related contexts (e.g., operating system), user-related contexts (e.g., role, personal data), device-related contexts (e.g., display size and available sensors), and system-related contexts (e.g., online/offline, interaction with other devices, services). Figure 9.1 shows the different contexts that are relevant to this work. Contexts are *directed*. Thus, to describe that a mobile application can be installed on different mobile devices, we say, "A mobile device is a context for a mobile application." Contexts may also be *transitive* Hence, we say, "The system is a context for a mobile application" instead of "The system is a context for a mobile device, which is a context for a mobile application." **9.2**

As shown before, the mobile application is the entity that detects and reacts to the appearing contexts of different dimensions (cf. Figure 9.1). Hence, context-awareness is primarily a property of mobile applications. In other areas, as Computer-supported cooperative work (CSCW) and Human Computer Interaction (HCI), the term context-awareness has also been applied, but in a different manner. In **9.3**

[1] It was originally called the *physical context* and the *logical context* respectively.

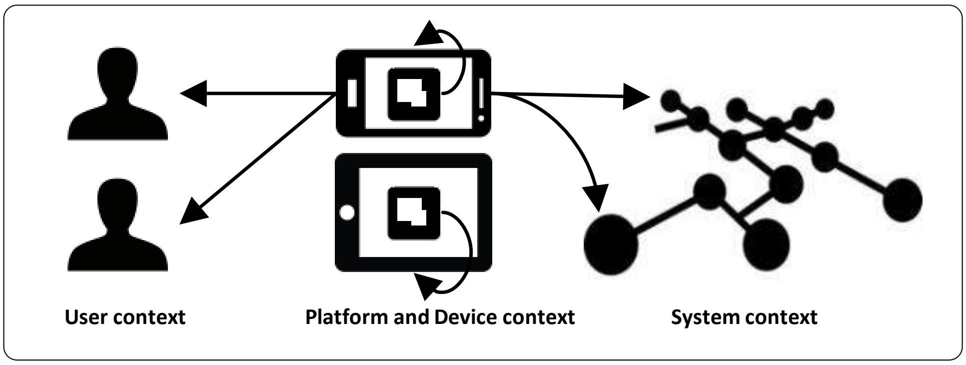


FIGURE 9.1: Considered contexts of mobile applications

this field of research, the (mobile) user and its context-awareness regarding the environment is in focus rather than the context-awareness of the mobile application.

9.4 Dourish and Bellotti [DB92] reports that during a CSCW task the users must be aware of the activities of other users. This provides a context for the own actions and will enable cooperative work. Greenberg and Marwood [GM94] show that the context-awareness of users affect the design of the interface (cf. Rodden et al. [Rod+98]) since the actions of collaborating objects and subjects must be propagated to the interface in order to enable context-aware behavior and interaction of the user. Moreover, this artificial reproduction of contexts is strongly related to virtual reality (VR) or augmented reality (AR) applications (cf. Aliaga [Ali97]). In such applications, real objects are simulated, or annotated with additional information. The users of such applications can interact with such virtual or real objects. Hence, the user behavior is driven by these contexts.

9.5 To sum up, in this work we will deal with the context and context-awareness of mobile applications, even if mobile users also may sense and react to contexts.

9.2 Mobility

9.6 Most contexts emerge and disappear due to the mobility of mobile devices. Thus, context-awareness is strongly related to *mobility*. The term *mobility* is often used in a misleading way, and is misconceived in the literature on mobile application engineering. In normal language usage, *mobility* is the capability of moving or being moved[2]. The term *moving* is broadly understood as spatial movement between geographical locations. But, we relax the definition and also define the change of logical contexts, such as different devices and network nodes, as a change of place. Considering that an object (e.g., a mobile device) or a subject (e.g., a mobile user) provides or consumes (i.e., uses) different functions respectively, we extend the definition and want the providing or the consuming of the functions to continue while the objects and subjects move spatially. An object or subject is *mobile* if it can operate while it is moving. Otherwise, we would call the object and the subject *portable*. This is the case if they can operate in different places, but not while moving. Objects and subjects can be *portable* and *mobile* at the same time. That is, they operate on a subset of features (called mobile features) while other features (called portable features) are only available once they reach a certain position (e.g., network coverage). Hence, mobility is a fine-grained property.

9.7 Another aspect is the object or subject whose mobility is considered. In general, the expression *Mobile x* or *x mobility* describes the involved object or subject x. This is often called the *dimension* of mobility. B'Far [B'f04] and Asoke et al. [Tal10, pp. 6, 7] defined several fine-grained dimensions of mobility. We take up and adopt the

[2] The definition was taken from the Merriam-Webster Dictionary.

three coarse-grained dimensions provided by Pandya [Pan04, Chap. 1]: (i) *terminal mobility*, (ii) *service portability*, and (iii) *personal mobility*.

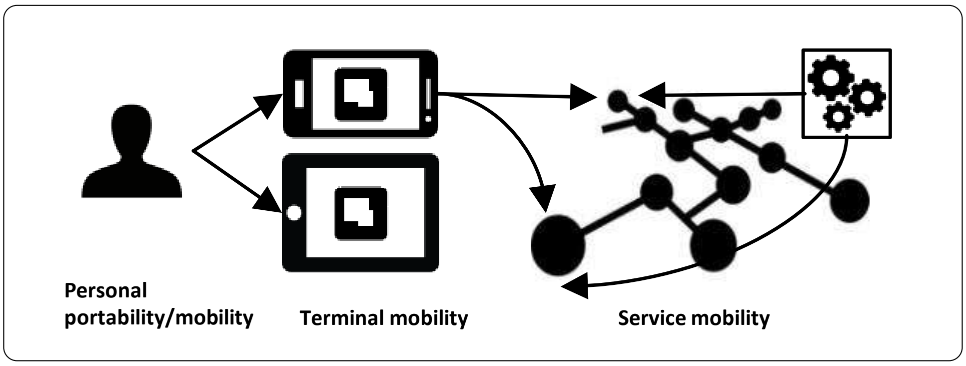


FIGURE 9.2: Mobility of different subjects and objects

Figure 9.2 shows these dimensions. The service on the right-hand side may be invoked through different access points of a network and by different mobile devices. This ability is called *service mobility*. The service runs on a server. Thus, the term *ubiquitous* is often used in the literature because the service can be invoked everywhere. Service mobility can be ignored in this work because we focus on native application implementations rather than on web-based mobile applications. **9.8**

A moving terminal (mobile device and mobile application) may connect to different access points (e.g., Wi-Fi, mobile networks). However, it can also be disconnected. A *terminal* (e.g., a mobile device) is mobile if it provides all the functions while moving. This also applies to the applications that are installed on this terminal. This *terminal mobility* is what is generally intended when we talk about mobility and mobile applications. **9.9**

The user may own and use one or more devices. Since s/he cannot access a service or mobile application while changing devices, we call this *personal portability*[3], in line with the given definition. **9.10**

The mobility of objects and subjects is *transitive*. Thus, if the user utilizes a terminal that is mobile, the user is mobile as well. **9.11**

To sum up, mobile applications that can operate reliably during movement, especially while passing through different contexts, may be called *mobile*. Otherwise, they are *portable*. Hence, in a narrow sense, mobility can only be guaranteed by context-aware applications. **9.12**

9.3 Platform-, Device-, and User Context Support

The following sub-sections shortly summarize the state of the art with respect to the different contextual dimensions respectively: **9.13**

9.3.1 Platform Context Support

The approaches shown in Section 8.1 support the platform context during the development of mobile applications. Either a model-driven development approach or a non-MDD cross-platform approach ensures the support for different platforms. Since a model-driven development approach seems more powerful to support different platforms by the generation of native mobile applications on one hand, and **9.14**

[3] Pandya [Pan04, Chap. 1] named this personal mobility

overcome the limitations e.g., high development cost through manual implementations and weak user coverage (cf. Dalmasso et al. [Dal+13, Table 1]) on the other hand, we will point to the approaches and frameworks listed in Table 8.1 as state of the art to deal with different platform contexts. Hence, platform context support is an inherent feature of the selected development approach, e.g., model-driven development, and thus we will not discuss this contextual dimension further in the second part of this thesis.

9.3.2 User Context Support

9.15 Since the state of the art in user context support is largely focused on the context of use (e.g., the physical and logical environment of a mobile application user) and rather on different user groups and user roles, we are not able to present any state-of-the-art concepts for user context support for mobile applications. However, Usman et al. [Usm+14] exploit use case diagrams for the generation of mobile applications. This framework might potentially be able to generate different user-specific variants of mobile applications based on modified use case diagrams. Despite this, a runtime adaptation of the mobile application depending on the current mobile user might not be possible.

9.3.3 Device Context Support

9.16 The support of the device context is more complex, because in contrast to the platform context, the device context can usually be evaluated at the runtime of the mobile application at the earliest. A certain device context affects at most the graphical user interface of a mobile application. However the availability of hardware sensors may affect the functional part of the mobile application. Since the adaptation of the functional part depending on the hardware configuration of a device is not handled very well by the state-of-the-art concepts, the adaptation of graphical user interfaces has been an active area of research for many years. A lot of work concerning dynamic graphical user interfaces is carried out by Calvary, Vanderdonk, Souchon, Limbourg and others ([Cal+03], [Lim+04], [Eis+01], [Van05], [Cal+02], [SV03], [FV04]). According to the unified reference framework given by Calvary et al. [Cal+03], a graphical user interface must be context-aware with respect to the users, the hardware and software platform and the physical environment. This context-awareness requires design time adaptation (originally named *predictive context*) and runtime adaptation (originally named *effective context*) of the graphical user interface. A common state-of-the-art practice to deal with the adaptation of graphical user interfaces is the application of domain-specific description languages and corresponding instance models which describe graphical user interfaces in an abstract way. Such descriptions can be transformed according to the different context of use (e.g., device contexts). More state-of-the-art concepts can be found in the case studies carried out by Calvary et al. [Cal+03, Chapter 3].

9.4 System Context Support

9.17 Research with a focus on system context support largely deals with the connection context of mobile devices, since the connection context affects the operability of a mobile device considerably. Hence, the primary goal is an architecture that provides an online- and offline-capable data and transaction management for mobile applications.

9.18 Therefore, the question arises how architectures can support this requirement [Rom+00] [Lem+13]. Mobile application developers can choose from a variety of technologies (e.g., native, hybrid, and web-based) and different architectures (rich-client and client-server) to develop mobile applications. Sometimes, they are unaware of the impact a chosen technology and architecture have on the mobility of a mobile application. Web-based architectures often require a permanent connection to the server, and therefore, are not suited to realize applications operating in an offline mode. In contrast, native and interpretive technologies enable architectures that may be used for working temporarily without a network connection.

Changing network conditions and transaction-oriented applications can cause problems with respect to data and transactions which have to be managed in an appropriate way: (1) a mobile application being part of a multi-user transaction system has to replicate and synchronize data in order to process transactions offline. (2) When being disconnected, the coordination of concurrent transactions is disrupted since the mobile application is unaware of transactions performed by other users in the network (no matter whether they are still online or offline). Conflicts may arise when modified replicated data are synchronized with the server which implies that offline transactions can be finished at synchronization time at earliest. **9.19**

In order to deal with the problems mentioned before, several authors (such as Satyanarayanan [Sat96], Pitoura & Samaras [PS12], and Book et al. [Boo+05]) propose a new architectural design. They propose a change of paradigms from the client-server or rich-client design (cf. Figure 7.3) to a mixture of both architectures. Hence, mobile applications can operate offline (using architectural components of the rich-client design), or online (using architectural components of the client-server design). We dub this architectural design an *extended client-server architecture*. The extended client-server architecture at first ensures the availability of data even if the client is disconnected. However, transactions that are performed offline must be synchronized later when the mobile clients reconnect with the server. Sometimes it is required that this synchronization does not fail, i.e., any transaction performed offline must be synchronizable with the server. This requires mobile transaction models. Mobile transaction models are aimed at maximizing concurrency and maintaining data consistency in a failure prone and low bandwidth mobile environment [Hel+96]. To this end, different mobile transaction models were proposed in the literature (e.g., Keypool transaction model [54]; [55] and Escrow transaction model [O'N86]; [LL09]). These transaction models guarantee conflict-free synchronization. **9.20**

9.4.1 Application Domain and Example Applications

To illustrate the application of online- and offline-capable mobile applications, we present example domains where such mobile applications can be used profitably, followed by two application scenarios. **9.21**

The three biggest retailers [50] in Germany in 2014, Amazon [02], Otto [40], and Zalando [78] provide mobile applications for different platforms. None of them supports transactions, such as viewing and ordering products, and mobile payment, in a disconnected mode. Using a replicated digital product catalog, users could view products while they are offline. They could also order offline and later send the order to the back end when the connection is re-established. An inherent problem within this setting is the limitation of mobile transaction methods for offline payments [SK12]. Since an order often requires payment in advance, secure payment is a crucial component of most e-commerce applications (e.g., 80% of the popular online shops provide PayPal [42] as a payment service [50]). Payment transactions usually contain an online clearing an confirmation step not allowing these transactions to be performed offline. Thus, mobile e-commerce applications for offline usage also require offline payment methods. **9.22**

In industrial settings, the problems are slightly different: mobile applications often support or substitute manual tasks (as, e.g., inventory, order picking, and maintenance logging). The involved data objects are more individual and tailored to the **9.23** surrounding business processes or real-world objects. Likewise, operations on these data objects are often more complex than in e-commerce and payment scenarios.

9.24 To sum up, online- and offline-capable architectures, and in particular, mobile transaction models are required whenever mobile applications modify either aggregated values (i.e., account balance and warehouse stock) with a set of repeatable operations (e.g., increment and decrement) or more customized objects (as they occur in e.g., a car rental) with a set of complex operations (e.g., pick-up, refuel, and event of damage). An object containing only an aggregate value attribute is named *summable object*, otherwise, it is called *individual object*. If the set of operations contains at least one altering operation, there is a potential conflict, i.e., the data objects may be accessed in a competitive manner.

9.25 In the remainder of this chapter, we will take a closer look at two specific applications – a payment application (covering an example for an aggregated value with repeatable operations) and a course booking application for gym members (dealing with individual objects and more complex operations) – to demonstrate the different ways of data access. We start the discussion of each application with the assumption that there is no replication for offline usage. This corresponds to a client-server architecture. Later in Section 9.4.4, we show implications of adding data replication.

9.4.1.1 Payment App

9.26 First, we consider applications for making mobile payments, such as Apple Pay [07], Google Wallet [14], or PayPal [42]. Traditionally, a banking account is administered on a server at the banking site. The dataset mainly consists a single value (aggregate), the account balance. This singleton is a so-called hot spot because every transaction changes or accesses this value. The set of transactions on a banking account is very limited (i.e.,withdraw cash, deposit cash, debit, credit, and getting account balance). For each banking account, the number of users is also quite limited. Users of a banking account are the bank itself and at least one bank client. The bank itself arranges debit and credit payments from or to internal or external accounts. Banking clients may perform cash withdrawals or deposits. We assume that the account is a credit account, i.e., that it should never be in the debit state (which would give raise to a conflict situation).

9.27 **Example** (Money transfer). We consider the following use case: a banking client wants to transfer money to another banking client from her/his mobile phone using near field communication (NFC). Most payment applications support this use case if both banking clients are online. The transaction is carried out as follows: the creditor delivers his/her account information to the debtor via NFC. The debtor sends a corresponding payment order to the back-end server as online transaction. The bank checks the cover of the payment order and executes the transaction. Finally, the creditor updates the account balance (again as online transaction) and confirms the transaction. If one mobile client is offline, the payment cannot be conducted. □

9.4.1.2 Course Booking App

9.28 As second example, we consider a course booking application for gym members. Examples of such applications are GymSync [16], GymJam [15] and BookFit [10]. Registered gym members can select course spots to practice particular exercises (e.g., Yoga and Pilates) within certain time slots. The set of operations is very limited (e.g., making or canceling a reservation or checking if a course spot is available). Conflicts are very likely because each member may select every course spot. Studio members may set course preferences to indicate which courses they will select in the future. We assume that the course booking application never allows the overbooking of a course spot (conflict situation).

Example (Reservation of a course). If a gym member is online and a selected course spot is available, a reservation transaction can be processed. If the mobile client is offline, no transaction can be conducted. □ **9.29**

9.4.2 Online- and Offline-Capable Architecture Design

In the following section, we present recommendations to the design of an online- and offline-capable architecture and its working model including the required components. These recommendations form a kind of reference architecture. Several conceptual implementations of this reference architecture use individual mobile transaction models. Although there are plenty of different mobile transaction models, the number of available products is still limited [Ber+04]. We discuss the applicability of these products in accordance with the requirements stated above and identify shortcomings. **9.30**

The loss of connection caused by the *terminal mobility* [Pan04] is not unusual [Fuc09] and should be handled by the architecture of mobile applications. In case of intentional or accidental loss of connection, it is necessary to delegate the server-located functions to the mobile device to be able to work in an offline mode. This is sometimes called *blurring of roles*. Satyanarayanan [Sat96] describes the resulting architecture as an *extended client server model* where the extended client takes over the role of the unreachable server. Pitoura and Samaras propose a similar architecture [PS12]. Book et al. [Boo+05] state that the mobility of applications is influenced by their architecture. In reverse, this means that mobile applications have to follow a particular architecture to be online- and offline-capable. **9.31**

Figure 9.3 shows the *extended client server model* which we later use as a blueprint of our generic architecture. The *application logic* of the client is extended by a local *transaction manager (TM)* which delegates all transactions either to the local database management system (DBMS) or to the centralized one on the server, dependent on the mobile application's connection state. Furthermore, the *transaction manager* comprises a *replica manager*. The *replica manager* is responsible for the replication of data to be used while mobile clients are offline. Offline processed transactions are logged by the *synchronization/reintegration manager*. Later, this *log* is used to reprocess the transactions performed offline. **9.32**

A *transaction manager* implements a particular mobile transaction model by specifying the behavior of replication, offline operations, and synchronization. These components (shown as gray colored areas in Figure 9.3) are individually tailored according to the properties of the used mobile transaction model. For example, they may be asked to be conflict-free. Online transactions are just passed to the server and not handled by the local transaction manager. In Section 9.4.2.4, we will give an overview of the different mobile transaction models that have been proposed in the literature. Actually, every proposed mobile transaction model results in an individual implementation of the *extended client server model*. To support a more flexible exchange of transaction models, we are heading towards a generic architecture. **9.33**

9.4.2.1 Working Model of the Local Transaction Manager

In Figure 9.4, a working model is shown which distinguishes the different context states and operation steps of the local transaction manager. A mobile client starts in an online context (*online transaction processing*) after an optional *initial setup* of the system. In this mode, an online transaction model (often named standard transaction model) is used. Mobile clients replicate the data while they are online (*Replication*). The client may stay online after the replication or may go offline. If so, **9.34**

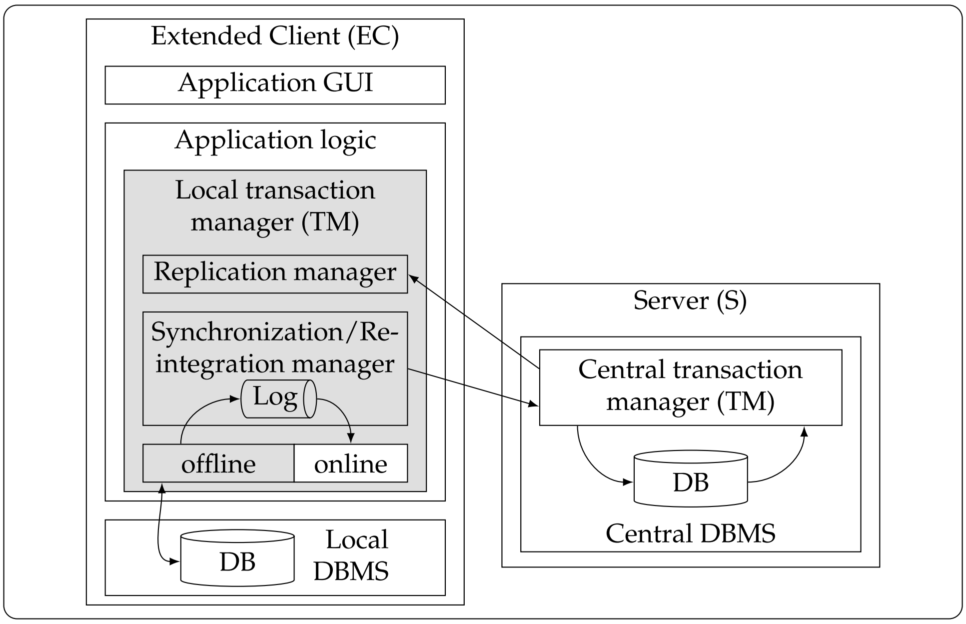


FIGURE 9.3: Extended client-server architecture

it operates offline (*offline transaction processing*) using a particular *mobile transaction model*. When the mobile client is back online, it must publish the modified copies (*Synchronization/Reintegration*). Gray colored areas in Figure 9.4 denote steps that are specific to the used mobile transaction model.

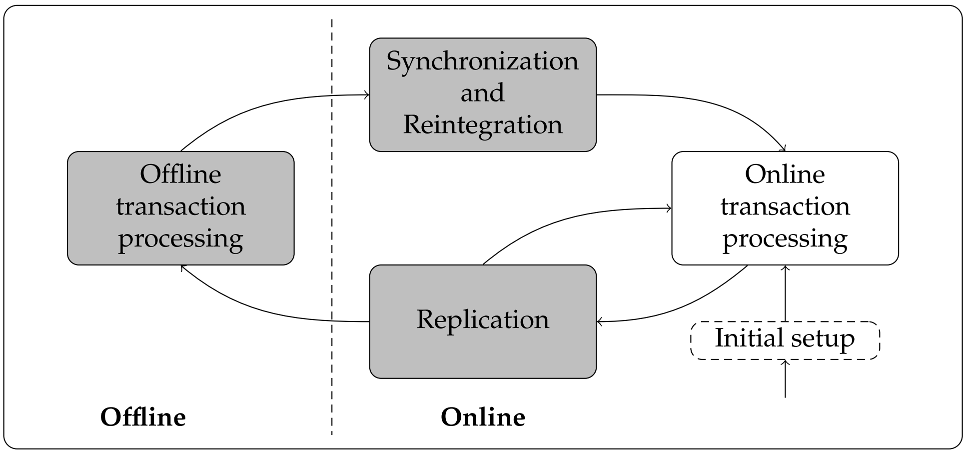


FIGURE 9.4: Working model of a local transaction manager (extended client-server model)

9.4.2.2 Anomalies

9.35 While working offline, concurrent isolated accesses to the clients' replicas may lead to the following kinds of conflicts when synchronizing the modified data. These conflicts are called anomalies:

9.36 *Deletion anomaly:* If a mobile client deletes a replicated record while working offline and another client reads or changes the primary copy of this record meanwhile, a deletion anomaly occurs.

9.37 *Insertion anomaly:* An insertion anomaly occurs when a new record is inserted into the servers' database and an identical record does already exist. This anomaly can occur if two mobile clients independently create a record and at least one mobile client is offline.

Modification anomaly: The modification anomaly is the most common one. Every time a mobile client changes a replicated record while working offline, another client might also have changed it. Such a conflict may be solved by synchronizing one replica and discarding the other one. The question arises which modification is prior over the other. **9.38**

Trivially, concurrent isolated read-only access of all mobile clients cannot lead to anomalies. Thus, if mobile applications use static data (e.g., encyclopedias and dictionaries) in a unidirectional way, mobility can easily be guaranteed by just replicating the used data. However, the amount of data to be replicated may be a limiting factor. **9.39**

9.4.2.3 Replication and Synchronization

Within the working model, a step for replication is needed to guarantee availability of data when being disconnected [Gol03a] [Gol03b]. Moreover, we are heading towards applications that may change and then reintegrate the replicated data into the server. Thus, the working model also requires a step for synchronizing and reintegrating modified data after working offline. Both detailed mechanisms of replication and synchronization heavily depend on the mobile transaction model used. In Section 9.4.4, we will present a generic replication strategy which is suitable for several mobile transaction models. In any way, replication strategies can be classified into *eager* and *lazy* [Gra+96] ones. Eager replication strategies try to update all copies in a single step to complete the transaction. This is inappropriate within our application area of mobile applications, since mobile clients cannot be updated while being disconnected. Lazy replication asynchronously propagates replica updates to other clients after offline transaction commitments. For doing so, the transaction is executed locally and then reprocessed later on a primary copy and other replicas. This is called *transaction-based* synchronization. Sometimes, it is sufficient to replace the primary copy with the changed replica. This kind of synchronization is called *image-based* synchronization. The synchronization approaches of concurrent accesses can be classified – similar to traditional database management systems [MN82] – into *pessimistic* and *optimistic* approaches. Pessimistic approaches include strategies for *conflict avoidance* while optimistic approaches provide at least *conflict detection* and often also *conflict-solving* strategies. **9.40**

9.4.2.4 Mobile Transaction Models

In the following section, we recall mobile transaction models that support lazy replication and pessimistic synchronization since they prevent the occurrence of anomalies. **9.41**

In order to find mobile transaction models with these characteristics, we recall the following work: Hirsch et al. [Hir+01] survey several mobile transaction models and compare them on the basis of typical requirements for this application domain ([TG95], [Dun+97], [LS97]). Serrano et al. [Ser+01] [SA+04] and Panda et al. [Pan+11] analyze the existing approaches in accordance to the well-known ACID (Atomicity, Consistency, Isolation, Durability) paradigm in a similar way. Mutschler and Specht [MS13] divide the mobile transaction models either into *first-class transaction models* (which processes transactions offline but need to be online to commit the transaction) or *second-class transaction models* (which processes transactions offline). **9.42**

Based on these reviews, we discard all approaches that are not able to prevent **9.43** conflicts and to work offline like the *Kangaroo transaction model* [Dun+97], the *Preserialization Transaction Management Technique (PSTMT)* [DG00], the *Prewrite Transaction model* [MB01], the *Two-tier transaction model* [Gra+96], the *Clustering transaction model* [PB95] [PB99] [Pit96], the *Reporting and co-transactional model* [Chr93], and the *Isolation-only transaction model* [LS94].

9.44 The remaining conflict-free transaction models can be subsumed under *semantical* approaches. Semantical approaches use the structure of the data or semantical properties of transactions performed on replicas [GM83]. We have selected the *Keypool transaction model* and the *Escrow transaction model* for use in our generic architecture.

9.45 The number of products is still limited, outdated, and very homogeneous in terms of used mobile transaction models. OracleLite [38], IBM DB2 Everyplace [18], Microsoft SQL Server CE [28], and Sybase Adaptive Server Anywhere [56] are some commercial mobile database systems (mDBMS). In general, their architecture corresponds to the *extended client server model*. All products use an *image-based* synchronization and do not support conflict prevention. Thus, durable offline transactions cannot be carried out locally.

9.46 With regard to the available products, we assume that the replication is possible within our scenarios but does not yet include a conflict-avoiding mechanism while processing transactions offline.

9.47 **Example** (Payment App). A debit transaction decreases the replicated *account* value of the debtor and increases the replicated *account* value of the creditor. The application checks the coverage of the replicated *account* value locally. The transaction can happen offline via NFC. At a later date, the banking client reprocesses the debit or credit transaction on the primary copy in order to synchronize the account balance (online transaction). However, if the debtor withdraws money and changes the primary copy before executing the synchronization, the coverage of the account cannot be ensured. The bank is unaware that the customer has already transferred money from the replicated *account*. Since the account may be in debit state, a conflict may arise. □

9.48 **Example** (Course booking App). The gym member uses a copy of the entire data set, i.e., of all course spots. A reservation transaction checks whether a course spot is unselected by other members and selects it. At a later date, the gym member(s) synchronize the changed course spots with the primary copies. If another gym member selected the same course spot, the transaction of one member gets lost during synchronization. Nevertheless, both members get a local commit of their transaction. Since a course spot may be overbooked, a conflict may arise. □

9.4.3 Problem Statement

9.49 Although conflict-preventing mobile transaction models exists, the available products do not use them. As stated by Gollmick [Gol06], barriers are the demarcation of the mobile database management systems (mDBMS) and the semantics of transactions located at the mobile client or at the server. Either the mobile application realizes a mobile transaction concept on the level of application logic, or the mobile database management system supports a seamless interface to use the semantical information of transactions being defined by application logic. With the focus on mobile development, the following question arises: how does a generic architecture that allows different mobile transaction models for online- and offline-capable mobile applications look like?

9.50 The existing work of mobile transactions models focuses on relational data models. Following object-oriented design, data models of mobile applications are object-oriented (i.e., class models). Thus, the existing concepts must be rethought and adapted to the context of object-oriented data modeling. Mobile application developers are often familiar with the object-relational mapping (ORM) to serialize objects into relations but unsettled in applying this concept in the context of mobile transactions involving replication and synchronization. Therefore, the next question is: can mobile transaction models be applied in the context of object-oriented application development and what are the effects?

Finally, the existing mobile transaction models have not been evaluated well. From the perspective of a mobile application developer, the conditions (e.g., connectivity, number of users, and data) under which mobile transaction models should be used are unclear. Mobile transaction models may bring profit to disconnected clients but may also cause additional costs (w.r.t. replication and synchronization). They may cause reduced performance for highly connected users (clients). These considerations lead us to the third question: which kinds of context conditions are assumed for a mobile application to profit from using mobile transaction models? **9.51**

9.4.4 Generic Online- and Offline-Capable Architecture Design

Based on the *extended client server model* and the working model presented in Section 9.4.2, we present a generic architecture for online- and offline-capable applications that can be instantiated with different transaction models (see Figure 9.5). In this section, we focus on the instantiation with conflict-free mobile transaction models, namely Keypool and Escrow. They seem to be especially promising for online- and offline-capable transaction processing. Knowing the differences between these mobile transaction models in terms of their individual replication and synchronization, we can modify the working model in order to use both mobile transaction models in a single generic architecture. Finally, we present the developed design along the steps of a modified working model. **9.52**

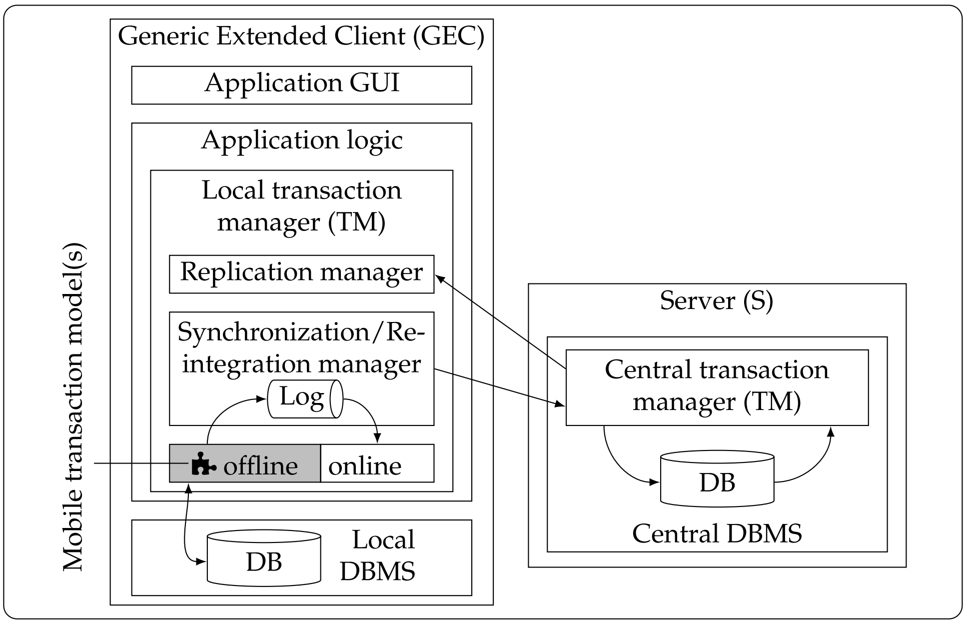


FIGURE 9.5: Generic extended client-server architecture

9.4.4.1 Conflict-Free Mobile Transaction Models

We focus on conflict-free mobile transaction models. The selected models use different strategies to prevent conflicts: **9.53**

Keypool Transaction Model

9.54 The *Keypool* transaction model [54]; [55] uses the structure of the given dataset. The basic idea of the Keypool method is to split the entire dataset into subsets that are distributed among the participating mobile clients. Figure 9.6 illustrates a data split to three mobile clients within the replication step. Every client gets an amount of data that is exclusively replicated. When a client is offline, it can operate on the replicated data without limitations. Within the synchronization step, the partial data is reintegrated into the primary copy using *image-based* synchronization. Independent of the operation to be performed while being offline, the result can be adopted by substituting the value of the primary copy for the value of the changed replica (i.e., the *image*). The Keypool approach avoids *deletion* and *modification* anomalies by design. Without additional provision, *insertion* anomalies may occur. During the course of this work, we ensured that insertion anomalies cannot occur by the use of an object-relational mapping framework.

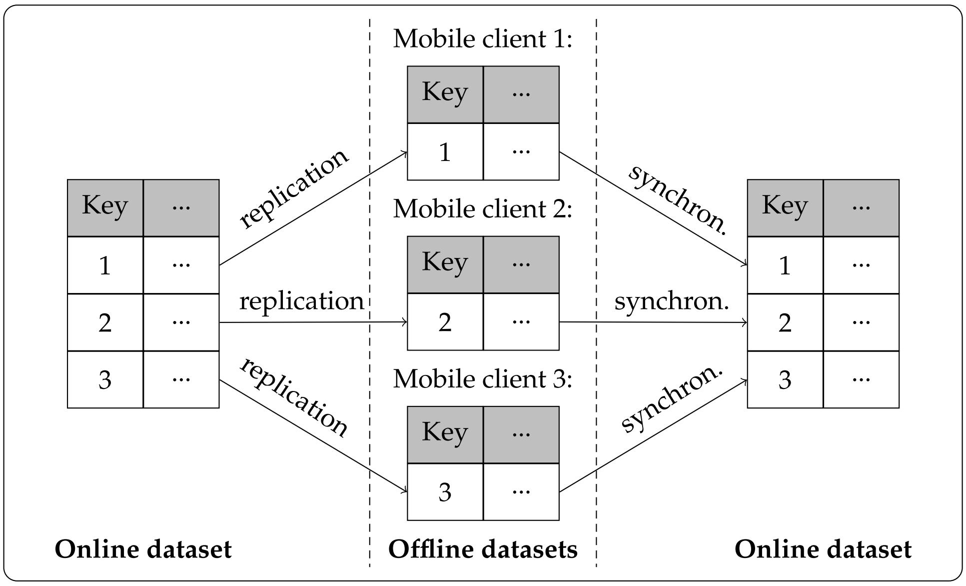


FIGURE 9.6: Keypool replication and synchronization

Escrow Transaction Model

9.55 The *Escrow* transaction model [O'N86]; [LL09] is well suited to access and modify aggregate data. The basic idea of the Escrow method is to restrict the set of transactions and/or the domains of their arguments when being performed offline. While the Keypool transaction model splits the dataset, and thereby, may risk to provide an undersized or empty dataset, the Escrow approach always provides the full dataset. Figure 9.7 shows the replication scheme of the data to two mobile clients. Every client gets a full copy of the dataset. Assuming that the semantics of provided transactions are known, every record is transformed at the step of replication such that conflicts cannot occur at the synchronization step. Considering mobile payment, for example, the debit transaction may cause conflicts. Therefore, the domain of its argument is restricted such that just small amounts may be withdrawn. One possible strategy is to equally distribute the amount among all participating clients as shown for the example aggregate values in Figure 9.7. Since several mobile clients may change the same value, this strategy always guarantees conflict-free synchronization afterward.

9.56 An *image-based* integration does not work here since either one or another image can be written back to the primary copy but not both. The other values would be lost (called Lost-update [Ber+95]; [Ady+00]). Thus, the reintegration of changed values has to be based on a *transaction-based* approach. It collects all transactions performed offline and replays them on the primary copy. The repeated transactions must have the same effects as being performed online but usually do not achieve the same value on the primary copy as on the replicated copies. This property is called *semantical serializability* [Ouz+09]. To ensure it, all operations must be repeatable (such as decrement and increment) and their semantics on restricted values has to correspond to the one on non-restricted ones. The Escrow approach avoids *insertion*, *deletion*, and *modification* anomalies by design. A generalization beyond aggregated values is the PRO-MOTION transaction model [WC97]. Within that approach, so-called *compacts* form local constraints and guarantee semantical serializability.

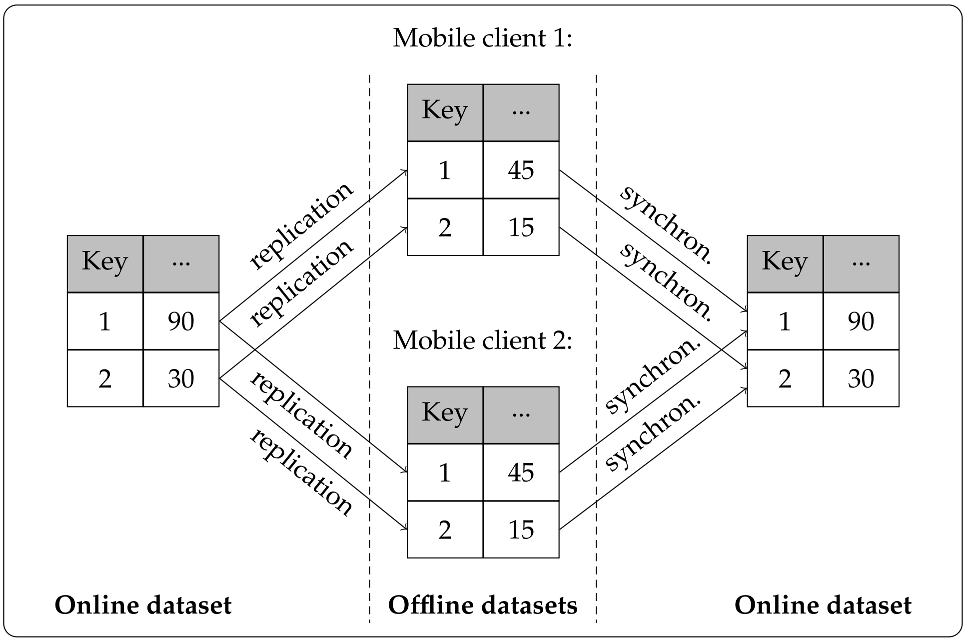


FIGURE 9.7: Escrow replication and synchronization

9.4.4.2 Modification of the Working Model

As presented in Section 9.4.2.1, replication and synchronization are the major steps of the working model. The mobile transaction models Keypool and Escrow use their own approaches to replicate and synchronize data. Keypool replicates data sets by splitting while keeping their values unchanged; Escrow, however, does not split the dataset but replicates it by changing every value of the dataset in accordance with the number of mobile clients. Synchronization is performed *image-based* by the Keypool method and *transaction-based* by the Escrow method. Hence, the replication and synchronization steps of these mobile transaction models cannot be mixed for providing more than one mobile transaction model in a single architecture. Each mobile transaction model needs an individual implementation. **9.57**

In order to circumvent this problem, we modify the working model by adding the following conditions: (1) The replication step is not allowed to limit the dataset or to transform its values. If a mobile transaction model requires a limited or modified set of data, the dataset must be preprocessed accordingly within the offline transaction processing step. (2) If a synchronization method is more powerful than another one (i.e., *transaction-based* covers *image-based* synchronization), the weaker method can be substituted by the stronger one. A set of mobile transaction models that satisfies these two conditions can be applied in our architecture, like the Keypool and Escrow **9.58** method. Furthermore, most of the conflicting mobile transaction models satisfy these conditions.

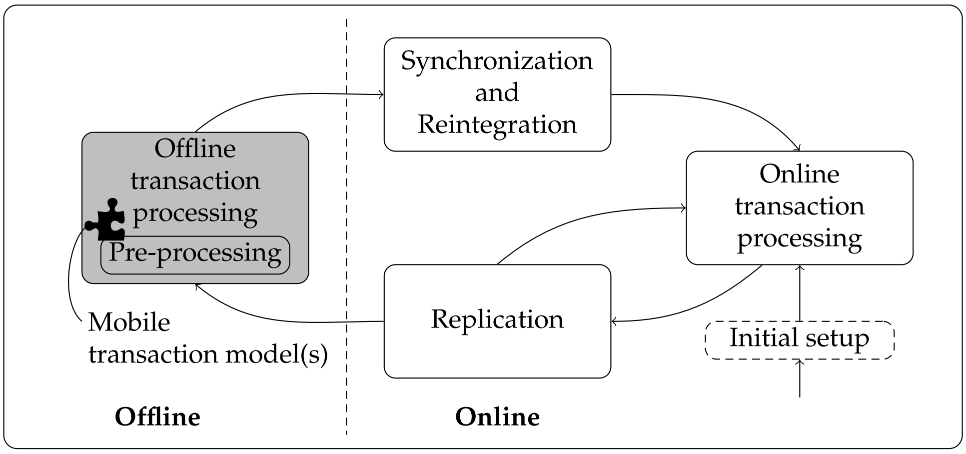


FIGURE 9.8: Working model of a local transaction manager (Generic extended client-server model)

9.59 Figure 9.8 shows the working model of local transaction managers being used in our architecture. *Replication*, *Synchronization and Reintegration* are independent generic steps, while the *offline transaction processing* implements the mobile transaction model including some pre-processing (gray colored area). Different mobile transaction models like Keypool and Escrow may be plugged in and work independently to the steps performed online.

Data Modeling and Initial Setup

9.60 Mobile applications are usually designed in an object-oriented way. Hence, the data model does not define relations, but uses object classes instead. If a database is used underneath, a class model can be translated into a relational data model by object-relational mappers (ORM). ORM frameworks can create empty database schemas from class models, and convert objects into table rows. Vice versa, database records may be translated back to object structures. In our presented data models, classes and attributes may be annotated by an asterisk ($*$) to indicate that objects of a certain class should be split (Keypool) or that an attribute is an aggregate (Escrow).

9.61 According to Figure 9.5, the server-side architecture is lightweight (i.e., no server facilities to set up the database), but it needs at least a database management system or a similar service to persist data. In our prototype implementation (described below), we use a relational database (MySQL 5.6) at the server side. The initial setup is triggered and performed remotely by a mobile client. It can only be performed once by the first appearing mobile client (cf. Section 7.3.3.1).

9.62 **Example** (Payment App). Figure 9.9 shows the data model of the payment app. It consists of the class *Account* only. A valid instance of this data model may have only one *Account* object; the asterisk at the attribute *amount* indicates that this object may be split and allocated to mobile clients. Thus, mobile clients may share this account object, particularly the attribute *amount*. □

9.63 **Example** (Course booking App). Figure 9.10 shows the data model of the course booking app. It contains course spots and persons. Course spots are not summable since each course spot is an individual object. This kind of modeling allows using the Keypool approach. However, it may lead to a large set of objects being difficult to handle. A reservation of a course spot is made by setting the participant pointer to a person. The asterisk at class *CourseSpot* denotes that its objects may be distributed among mobile clients. □

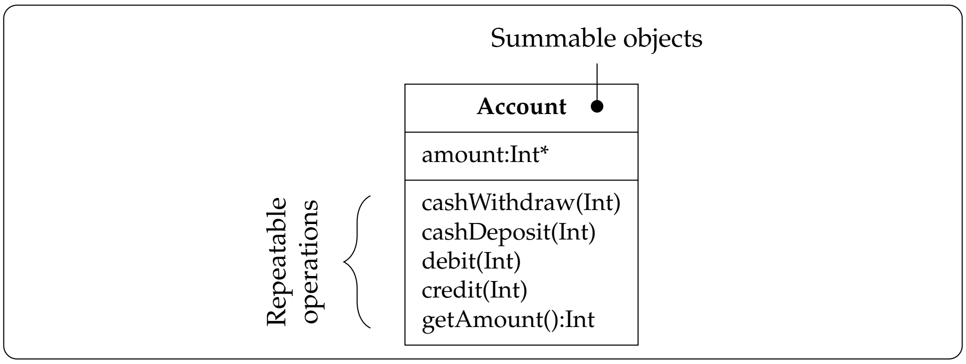


FIGURE 9.9: Data model of the payment app

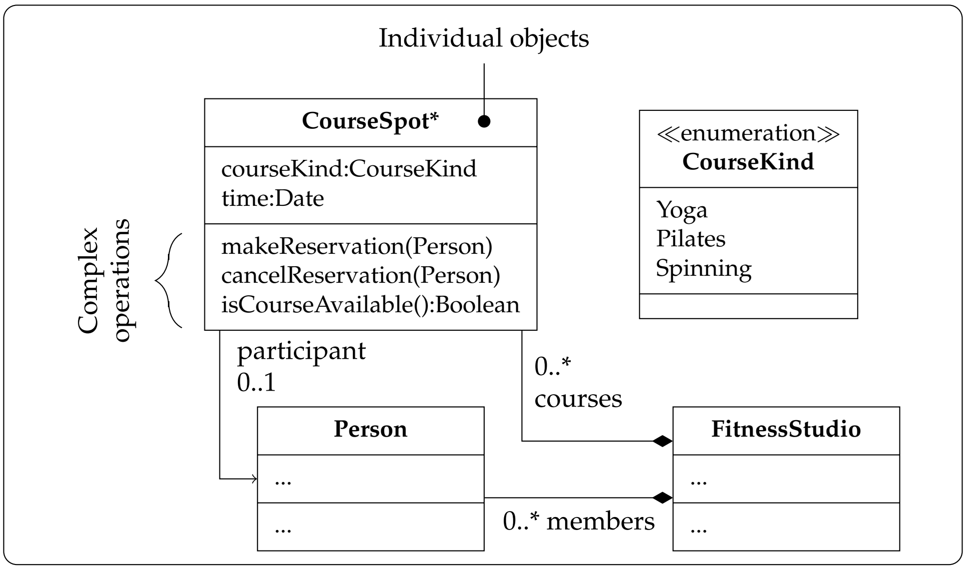


FIGURE 9.10: Data model of the course booking app

Online Transaction Processing

The online transaction processing step of our generic architecture is not affected by the changes made to the working model. It operates as stated in Section 9.4.2.1. **9.64**

Replication

Due to working model modifications, the replication step of our generic architecture copies the entire set of data to the mobile clients (*full replication*). The replication step is – similar to the initial setup of the server-located database – triggered by the clients. This is called *pull-based replication* in opposite to the *push-based replication* [Ita+05]. **9.65**

Offline Transaction Processing

The offline transaction processing is the crucial part of our generic architecture. Replication is required to operate offline but also involves the risk of synchronization conflicts. The selected mobile transaction models should ensure conflict-free synchronization and commit transactions immediate without having to wait for reconnection. **9.66**

Synchronization conflicts may occur every time when a schedule (i.e., a sequence of offline transactions on replicas) cannot be *serialized* on the primary copy. Such a **9.67** schedule is *serializable* if it is equal to a serial schedule on the primary copy. The most-commonly used definitions of this equivalence concern the order of reading and writing operations (conflict equivalence) or the view relation (view equivalence) on relational variables. This is hardly applicable to replicated isolated objects because mobile clients cannot communicate while they execute the transaction and therefore they cannot detect conflicts with other isolated mobile clients [Ber+87].

9.68 A conflict is given if at least two transactions t_1 and t_2 are performed *successfully* offline and cannot be re-processed later on the primary copy in an arbitrarily chosen order (no state commutativity) or the transactions return values dependent on the re-processing order (no return value commutativity [WV02]).

9.69 Requiring state commutativity and return value commutativity implies strict read and write operations. Dependent on the mobile application, it might be appropriate to relax the conflict definition and tolerate weak read but strict write operations [PB95]. For the course booking app, for example, the operation *isCourseAvailable* may use inconsistent copies, but the operation *makeReservation* is allowed only on consistent copies.

9.70 The mobile clients only know how many concurrent mobile clients are registered at the time of replication and which operations they may perform. Based on that fact, a conflict matrix can be set up in advance. Tables 9.1 and 9.2 show which conflicts may occur between the transactions involved in our example apps. We consider only transactions finished successfully.

TABLE 9.1: Conflicts in the payment application[1]

t_1/t_2	`debit`	`credit`	`getAmount`
`debit`	Yes	No	Yes (No)[2]
`credit`	No	No	Yes (No)[2]
`getAmount`	Yes (No)[2]	Yes (No)[2]	No

TABLE 9.2: Conflicts in the course booking application

t_1/t_2	`makeReservation`	`cancelReservation`	`isCourseAvailable`
`makeReservation`	Yes	Yes	Yes (No)[2]
`cancelReservation`	Yes	Yes	Yes (No)[2]
`isCourseAvailable`	Yes (No)[2]	Yes (No)[2]	No

9.71 Using the Escrow method, the *aggregate* that is processed by conflicting offline transactions is divided among the set of mobile clients beforehand. Besides a full allocation of the *aggregate* to the mobile clients, it can also be limited by a factor so that a part of the aggregate may remain unallocated. In case of the payment app, the amount is distributed among the set of mobile clients. All clients can access the summable *account* object. However, the conflicting offline transactions accept just a limited domain of argument values (e.g., a limited debit amount) compared to the online processing.

9.72 The replication strategy for the Keypool method filters the whole set of objects. The filter function maps objects uniquely to mobile clients but can also keep objects unallocated. *Round-Robin* is an example filter function that assigns data objects to one mobile client after the other and starts again at the first client as long as objects are available. To allocate preferred objects, mobile clients may indicate their intention with so-called *preference sets*. In that case, the filter function maps objects according to the users' preferences if possible. However, filtered-out objects cannot be accessed in a disconnected state, and consequently, cannot be involved in synchronization conflicts. For the course booking app, every course spot is mapped

[1] The transactions cash debit and cash credit are deactivated for the mobile client.
[2] Relaxed conflict definition (tolerate weak read).

to one mobile client at most. Every mobile client can work on a disjoint subset of objects. All the transactions processed offline are logged for both mobile transaction models to allow the subsequent synchronization after re-connection (as explained below).

Example (Money transfer). In this scenario, a couple shares a banking account. Before accessing the account object offline, the amount value (aggregate) will be divided into equal amounts. For example, an amount of 100$ will be divided into 50$ located at one person (client 1) and 50$ at the other person (client 2). Both can transfer 50$ offline. In doing so, the account is always on the credit side. The downside of this strategy is that they cannot transfer 100$ (online[4] or offline), although they have the amount of money. □ **9.73**

Example (Reserving a course spot). Given a set of course spots being replicated to a set of mobile clients, the filtering functions map the course spots according to the preferences of gym members. That way, the subset of course spots assigned to a mobile client may contain a preferred spot offline (to make a reservation). Thus, an offline transaction can take place. Hence, a gym member can make his/her reservation offline without causing a conflict during synchronization. A disadvantage of this strategy is that a gym member cannot reserve a course spot which is not mapped to his/her identification, even if the spot is not selected by any other member. □ **9.74**

Synchronization and Reintegration

A modified replica must be synchronized with the primary copy and the replicas of the other mobile clients at some point in time. **9.75**

Synchronization scheme: The proposed generic architecture uses a hub-oriented synchronization scheme. Every mobile client synchronizes modified replicas with the primary copy on the server (hub). Since the server does not push the announced changes to the mobile clients, the mobile clients are asked to pull all the changes from the server. **9.76**

Image-based synchronization: As mentioned before, all commercial products use an image-based synchronization which exchanges the primary copy by the image (or value) of the modified replica. For this synchronization method, it is not necessary to know the semantics of the performed transaction. Unfortunately, the image-based synchronization requires an $(n-1)$ consistency of the replicas which requires that at most one replica is modified. A consistency less than $(n-1)$ produces more than one modified image which cannot all be synchronized with the primary copy (i.e., lost updates would occur). Image-based synchronization is sufficient for the Keypool method because an object is accessed by at most one mobile client. **9.77**

Transaction-based synchronization: A rarely used synchronization method is the transaction-based method which synchronizes the primary copy by reprocessing all the transactions performed on the replicas. This is possible if every transaction performed offline preserves the precondition of every other transaction performed offline. Thus the synchronization method requires that the semantical effects of all the performed transactions are known. The transaction-based synchronization does not require any consistency level and may integrate fully inconsistent sets of replicas. Transaction-based synchronization is adequate for the Escrow method because the distribution of the aggregate preserves the preconditions of the involved conflicting transactions. Since the transaction-based synchronization method covers the image-based synchronization method, the transaction-based synchronization is used for both transaction models within our architectural design assuming that the semantics of all transactions is known. **9.78**

[4] If both mobile clients are online they can transfer 100$ while using a standard transaction model.

Chapter 10

Requirements for Context Support

The following requirements reflect the context support of mobile applications. Since the two-level modeling approach mostly backs context support (cf. Figure 1.3), only a few additional modeling elements are required to provide context modeling. Following the notion of model-driven development, mobile application developers can create mobile applications with a context-aware architecture (cf. Figure 1.4), but not directly by modeling such an architecture in an explicit manner. That is because the context-specific variants are specified and realized by different design and runtime models, and a few additional model annotations and code generator settings (e.g., to produce online- and offline-capable versions of a mobile application). **10.1**

10.1 Architectural Requirements

The architectural requirements of mobile applications to be generated are mostly driven by the goal of context-awareness. The first two requirements, *support of user roles* and *heterogeneous device support*, focus directly on the *user* and the *device* context. The former requirement of single user systems with back-end access (cf. Section 4.2.2) is extended by the multi-user *interoperability* requirement to the extent that mobile applications also work on a shared set of data objects (e.g., reintegrate local modifications). Moreover, the generated mobile applications should be online- and offline-capable. **10.2**

10.1.1 Support of User Roles (User Context)

Description: The generated mobile applications should be able to support role-driven variants. **10.3**

Explanation: User roles hide, in general, the complexity of a mobile application and provide only the functionality needed for a certain role. Since the generated mobile applications will support role-driven variants for different purposes, this feature is motivated by a particular design decision. The mobile applications should be *interoperable* and provide the data acquisition or sent data from/to a back-end server. Usually, back-end servers also have to provide facilities to create and maintain application data. This requires additional development effort for the back-end system. To circumvent this development effort, especially using additional technologies (e.g., HTML and JavaScript) at the back-end site, the mobile application should also be used to acquire and maintain data so that the back end can be lightweight (e.g., only a database server without application logic). We will not further elaborate on the development of a back end in a model-driven way, in particular because the model-driven development of web applications is an already developed approach (e.g., Bohlen [24]). To realize such a front-end administration, the mobile application must support role-driven variants, providing user roles for both administrative and mobile end-user tasks. Based on requirement 4.1.2, which **10.4**

claims that the process model might be used at runtime to instantiate parts of the mobile application, the roles might be supported by this functionality.

10.5 **Acceptance:** The requirement is fulfilled if the deployed mobile application supports the different predefined user roles without the redeployment of the mobile application.

10.1.2 Heterogeneous Device Support (Device Context)

10.6 **Description:** The generated mobile applications should reflect the fact that they probably run on different mobile device types of the same software platform.

10.7 **Explanation:** The problem of heterogeneous device types should be handled by two functionalities. First, the model-driven development infrastructure should provide the functionality to adapt an app model to a device type at the design time before the mobile application is generated. The application of this technique requires the targeted group of devices to be known at the design time. Thus, the graphical user interface is configured for a particular device type but is *static*. Second, the generated mobile applications should provide a runtime configuration of the graphical user interface. This functionality exploits requirement 4.1.3, which claims that the graphical interface model may be used at runtime. Moreover, the runtime configuration is not limited to changes of the graphical user interface because the device-specific processes can be instantiated at runtime as well.

10.8 **Acceptance:** The requirement is fulfilled if the graphical user interface and the behavior can be adjusted at the design time or the runtime to a specific device type.

10.1.3 Interoperable, Multi-User Systems

10.9 **Description:** The generated mobile applications should be able to access and modify data from external systems (e.g., a database server) as part of an interoperable, multi-user system.

10.10 **Explanation:** Since most of the industrial and commercial mobile applications use external data and services, they can be characterized as *interoperable* systems. Thus, our collaborating domain experts need the generated mobile applications to be *interoperable* as well as to access external data and services. Since the accessed back-end system usually performs data and transaction management functions (e.g., by an underlying database management system), we call the data and the transaction management *centralized*.

10.11 **Acceptance:** The requirement is fulfilled if the generated mobile applications can acquire data from a back-end system and/or write modified data back to a back-end system. Additionally, they must be able to access external services (e.g., Map-Services).

10.1.4 Online and Offline Capability (System Context)

10.12 **Description:** The generated mobile applications should be able to operate in online- and offline-contexts. The local transaction manager should organize transactions depending on the required conflict level (e.g., allowing, avoiding, or prohibiting conflicts).

10.13 **Explanation:** Mobile applications pertaining to an *information system* or a *transaction system* usually perform typical functions like processing data queries or executing transactions. Mobile applications that realize such systems must comprise additional architectural components and functions to operate reliably during the changing connection states.

10.14 Considering a mobile application that implements an *information system*, its architectural design must guarantee that the data is available even if the back end cannot be reached. Hence, a *hybrid* data and transaction management system, which includes at least a *replication* mechanism, is needed. *Replication*, in turn, needs a local storage system (e.g., database) for acquired data. Thus, for a mobile application to realize a *hybrid* data and transaction management system, *local* data and transaction management is also needed, which can be used in *standalone applications* as well. To sum up, a *replication* mechanism makes an *information system* more robust in changing network conditions.

10.15 A *transaction system* needs another architectural component, i.e., *synchronization*. This component is needed because offline transactions performed on the locally stored data might modify this data. These modifications must be synchronized with the primary copies when the device reconnects to the network. Assuming that a *transaction system* is a multi-user system, any kind of *conflict* can occur when modifications are written back to the primary copies. Thus, the generation process and the resulting mobile application should reflect this and provide mechanisms to allow, avoid, and prohibit conflicts. To increase the transactional throughput of online- and offline-capable mobile applications, state-of-the-art techniques should be applied. Utilizing the well-known *mobile transaction models* provided in the literature, the generated mobile applications should work in a conflict-free manner, even for potentially conflicting processes.

10.16 To sum up, the generated mobile applications should work both offline and online by using appropriate architectural instantiation of data and transaction management components and guarantee certain conflict levels.

10.17 **Acceptance:** The requirement is fulfilled if the generated mobile applications can replicate and synchronize the modified data in a multi-user environment and perform transactions locally even if the network connection is temporarily broken.

10.1.5 Non-Functional Architectural Requirements

10.18 The functional requirements for the support of user roles (user context) and heterogeneous device support (device context) should be realized by the two-level modeling approach. Particularly, the app model should be used to realize a *design time instantiation* of role-specific mobile applications. In turn, a role-specific *runtime instantiation* of the generated mobile application should be possible by the provider model.

10.19 The device support should be realized in the same manner. A *design time adaptation* of the app model (e.g., a model transformation) supports the targeted devices if known at design time. Additionally, the provider model should be used for a *runtime adaptation* of the generated mobile applications, since the device context is often unknown at design time.

10.20 Although other contextual dimensions and contexts are not in focus of this thesis, the provider model (runtime model) could be used, in general, to adapt the mobile application at runtime to further contexts. Hence, the design of the mobile application architecture should be extendable in terms of context-support for further contexts.

10.21 A non-functional requirement related to the online- and offline-capable functionality of the mobile applications is the distribution of functionality among the mobile client and the server. The functionality of the data and transaction management should be located only on the mobile client i.e., the mobile client pulls the data objects during the replication and pushes the data objects during the reintegration to the server. In turn, no functionality except the database management systems is located on the server.

10.2 Modeling Language Requirements

10.22 Besides the modeling language requirements given in Section 4.1, the previously described online and offline capability needs the declaration of data classes, which should be handled in a connectivity-aware manner. That is, they should be part of a replication and synchronization scheme.

10.2.1 Declaration of Online- and Offline-Capable Data

10.23 **Description:** To provide online- and offline-capable data and transaction management, further annotations must be supported for the data model.

10.24 **Explanation:** While the modeling elements of the process model provide clear semantics regarding the operations to be performed on the object model, individually modeled operations need a declaration of how to act on the data objects. These implicit and explicit declarations of data access are necessary to perform a conflict analysis on the app model and ensure conflict-allowing, conflict-avoiding, and conflict-prohibiting mobile applications. Besides, the mobile application modelers should declare which sets of objects should be managed in a context-aware manner.

10.25 **Acceptance:** The requirement is fulfilled if the modeling process provides minimalistic model annotations that support the conflict analysis and the online- and offline-capable management of data and transactions.

10.3 Tool Requirements

10.26 Finally, tool requirements describe the needs for the tools that support the model-driven development process of context-aware mobile applications. The main requirements deal with modeling and the code generation and are already handled in the first part of the thesis (cf. Sections 4.3.1 and 4.3.2). In addition to the graphical model editor (used for the app model) a model editor for the provider model is required. Furthermore, the tooling set should also provide a simulation system to predict the throughput of mobile applications during different connection situations.

10.3.1 Provider Model Editor

10.27 **Description:** To create provider models, mobile application developers and providing users need a model editor that provides the creation of provider models.

10.28 **Explanation:** Based on the app model, mobile application developers or providing users want to create valid instances of this app model. In particular, they should be able to create *Object-*, *Style-*, or *Process instance model*s (cf. Figure 6.1). Hence, a graphical model editor should be available for the *Style-* or *Process instance model*. The graphical concrete syntax of the *GUI* and the *Process model* should be reused respectively. Since the data model is a standard Ecore model, a corresponding tree-based model editor can be generated by the EMF. Moreover, object models can be created with the generated mobile applications itself using the CRUD functionality.

10.29 **Acceptance:** Textual or graphical model editors to create provider models (i.e., *Object-*, *Style-*, or *Process instance model*) are available to mobile application developers and providing users.

10.3.2 Simulation System

Description: A simulation system should predict the transactional throughput of the designed and generated online- and offline-capable mobile applications under various system conditions. **10.30**

Explanation: The consulted domain experts admit that it is generally hard to predict how many conflicts actually occur during synchronization in a deployed online- and offline-capable mobile application. The number of conflicts depends on application-specific behavior (conflictuality), the available data records, and the access behavior (e.g., hot spots) of the mobile end users. Hence, the simulation system must predict the transactional throughput of a designed mobile application based on the modeled application behavior. The simulation system should work on initial artificial data (conforming to the application-specific data model). It should also work on real-world data if the application has already been rolled out. Thus, the simulation system delivers detailed data under which conditions an online- and offline-capable mobile application can be used advantageously, particularly which conflict levels should be selected. **10.31**

Acceptance: The requirement is fulfilled if the simulation system can deliver detailed simulation results for any app model that follows the domain-specific modeling language, i.e., its metamodel. **10.32**

10.4 Discussion

We have again discussed the requirements of both the first and second parts of this thesis in light of (i) completeness, (ii) consistency, (iii) feasibility, and (iv) testability, as presented in the first part of this thesis (cf. Section 4.4). We can certify that the overall set of requirements is also consistent without contradictions. **10.33**

Chapter 11

Domain Analysis (Mobile Contexts)

In this chapter, we continue the domain analysis from Chapter 5 to add context-related features to the feature model. We use the same sources of knowledge and follow the same methodology as in Chapter 5. Hence, we start directly with the feature identification and definition. **11.1**

11.1 Feature Identification and Definition

We identify additional features inside the feature group *data and transaction management* and *context-awareness*. **11.2**

Data and transaction management: The feature group is extended by *hybrid* data and transaction management. *Hybrid* data and transaction management requires *Replication*, *Synchronization*, and *Conflicting* and *Conflict-free* transaction management as possible features. **11.3**

Context-awareness: The feature group *user context-awareness* is extended by three sub-feature groups – *user context-awareness*, *device context-awareness*, and *system context-awareness*. **11.4**

User context-awareness: The feature group *user context-awareness* consists of the features *user preferences*, *user role*, and *user location*. According to our approach, the feature *role* can be subdivided into *providing user* and *end user*. **11.5**

Device context-awareness: The awareness of the *device context* comprises the *device type* feature group. **11.6**

Device type: The feature group, *device type* contains *Smartphone*, *TV stick*, *Wearable*, and *Tablet* as possible devices. Device type features usually require or excludes different hardware features. For example, a *TV stick* device type has no *sensors*. **11.7**

System context-awareness: The feature group, *system context-awareness* comprises the feature group *network condition*. The feature group *network condition* can be subdivided in *online*, *offline*, and *hybrid*. **11.8**

Network connection: The required network connection is strongly related to the data and transaction management. Features of this feature group are *offline* and *online*. **11.9**

11.2 Feature Model

The hierarchical feature model shown in Figure 11.1 extends the feature model from the first part of this thesis (cf. Figures 5.2 and 5.3) by additional features of the feature group *data and transaction management* (Figure 11.1b) and additional features of the feature group *context-awareness* (Figures 11.1c, 11.1d, and 11.1e). **11.10**

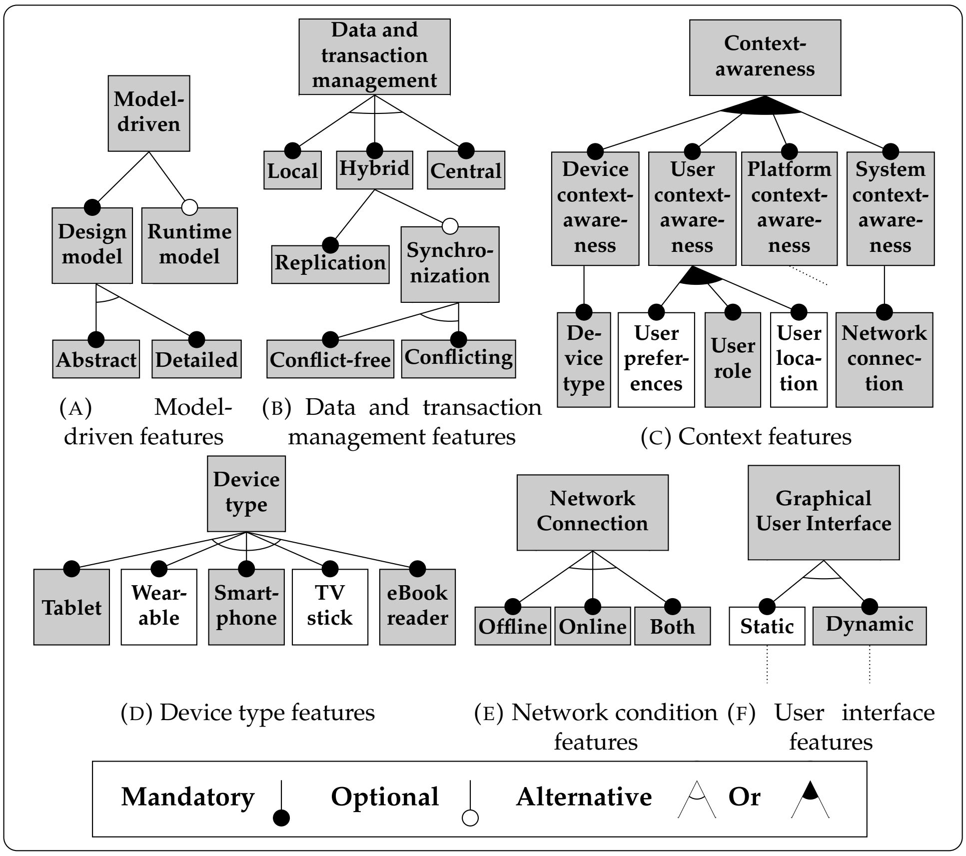


FIGURE 11.1: Feature model (Pt. III/III)

11.2.1 Feature Composition Rules

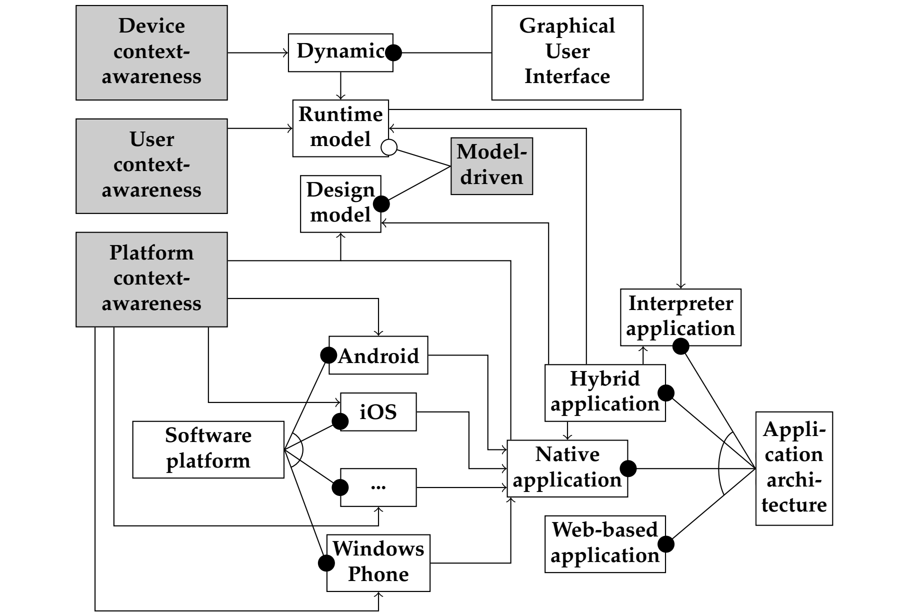
11.11 According to the extension of the hierarchical feature model, the feature composition rules must also be extended. Figure 11.2 shows the second part of the feature composition rules.

11.12 Figure 11.2a shows that the *device context-awareness* requires a *dynamic* graphical user interface, which in turn requires a *runtime model*. The *user context-awareness* feature also requires a runtime model. In turn, the runtime model requires an *interpreter application*. The *platform context-awareness* indirectly (through the concrete platform) implies the *native application* feature. Thus, both *native* and *interpreter application* features are required. Hence, the generated mobile applications are *hybrid applications* because they have native, compiled components as well as runtime components.

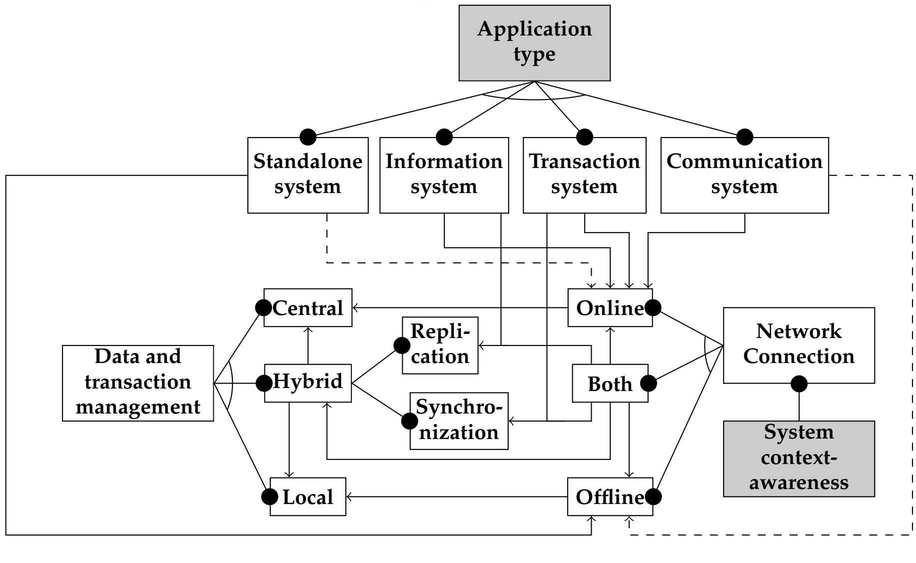
11.13 Figure 11.2b shows the composition rules for the introduced *system context-awareness* feature. There is a strong relationship between the *application type* feature and the *system context-awareness* which requires different ways of *data and transaction management*. For example, a *standalone system* requires only an *offline* network connection (and excludes an online network connection) which implies a *local data and transaction management*. In turn, a *transaction system* which should be online- and offline-capable (*both*) requires *replication* and *synchronization* as part of a *hybrid data and transaction management*.

11.2.2 Feature Binding

11.14 Considering the feature binding, the introduced context features can be instantiated both as *compile time features* and as *load-time features* (cf. Section 10.1.5). The *device*



(A) Composition rules for the feature groups *Device context-awareness*, *User context-awareness*, *Platform context-awareness*, *Software platform*, *Graphical user interface*, *Model-driven*, and *Application architecture*



(B) Composition rules for the feature groups *Application type*, *Data and transaction management*, and *System context-awareness*

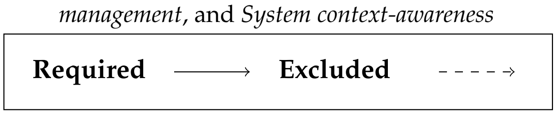


FIGURE 11.2: Feature composition rules (Pt. II/II)

context-awareness feature and the *user context feature* are additional *runtime features* because they are not necessarily limited to *load-time* instantiation. However, a running mobile application will not change the underlying device, since we see no usage scenario for *runtime* device reconfiguration. According to the *system context-awareness* feature, a runtime instantiation of this feature is not recommended. In this case, i.e., switching from online- and offline-capable mode to an online-only mode requires that all other clients will do the same because the data objects which are distributed must be completely synchronized.

11.3 Focused Features

11.15 The second part of this thesis will focus on all context-aware features. Different *user roles* (e.g., *provider* and *end user*) create the configuration of a mobile application concerning functionality, graphical user interfaces, and data. Owing to the lack of knowledge about *hardware platforms* (e.g., screen size) or *device types* of the applications' host device and the targeted user group at design time, the mobile application can be flexibly configured at runtime. Otherwise, a default runtime configuration will be derived from the design model. The generated mobile applications also support the integration of other services (e.g., calling other applications and using web services). Given that such mobile applications should be runtime adaptable in terms of *user*, *device* and *system contexts*, their architecture must support a *dynamic* instantiation of graphical user interfaces, processes, and architectural components. In order to support different network conditions (as part of the system context) in a dynamic manner, the generated application architecture supports a *hybrid* data and transaction management. In particular, the application can work *offline* and *online* according to potentially changing *network connections*. The generated mobile applications shall support *replication* and different modes of *synchronization* including *conflict-free* and *conflicting* integration of locally performed transactions.

11.16 Table 11.1 shows which requirement is addressed by which feature.

TABLE 11.1: Mapping of the focused feature groups to the requirements of the MDD framework (Pt. II/II)

Part	Requirement No.	Requirement name	Feature group/s
II	10.1.1	Support of User Roles (User Context)	Context-awareness (User context)
II	10.1.2	Heterogeneous Device Support (Device Context)	Context-awareness (Device context), Hardware platform
II	10.1.3	Interoperable, Multi-User Systems	Data and transaction management, Application type
II	10.1.4	Online and Offline Capability (System Context)	Context-awareness (Systems context)
II	10.2.1	Declaration of Online- and Offline-Capable Data	Context-awareness (Systems context)
II	10.3.1	Provider Model Editor	Context-awareness (User context, Device Context)
II	10.3.2	Simulation System	Context-awareness (Systems context)

Chapter 12

User Contexts

As shown in the presentation of the requirements Chapter 10, an aspect of general context support is the support of user contexts. As illustrated in Figure 9.1, a mobile application is aware of the user context if it can be adapted to the requirements of different mobile users. More precisely, a mobile application generated once and distributed among several mobile devices should support different *kinds* of mobile users. This should not be confused with a multi-user system, in which different mobile users act simultaneously or asynchronously on an identical mobile application installation, or with a multi-user capable back-end service. Besides, it should be clarified that the *user context* should be well differentiated from the *context of use*, which may apply to a mobile user but generally requires more sophisticated context recognition methods (e.g., motion detection, social context detection, and location-awareness) and complex context-processing mechanisms. A mobile user is usually recognized by her/his credentials (e.g., user name) or other individual properties. Hence, the context-sensing mechanism of a user context is trivial. **12.1**

Different kinds of mobile users are characterized by user-roles (or groups) that vicariously define the privileges of a set of concrete mobile end users. These privileges can reduce the accessibility to data objects, or the number of use cases or processes that are available in an installed mobile application. Hence user context-awareness requires the instantiation of mobile application variants according to the privileges a mobile user, and respectively, the associated user role, enjoys. While the context-sensing mechanism of a user context is trivial, the adaptation of mobile applications according to this user context is generally not. The design of the mobile application must reflect the fact that some of its features must be limited or extended depending on the current user, i.e., its user role. **12.2**

When using a traditional software development process, the user roles of a mobile application must usually be declared at design time. A textual description or a use case diagram might declare/visualize the actors (roles) of a software system and the use cases/processes that are available for a particular actor. Mobile application developers will implement this specification in a static way. It is usually fixed during the life cycle of the mobile application. In contrast to this traditional way of user context support by statically coded role concepts, we will show how the designed model-driven development infrastructure handles user roles dynamically and, thus, supports user context-awareness in a more flexible way. The designed model-driven development infrastructure provides both a design time and runtime approach to support different user contexts: **12.3**

12.1 Design Time Instantiation

The designed model-driven development infrastructure allows the modeling of role-based mobile application variants by modifying the process model. Based on a description of user roles and sets of accessible use cases (processes) for each user role, the mobile application developers can create variants of the process model. **12.4**

Each supported user role needs a variant of the process model which is built as follows: if the actor has a relation to a use case, the process representing this use case and its occurrence in the process selector tasks will not be changed. If the actor has no (transitive)[1] relation to a particular use case, the representing process must be removed from all process selector tasks in the process model. The resulting process model manifests the mobile application variant for a particular user role. The resulting mobile application provides only the specified processes, and respectively, the use cases.

12.5 **Example** (Design user context-aware variants of the simple phone book application). Suppose a company uses the phone book application as a company phone book. In such a scenario, a phone book manager might administer the records of the company phone book using the management process (CRUD) or the individual processes for creating, reading, updating, or deleting contacts. The employees use the phone book in a limited way. They are only allowed to read the contact records and make calls. This user role cannot make any modifications. Figure 12.1 shows a use case diagram that models the relation between the different use cases and the actors.

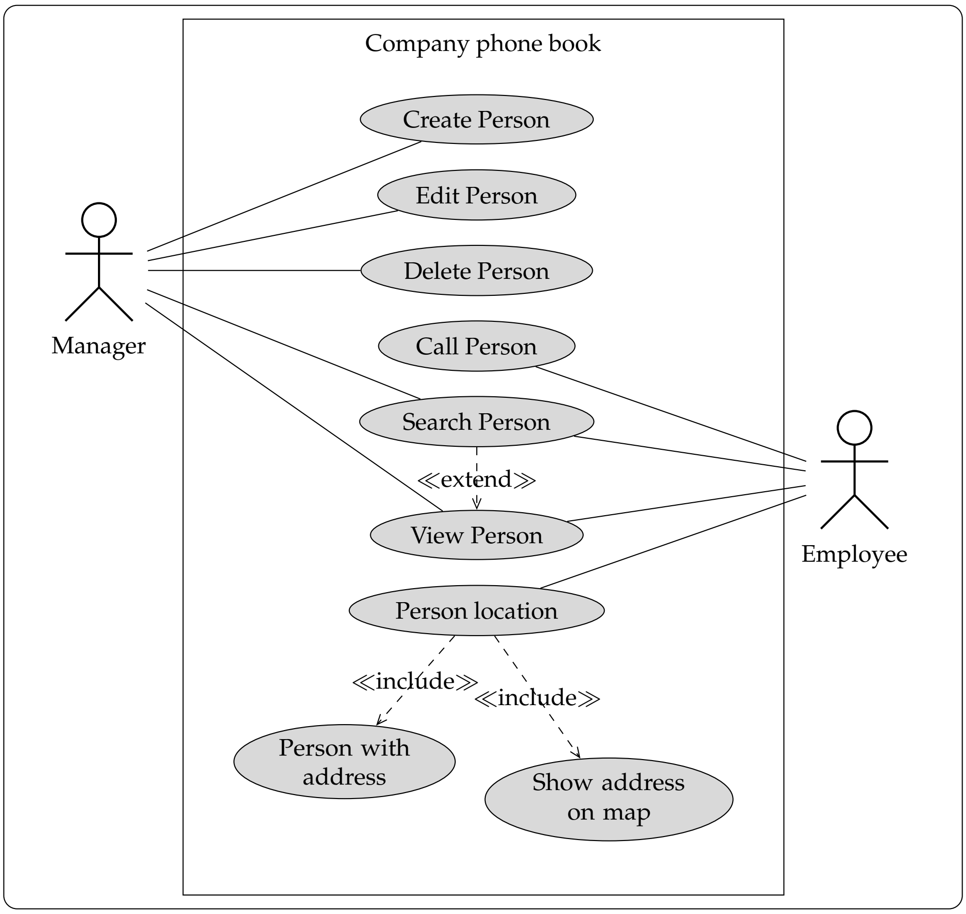
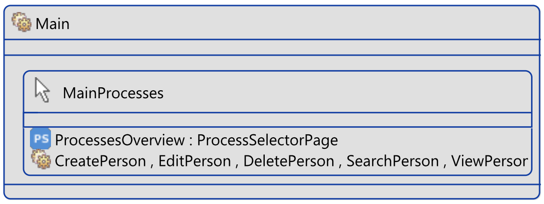


FIGURE 12.1: Actors and use cases of the company phone book application

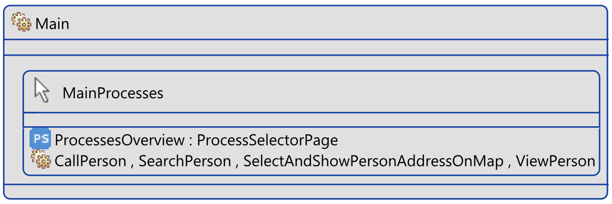
12.6 Based on this information, the mobile application developer can modify the process model (cf. Figure 6.8a) according to the given roles. Figure 12.2 shows the variation of the process model for the given roles. Processes that are no longer registered in the *ProcessSelector* Task might still remain in the process model.

12.7 After the modifications to the process model, the code generator can translate

[1] A use case may relate to other use cases by an <<extend>> or <<include>> relation. Similarly, a process relates to other processes by a sub-process relation.



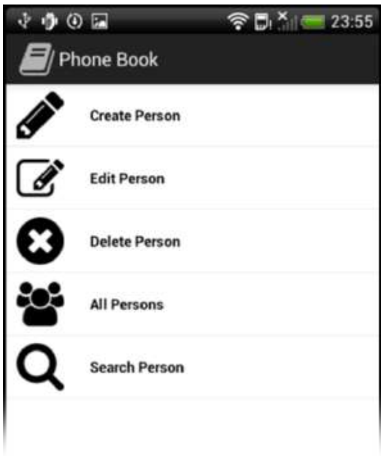
(A) Main Process for the user role *Manager*



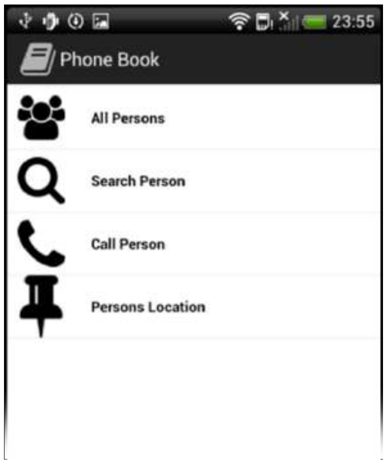
(B) Main Process for the user role *Employee*

FIGURE 12.2: Modified process model of the company phone book application (excerpt)

the app models into role-specific mobile application variants. The main entry screens of these variants are shown in Figure 12.3. Besides, the code generator recognizes the changes on the process model and generates an optimized version of the mobile applications, as shown in Table 12.1. The optimized variants of the mobile application contain only the program code that is required for the desired user context. This can reduce the program code size of the generated mobile applications considerably. □



(A) Main menu for the user role *Manager*



(B) Main menu for the user role *Employee*

FIGURE 12.3: Resulting variants of the company phone book application

TABLE 12.1: Number of processes and lines of code (LoC) of the mobile application variants

Mobile application (role variant)	**Processes**[2]	**LoC**
Corporate phone book	13	14324
Corporate phone book (Manager)	6	9606
Corporate phone book (Employee)	5	5643

[2] This number includes transitively used processes.

12.8 Providing different user contexts by design-time instantiation of processes has several advantages. The generated mobile application variants are minimalistic in terms of code size because unused program code will not be generated. This also provides more security since the generated mobile applications are steeled against the malicious activation of use cases that should not be available for a certain user context.

12.9 There are also disadvantages of the design-time instantiation method. Every user role needs its own model. More importantly, every user role runs on a separate implementation variant of the mobile application. In our case, the company phone book application might have up to 8,192 (2^{13}) variants because every subset of the set of processes (here 13) form a potential variant of the mobile application. Besides, a mobile user cannot easily switch between different user roles because s/he has to install different mobile application variants, one of each of the desired user roles.

12.2 Runtime Instantiation

12.10 To overcome the high number of potentially producible mobile applications, we apply the two-level modeling approach (cf. Section 1.4.1). This approach allows the deployment of an identical full-featured mobile application to any kind of mobile end user. The full-featured mobile application interprets the user-specific provider model and instantiates only the processes that are related to the role of the mobile user. Hence, even if the number of provider models is very high, the mobile application needs to be generated only once. This combination of the model compiler and the model interpreter approach, called the hybrid approach, provides an acceptable degree of security because the instantiated processes are static (hard-coded) and cannot be modified (e.g., by the injection of harmful runtime models) in a malicious way.

12.11 The graphical concrete syntax of the *process instance model* looks exactly like the *process model*. A part of it is shown in Figure 12.2. The only difference is that the process instance model is interpreted at runtime whereas the process model is processed at the design time of the mobile application. If the mobile application modelers decide to apply both design-time instantiation and runtime instantiation, the process instance model can only contain an equal set or a subset of the design time processes. More precisely, the design-time model must contain all the processes that should be potentially available at runtime. A design-time configuration cannot be extended in terms of the available processes during runtime, since the required program code is not included in the deployed mobile applications. Despite this, redeployment might solve this problem.

12.12 The user roles may also cover the security and privacy aspects of mobile applications. As explained earlier, one of the design decisions is to model permissions – not globally, at the level of application security, but locally, at the level of process tasks. Thus, the generated mobile application can be extended or limited according to different permission levels. For example, a mobile user who wants the mobile application to not access the phone application on a smartphone can select a corresponding user role that excludes the process *Call person*. Similarly, user roles can be created for completely different motivations (e.g., security, privacy, energy consumption).

12.13 Besides, the runtime instantiation method also offers the provision to extend or limit the available processes temporarily (e.g., disconnected mobile device, insecure network connection). This is a very beneficial property regarding conflict-handling in offline situations, as we will show later (cf. Chapter 13).

12.2.1 Runtime Instantiation Implementation

The runtime instantiation of provider models, i.e., *process instance models*, is implemented in the following manner: the generated mobile application tries to load a *process instance model* (provider model) at the start time. If no *process instance model* is given, the mobile application uses the default configuration (cf. Figure 12.2) of processes that are given by the *process model* (app model). Otherwise, the *process instance model* will be loaded. According to requirement 4.1.2 (Abstract and Detailed Behavior Modeling) and the domain-specific modeling language definition of the provider model given in Section 6.3.4, a *process instance model* (provider model) is on the same abstraction level as the *process model* (app model) and uses the same syntax. Hence, the loaded *process instance model* also contains a *Main* process. This *Main* process reconfigures the processes that are shown in the main menu of a mobile application in the sense that it defines which processes are instantiated or not. However, for reasons of safety, a *process instance model* cannot redefine an existing process (e.g., introduce another behavior) or define a new process. Instantiated processes must always be defined in the *process model*; otherwise, they cannot be instantiated. **12.14**

12.3 Demonstration

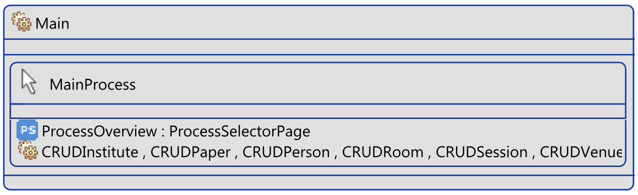
As stated earlier (cf. Section 10.1.1), the general architectural design of the generated mobile applications also includes back-end functionality. Hence, the generated mobile applications include processes that deal with either the creation or the consumption of data objects. This often requires the separation of at least two user roles – one for the data providing users and another for the mobile end users. In the following section, we demonstrate the support of user contexts, i.e., the role-driven variants, of our generated mobile applications with two case examples: a *conference application* and a *word trainer application*. **12.15**

In order to evaluate the feature of user context support, we discuss the question, whether the two-level approach can be used in practice (e.g., real scenarios) as support for user context. The case examples shall demonstrate that the two-level approach can be used in practice and support the context of users by a flexible runtime instantiation of processes according to user roles. Moreover, the approach is also beneficial in a business case, e.g., to unlock different purchasable use cases (not shown). **12.16**

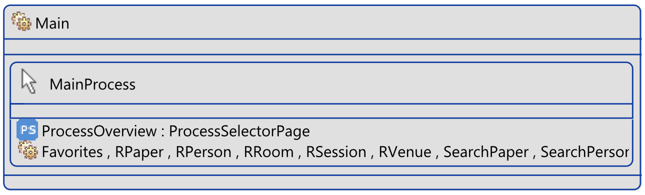
12.3.1 Case Example 4 (Conference Application)

The *conference application* guides participants through a scientific conference by providing information about events and their scheduling and location similar to a printed conference program. The mobile application provides additional functionality for conference administrators to create and modify data objects (e.g., papers, session, rooms, speakers, session chairs, and authors). The generated mobile application was given to all participants of the MoDELS 2014 conference in Valencia, Spain [44]. **12.17**

The app model includes a data model (Figure B.16) that reflects the domain-specific data objects mentioned before, a process model (Figures B.18 ff.) that provides all the necessary processes for managing and viewing data objects (e.g., CRUD processes), and a GUI model (Figure B.17) that provides a default design for the generated mobile application. The app model can be reused for other conferences because the customization (e.g., conference-specific design and color schemes) is realized entirely by the provider models. Thus, the app model provides neither **12.18** concrete data objects nor conference-specific processes or styles, which makes it reusable in the same domain.



(A) Main Process for the user role *Conference Administrators*



(B) Main Process for the user role *Conference Participants*

FIGURE 12.4: Main processes of the conference application

TABLE 12.2: Use cases for the conference administrators (Providing User)

Icon	Use Case	Description
	Main Menu	The conference administrator variant shows the main menu with a set of available use cases (*ProcessSelectorPage*) and allows entry into a CRUD-process (*CrudGui*) with full permission (ALL) for the selected entities: Institute, Paper, Person, Room, Session, and Venue.
	Institute (CRUD)	These use cases allow creating, reading, updating, and deleting institutes, papers, persons, rooms, sessions, and venues. A CRUD use case is processed as follows: An entity can be selected from a list of entities (*SelectableListPage*). The simple selection of an entity just shows the user its details in a non-editable-form (*ViewPage*). In this case, associations with other entities are shown in a tabular form. Before editing (updating) an entity, the user has to choose it from a list of the available entities. A long tap on the screen opens it in the edit mode. To edit an entity (create or update), a single view is shown as well, allowing the user to edit all details (*EditPage*). Associations between the entities can be set in a drop-down-list (1:1 cardinality) or a list of checkboxes (1:n cardinality).
	Paper (CRUD)	
	Person (CRUD)	
	Room (CRUD)	
	Session (CRUD)	
	Venue (CRUD)	

12.19 The provider model for the conference administrators (cf. Figure 12.4a) enables the processes that are relevant for data acquisition (see Table 12.2). In turn, the processes that are designed only for the mobile end users, i.e., conference participants, are not available in the administrator's variant. The object model might be empty, i.e., the mobile application starts without a prepared data set. The style model might provide a conference-specific style, otherwise, the default style is used.

Figure 12.5 shows the generated mobile application being configured for the confer- 12.20
ence administrators.

(A) Main Menu with default CRUD processes
and the standard navigation of CRUD process for the entity *Paper*

(B) Main Menu with default CRUD processes
and the standard navigation of CRUD process for the entity *Session*

FIGURE 12.5: *Conference Administrators* (Providing user) view of the mobile application

The provider model for the participants of the conference (cf. Figure 12.4b) enables 12.21
the mobile end user processes (see Table 12.3), i.e., read-only access to papers, sessions, rooms, etc., of this conference. Furthermore, the sessions can be marked as favorites. A new entry in the user's calendar can be inserted, so as not to forget a favorite session. Since the standard read processes provide no convenient search function, the processes *SearchPaper* and *SearchPerson* provide such functionality. The object model contains the data objects that are created by the conference administrators beforehand. The style model usually provides a conference-specific design scheme, as it is in our case.

TABLE 12.3: Use cases for the conference participants (Mobile end user)

Icon	Use Case	Description
	Main Menu	This conference participant variant shows the main menu with a set of available activities (*ProcessSelectorPage*). It offers a CRUD-process (*CrudGui*) with restricted permission (read-only) for the selected entities: Paper, Person, Session, and Venue. Room is a special use case in this list, supporting the depiction of room plans. In addition, there is a special use case, Favorites, and users can add a session to it.
	Add Favorite (Favorites)	In this use case, all the existing sessions are displayed in a list (*SelectableListPage*). The user can select a session that is added to her/his favorites. Furthermore, an entry (with the selected session data) is created in the calendar.
	Remove Favorite (Favorites)	In this use case, all the favorites are displayed in a list (*SelectableListPage*). The user can remove a session from favorites.
	Paper (R)	These use cases help list all the entities of the indicated type, e.g., list all papers. The list entries may be selected to view all details of those entries.
	Person (R)	
	Session (R)	
	Room (R)	In this use case, all existing rooms are displayed in a list (*SelectableListPage*). The user can select one object and see its details (Plans) in MediaPages. The details show all the available plans saved for this room. The user can select one plan and see an image with the marked room.
	Search Paper	These use cases offer a search function of the indicated type, e.g., papers. The first appearing form (*EditPage*) gathers the search criteria. Subsequently, a result list appears (*SelectableListPage*). The user can select one object and see its details in a separate view (*ViewPage*).
	Search Person	
	Venue	In this use case, all the existing venues are displayed in a list (*SelectableListPage*). The user can select an object and see the position in Google Maps (*MapPage*).

12.22 Figure 12.6 shows the generated mobile application being configured for the conference participants.

(A) Main Menu with default read processes
and the standard navigation of read process *Session*

(B) Process *Add/Remove Favorite*

(C) Process *RRoom*

(D) Process *SearchPaper*

FIGURE 12.6: *Conference Participants* (Mobile end user) view of the mobile application

12.3.2 Case Example 5 (Word Trainer Application)

12.23 The *word trainer application* is the result of an interdisciplinary project between the Institute of Computer Science (Software Engineering research group), in which the model-driven development infrastructure was developed, and the Institute for German Literature and Linguistics (research group of *German as a Foreign Language*), both located at the University of Marburg.

12.24 The aim of this word trainer application is to support migrant learners with little or no knowledge of German and little literacy skills to learn vocation-based vocabulary. The content is supported by pictures and videos. The word trainer application provides pronunciation, as well as various means to test the learner's knowledge. Here again, two role variants are needed. First, language teachers create content (e.g., words, pictures, audio records, and data for self-assessment) for the mobile application. Second, the learners use this content to improve their job-specific language skills. However, both user roles provide the ad-hoc creation of content, using the facilities of the mobile device (e.g., record a spoken word, or take a picture with the built-in camera). This design makes a back-end application for the creation of a data object needless. Particularly, the hardware facilities of mobile devices (e.g., microphone and camera) should be employed during the ad-hoc creation of the learning content. The ad-hoc creation allows learners to add job-specific vocabulary during on-the-job periods, i.e., while getting an instruction or processing a task. Since a lot of study content already exists, the domain experts need an import mechanism for the study content. A manually coded component allows *bulk import* of data objects that are converted from widely accepted standard format files (e.g., Spreadsheets). Figure 12.7 shows the final architecture of the word trainer application. The generated word trainer application was tested during different laboratory experiments with real learners and could finally be released for the targeted group of learners [43].

12.25 The word trainer application was mostly realized as part of a student laboratory project and a final project (cf. Section 7.4.4). The final project included the realization of the required changes to the model-driven development infrastructure (e.g., the introduction of an eLearning Page, an extension of the code generator), as well as the model-driven development of the word trainer application. Hence, the developer acted both as the infrastructure developer and the mobile application developer while applying the agile bottom-up approach presented in Section 3.1.

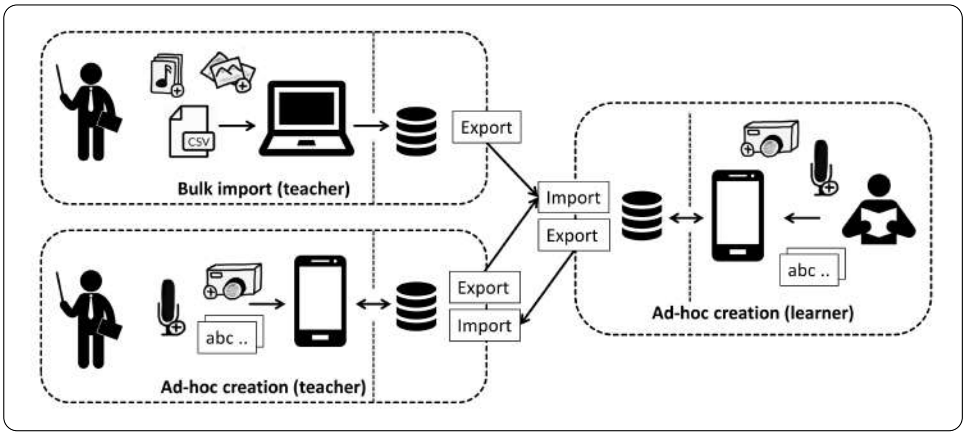
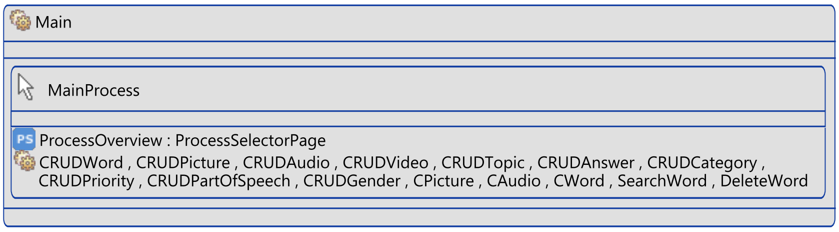


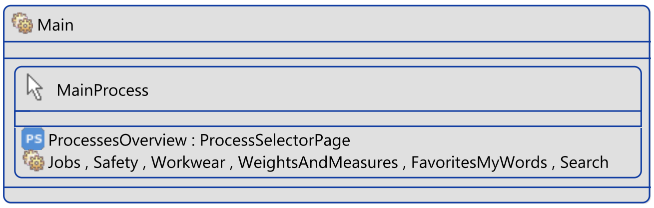
FIGURE 12.7: Architecture and roles of the mobile learning application

12.26 The app model includes a data model (Figure B.28) that reflects domain-specific data classes for learners and teachers, a process model (Figures B.30 ff.) that provides all the necessary processes for managing and viewing data objects, and a GUI model (Figure B.29) that provides a default design for the generated mobile application.

The provider model for the teachers (see Figure 12.8a) enables the processes that are 12.27
relevant for the creation of the vocabulary (see Table 12.4). In contrast, the processes that are designed only for the end users, i.e., the learners, are not available in the teacher's variant. The object model might be empty, or a set of records could be imported from another device. In its current form, the word trainer use no specific style set, i.e., the default style set is used.



(A) Main Process for the user role *Teacher*



(B) Main Process for the user role *Learner*

FIGURE 12.8: Main processes of the word trainer application

Figure 12.9 shows the generated mobile application being configured for the teachers. 12.28

FIGURE 12.9: *Teachers* create a word object starting from the main menu

The provider model for the learners (Figure 12.8b) enables the mobile end-user 12.29
processes (see Table 12.5), i.e., the learning and testing processes of the different topics. Furthermore, the learner can create favorites and search within the vocabulary. The object model contains the records prepared by the teachers. In addition, the learners can add own words to the vocabulary. A particular style model is not provided. Hence, the teacher's and the learner's variants of the word trainer application follow the default design.

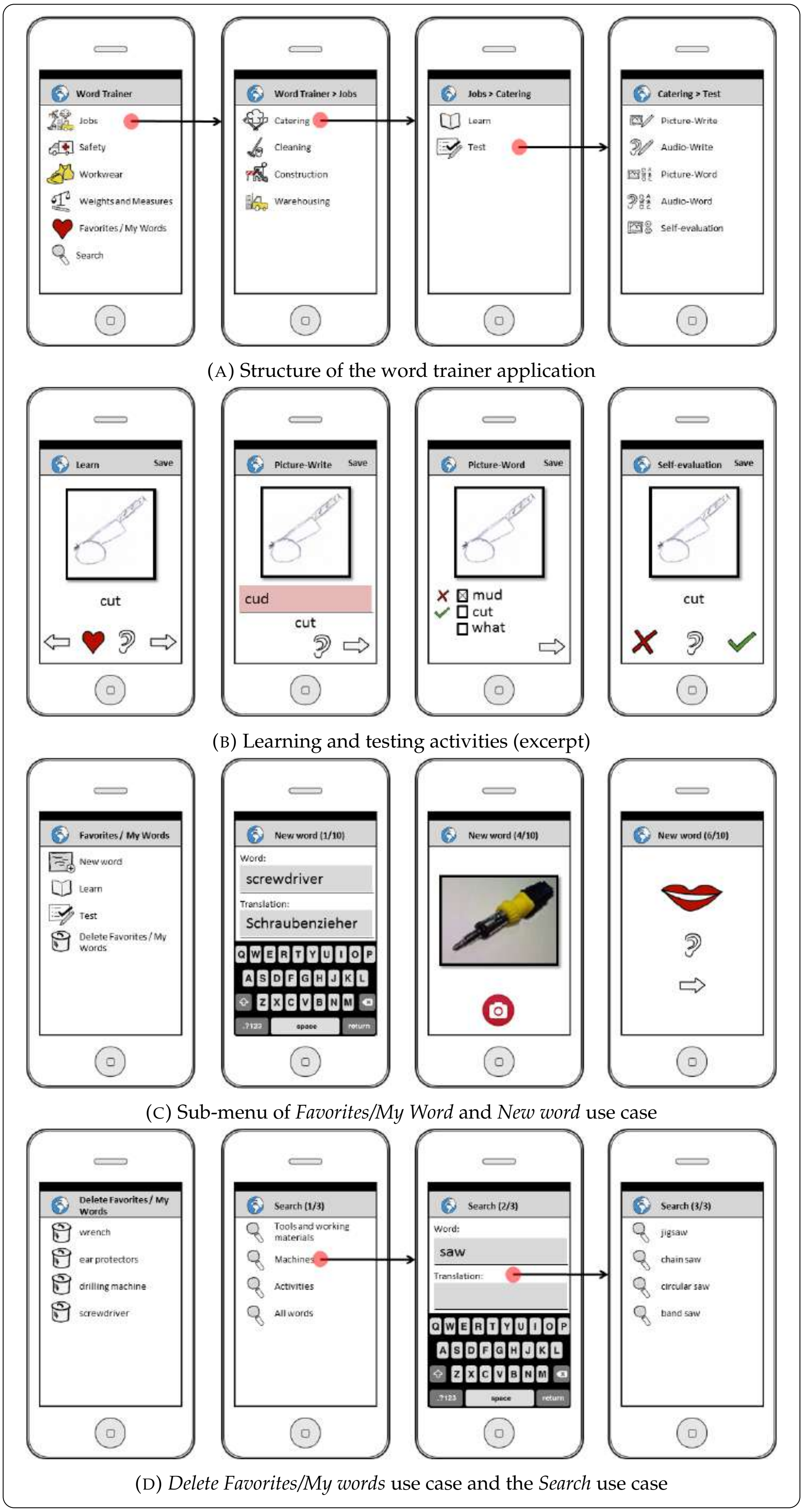
TABLE 12.4: Use cases for the teachers (Providing User)

Icon	Use Case	Description
	Main Menu	This word trainer variant shows the main menu with a set of available activities (*ProcessSelectorPage*), and allows the user to enter a CRUD process (*CrudGui*) with full permission (ALL) for the selected entities: Word, Picture, Audio, Video, Topic, Answer, Category, Priority, PartOfSpeech, and Gender. Additionally, the application variant provides five individual processes for the creation of the learning content.
	Word (CRUD)	These use cases allow creating, reading, updating, and deleting of words, pictures, audios, videos, topics, answers, categories, priorities, part of speech, and gender objects. The modification of data objects by the CRUD processes works as described for the *conference application* (cf. Figure 12.5). However, the dependency of the data classes requires a particular order of creation. For example, the creation of a word object first requires the creation of dependent objects like a picture, an audio, or the like. Otherwise, the required features of the word objects cannot be offered, and the object is invalid. To circumvent this problem, the word trainer variant for the teachers provides customized processes to make the creation of data objects more convenient.
	Picture (CRUD)	
	Audio (CRUD)	
	Video (CRUD)	
	Topic (CRUD)	
	Answer (CRUD)	
	Category (CRUD)	
	Priority (CRUD)	
	PartOfSpeech (CRUD)	
	Gender (CRUD)	
	Picture (Create)	In this use case, a new picture object can be created. The process opens a *TakePicturePage*, which uses the built-in camera of the device to take a picture.
	Audio (Create)	In this use case, a new audio object can be created. The process opens a *RecordPage*, which uses the built-in microphone of the device to record a spoken word.
	Create Word	The use case *Create Word* case provides the creation of a new word object. It processes the steps for creating the referred sub-objects (e.g., audio or picture objects), and eases the creation of word objects.
	Search Word	The use case *Search Word* provides a search function on the vocabulary.
	Delete Word	The use case *Delete Word* allows the deletion of word objects.

12.30 Figure 12.10 shows the generated mobile application being configured for the learners.

TABLE 12.5: Use cases for the learners (Mobile end user)

Icon	**Use Case**	**Description**
	Main Menu	This learner variant shows the main menu with a set of available activities (*ProcessSelectorPage*). It offers only individual processes for learning and testing the vocabulary. Furthermore, the existing words can be added or removed to/from a personal vocabulary, new words can be created or deleted, and the vocabulary can be searched.
	Jobs	This use case guides the learner to different job-specific topics (e.g., Catering, Cleaning, Construction, and Warehousing). Each of these categories provides a study mode and different testing modes.
	Safety	These use cases guide the learner to a general vocabulary of the respective category.
	Workwear	
	Weights and Measures	
	Learn	This use case displays the word that should be learnt, a picture of this word, and an audio file for its pronunciation. The user can add or remove the word to/from her/his personal favorites.
	Test	The use case *Test* guides the learners to a selection of different test modes.
	Picture-Write	The *Picture-Write* test checks the learner's writing skills. The learners see the picture of a word learnt earlier and try to write the corresponding word. Due to ambiguity in the pictures (the learners may recognize different objects), an audio file replays the word in question.
	Audio-Write	The Audio-Write test activity is similar to the Picture-Write test activity, with the sole difference being that it displays no picture of the word in question.
	Picture-Word	The Picture-Word test checks a learner's reading comprehension by showing a picture and offering multiple choices of answers.
	Audio-Word	The Audio-Word test is similar to the Picture-Word test, but with the difference that an audio is used in this case instead of a picture.
	Self-Evaluation	The Self-evaluation test presents a word, a picture, and an audio. The learners can decide if they know the word or not.
	Favorites/ My Words	The use case *Favorites/My Words* provides learning and test modes for the personal vocabulary. Furthermore, new words can be created and deleted.
	Create Word	The use case *New Words* is a customized process that provides the creation of personal words.
	Delete Favorites/ My Words	The use case *Delete Favorites/My Words* removes words from the personal vocabulary. If the word is part of the existing vocabulary, the learner cannot delete it. It is only removed from the personal vocabulary. If the word was created by the learner, it will be deleted.
	Search Word	The use case *Search Word* provides a search function on the vocabulary.



(A) Structure of the word trainer application

(B) Learning and testing activities (excerpt)

(C) Sub-menu of *Favorites/My Word* and *New word* use case

(D) *Delete Favorites/My words* use case and the *Search* use case

FIGURE 12.10: *Learners* view of the mobile application

Chapter 13

Device Contexts

Although platform-specific code generators (e.g., Android, iOS) already handle the support of the software platform context, these software platforms are installed on different device types. For example, an Android application might be run on a smartphone, a tablet, a TV-stick or another device (cf. Figure 11.1d). Moreover, each device type is represented by a concrete device from a plethora of different devices from different vendors. Actually, many mobile applications lack device compatibility, since graphical user interfaces cannot be guaranteed to always work as expected. Similarly, the functionality – which relies on specific sensors, interfaces, or other facilities (memory, processor) – cannot be executed if a device does not support it. Many mobile applications are rated poorly based on negative user experiences. In general, mobile application developers cannot specify (or test) all device-specific variants to cover the vast set of different device types. A pragmatic way to deal with device heterogeneity is to reduce the set of platform-supported devices. An example would be iOS which runs only on a very small number of vendor-specific devices (iPad, iPhone, and iPod). In turn, more open platforms, such as Android, do not follow such a paradigm. **13.1**

The *device type* (cf. Figure 11.1d) often implies the available features, such as *Screen*, *Sensor*, *Interface*, and *Memory* properties. Hence, the device context-awareness needs a mobile application to be adapted to the features of the device. While the context-sensing of the device type or the particular features are provided by built-in functions of the platform, the adaptation of graphical user interfaces and the application functionality is not trivial. For example, since most software platforms provide both declarative and programmatic specifications of graphical user interfaces, not all properties that can be set in a declarative manner (i.e., an XML layout description) are configurable at runtime by a programmatic specification. Additionally, parts of an application that require specific sensors, which might not be available on every device, must react adequately to the hardware configuration of a particular device, which must be managed by the mobile application developers. Currently, the facilities to adapt a designed graphical user interface automatically are very limited (e.g., relative layout dimensions, graphical resources with different resolutions and sizes), even if the literature proposes several concepts. Similarly, a written native program code that deals with hardware features must be adapted manually to provide an adequate variant for every potential device. **13.2**

The model-driven development infrastructure deals with the problem of device context-awareness in the following way: the different concerns (data, behavior, and graphical user interfaces) that are captured in an app model can be transformed at design time to meet the needs of a device type or a particular device. Hence, in this work, we show the general application of app model transformations, rather than a comprehensive catalog of app model transformation rules of every potential device type. Unfortunately, the resulting app model variants lead to a huge set of mobile applications. To alleviate this problem, runtime adaptation of device-specific options can be used. Hence, the designed model-driven development infrastructure provides both a design time and runtime approach to support the different device contexts: **13.3**

13.1 Design Time Adaptation

13.4 The model-driven development infrastructure supports the adaptation of mobile applications to device-specific requirements by adapting the corresponding app model at the design time. Although an app model can be modified manually, a transformation-based and semi-automated adaptation of the app models is used based on the available model transformation tools (cf. Section 6.9.2). Generally, the app model transformation is not fully automated because its rules might need manual post-processing steps (e.g., fixing the annotated code, which cannot be transformed automatically). An app model is transformed endogenously, i.e., input and output models are expressed in the same modeling language. An advantage of transformation-based adaptation is that the transformation rules can be reused for the different app models. Moreover, the transformation rules may be composed, which is useful because the device features can occur in almost any combination. Whether the app model transformation rules can be composed arbitrarily or obtain a particular order depends on their conflicts. We do not discuss this conflict analysis within this work, but refer to the available foundations and tooling [Bor+15].

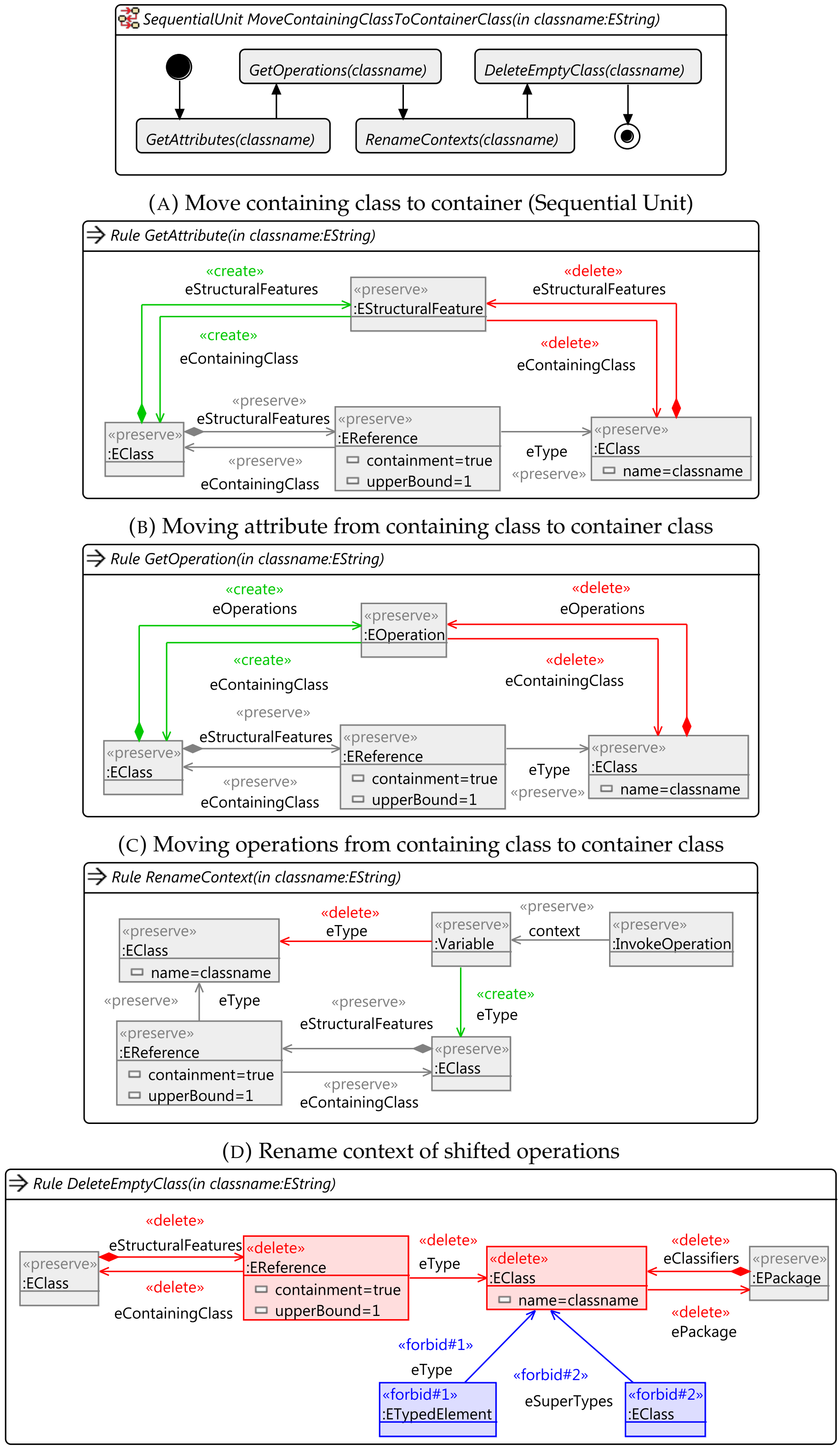
13.5 To sum up, the procedure of design time adaptation is as follows: the app model will be transformed in such a way that it (more precisely the generated mobile application) fits better to a particular device. This transformation runs automatically. If the device requires very specific adaptations, different transformation rules may be applied. Hence, these transformations should keep the original model for further adaptations.

13.6 **Example** (Transform the simple phone book app model to a tablet/notebook version). As Figure 6.2b (right-hand side) and Figure 7.5 show, the separation of classes generates the tabbed fragments or widgets that allow convenient access to the data while using a smartphone. Not too much information is displayed inside a single screen, nor is the information truncated.

13.7 Assuming again that the phone book application is used in a company scenario, the phone book managers might prefer a device with a keyboard and/or a large screen, e.g., a tablet or a notebook, because their tasks focus on the creation of data. The screen of the tablet or notebook should be used optimally. Reusing the formerly used app model, particularly the data model, leads to a poorly utilized screen with only a few graphical user interface elements. Moreover, a hardware keyboard is used instead of an emulated software keyboard, which frees some space on the screen. Hence, to provide more information on a single screen, the data model is be transformed. A class that contains another class (max. `upper bound=1`), such as *Person* and *Address* (cf. Figure 6.4), is merged by moving the attributes and the operations to the container class. Figure 13.1 shows the transformation rule and the sub-rules that implement the model transformation. The rules are formulated in the abstract syntax. The color scheme indicates the parts that should be matched and *preserved*, *deleted*, *created*, and the structures that are *forbidden*. The argument *classname* (e.g., *Address*) of the main rule (cf. Figure 13.1a) denotes the class that should be merged with the container class. This argument is mapped according to the sub-rules.

13.8 Particularly, the rule consists of four sub-rules that affect the data model and the process model. The first sub-rule *GetAttribute* (cf. Figure 13.1b) deletes the relations from the containing class to the structural features (e.g., *EAttribute*, *EReference*), preserve the structural feature, and create relations from the container class to the structural feature. Hence, the structural feature now belongs to the container class. This rule is applied several times inside a *Loop Unit* (not shown) until it cannot be matched i.e., executed again.

13.9 The second sub-rule *GetOperation* (cf. Figure 13.1c) works in an analogous way and



(A) Move containing class to container (Sequential Unit)

(B) Moving attribute from containing class to container class

(C) Moving operations from containing class to container class

(D) Rename context of shifted operations

(E) Delete (empty) containing class

FIGURE 13.1: Rule #1 – Move containing class to container (rules and sub-rules)

moves all operations (*EOperations*) from the containing class to the container class.

The third rule (cf. Figure 13.1d) deals with operation calls, which refer to the shifted 13.10
operations. The attribute *context* of the task *InvokeOperation* refers to a *Variable* which

holds a typed object. The operation will be executed on this object. Since the type (*eType*) of the *Variable* is no longer valid due to the shifting of the operation, it must be changed to the correct type, since it is the container class (e.g., Person). This transformation affects the process model.

13.11 Finally, the fourth sub-rule (cf. Figure 13.1e) deletes the containing class. Since the structural features and operations are removed in the preceding steps, the container class must be empty. However, the containing class can serve as a type or supertype of another class or part of an inherited relation. In this case, the containing class cannot be deleted and the steps executed before are not saved.

13.12 However, model transformations that allow adaptation to specific devices are not always focused on the data model or pure reorganization. The following model transformation affects both the process model and the GUI model by introducing tasks and pages. While using a notebook without a touch-screen, it might be difficult for the users to scroll through the long lists as they appear on several pages (cf. Figure 6.2c and Figure 6.2d). But, text input can be done better by using however the hardware keyboard. Thus, the user might want to specify the search criteria beforehand, instead of selecting an item from an unfiltered set. Thus, it would be beneficial to provide a search-criteria page before every list-selection page to reduce the number of objects appearing in a list. Figure 13.2 shows a rule that transforms such processes by introducing a search page.

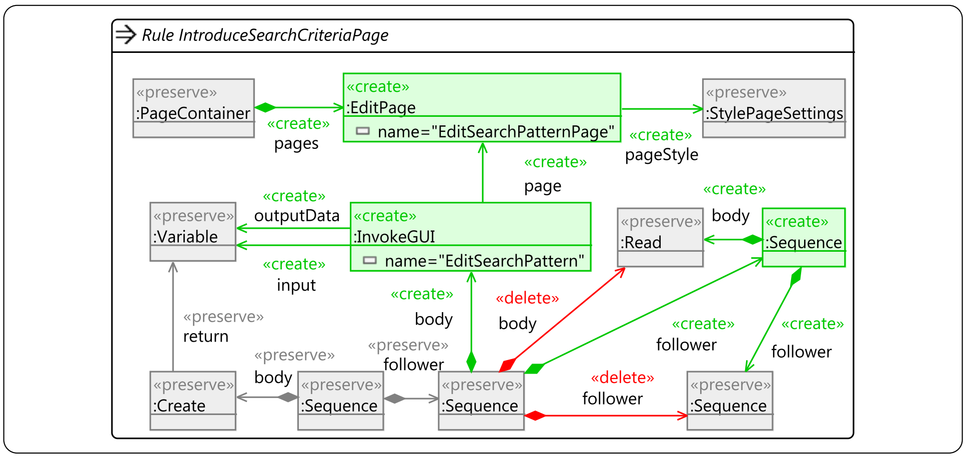


FIGURE 13.2: Rule #2 – Introduce search criteria pages before read tasks

13.13 The app model transformation rule shown in Figure 13.2 recognizes the sequences of *Create* tasks followed by a *Read* task. In such a case, an *InvokeGUI* task is introduced between these tasks by shifting the *Read* task to a newly created *Sequence* task. The introduced *InvokeGUI* task needs a corresponding *EditPage* for editing the search criteria. The *EditPage* is part of an existing *Page Container* and refers to a *StylePageSettings* element.

13.14 Before showing the resulting device-specific mobile application, we will discuss the advantages of applying app model transformations over the traditional refactoring or modification of the codebase without model-driven development methods. First, the app model transformation rules are defined at the metamodel-level. Hence, the app model transformation rules can be reused potentially for any instance of the domain-specific modeling language, i.e., any app model. In contrast, a modification of the mobile application at the code level could not be reused to adapt other implementations to a device type or device.

13.15 Second, we can evaluate the effort needed to adapt a mobile application using traditional code refactoring in contrast to the high-level approach of model transformation. Therefore, we generate the native program code from the unmodified phone book app model. Subsequently, the presented app model transformations

TABLE 13.1: Number of adapted elements at modeling level and implementation level

Applied rule/s	**Modeling level (counting *ENamedElements*)**				**Implementation level (counting files)**			
	Deleted	Modified	Added	Total	Deleted	Modified	Added	Total
-	0	0	0	165	0	0	0	291
Rule #1	13	11	0	152	26	102	0	265
Rule #2	0	12	18	183	0	58	10	301
Rule #1∘#2	13	23	18	170	26	104	10	275

are applied to the phone book app model and the native program code is generated anew. Based on both implementations, the number of adapted artifacts can be determined. Hence, we can estimate how many traditional changes at the level of code would be necessary by comparing both generated implementations. Table 13.1 shows the results. Since the number of changed code-based artifacts is very extensive in contrast to the small set of used rules and affected model elements, we have the premise that app model transformation could generally benefit the modification of mobile applications to meet device-specific requirements.

FIGURE 13.3: Tablet/Notebook version of the phone book application

Finally, Figure 13.3 shows the resulting device-specific mobile application variant **13.16**
of the corporate phone book application from the perspective of a providing user. Compared to the smartphone version of the phone book application, the creation of a contact (cf. Figure 6.2b) is more convenient on a tablet, because the whole record

can be entered on a single form and the user may enter data more efficiently using a hardware keyboard. □

13.2 Runtime Adaptation

13.17 As presented in the previous section, the app model transformation always delivers an individual app model and a corresponding individual mobile application. At present, the app stores or app marketplaces provide no device-specific query response, i.e., they cannot automatically deliver a mobile application variant optimized for the mobile user's device. The reason for this policy is that most mobile application vendors cannot or will not provide several device-specific versions of a mobile application because the development effort will increase considerably. Due to this deployment policy, a single mobile application that should cover device-specific features of different devices must be configurable at runtime.

13.18 Again, the two-level modeling approach (cf. Section 1.4.1) is used to adapt the mobile application to specific device contexts. According to the requirements of the domain-specific modeling language given in Sections 4.1.1, 4.1.2, and 4.1.3, only the process model and the GUI model should be interpreted at runtime. Similarly to the runtime instantiation of the mobile application for the user context support, the *process instance model* can also be used to activate/deactivate device-specific variants of the processes. Additionally, the *style model* is used to adapt the graphical user interface of the mobile application to a particular device. This feature can also be used to provide custom settings for the graphical user interface (user settings). The ability of runtime adaptation should not be confused with a self-adaptive system. The architecture of the generated mobile application provides device-specific adaptation as shown in the following example. However, the adaptation is always triggered manually (e.g., loading a particular instance model at runtime).

13.19 **Example** (Transform the phone book runtime model to a non-GPS compatible and low-resolution variant). Suppose the corporate phone book application has to be installed on older devices without GPS and with a small screen size. The lack of GPS functionality causes an error when displaying the contacts *NearToMe* (cf. Process B.1.3.13). The low screen density might, in general, cause problems in the legibility of the graphical user interface widgets.

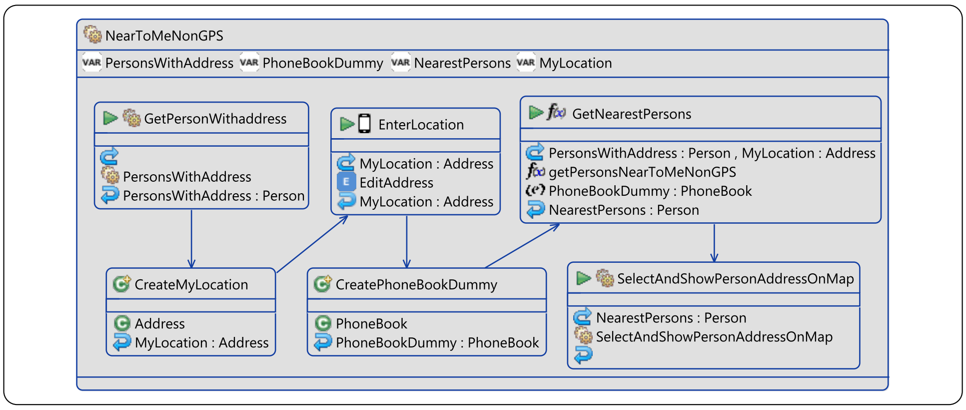
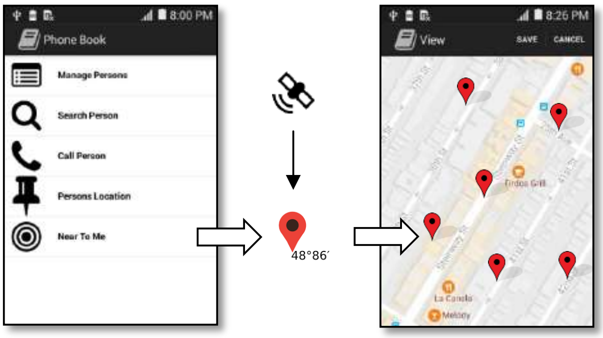


FIGURE 13.4: Non-GPS variant of the process *NearToMe*

13.20 First, to deal with the lack of GPS functionality, mobile application developers should offer processes that can provide the required functionality in other ways. The process *NearToMe (NonGPS)* (Figure 13.4) models the same functionality of the original process *NearToMe* (cf. Section B.1.3.13). However, getting the location of the mobile application user is realized without using the GPS functionality. Particularly, the mobile user enters a postal address of his/her current position, and an address look-up service delivers the geographical coordinates (like GPS does). Figure 13.5 shows the processes and their differences *NearToMe* and *NearToMe (NonGPS)* from a mobile user perspective. The instantiation of these processes is done in the same way as that for the user context support (cf. Figure 12.1).



(A) Main Menu with *NearToMe* Process using GPS functionality

(B) Main Menu with *NearToMe* Process
using address look-up service for emulating GPS

FIGURE 13.5: Device-specific variants of the process *NearToMe*

Second, the low screen size of the used device causes downscaling of the graphical **13.21**
widgets, which makes the text inside these widgets barely legible, especially if the contrast between the background and the font color is not very high. Thus, the downscaling may violate the given guidelines (cf. W3C WAI WCAG 2.0 – Web Content Accessibility Guidelines 2.0 [76]; also [Mei+10]), which are obligatory for mobile applications of certain sectors (e.g., public administration[1]). For example, Guideline No. 1.4.3 of the mentioned WCAG requires a contrast ratio of 4.5:1 for normal text (<18 points) and 3:1 for large text (>= 18 points). Suppose the mobile application shown in Figure 6.2 is deployed and executed on a 4.7" device with a resolution of 325 x 578 pixels (density 141 pixels per inch). Given that the font size of the generated mobile application is 36 dp (device-independent points)[2], the physical font size that appears on the screen is greater than 18 physical points (0.25 inches/0.635 cm). Hence, the given contrast ratio of the font color (`#555555`; RGB[85,85,85]) and the background of a selected element (`#7ecce8`) is 4.15:1, and thus valid for the font size occurring in the mobile application. In turn, if the same mobile application runs on a 4.7" device with a resolution of 415 x 737 pixels (density 180 pixels per inch), the font size in the mobile application is only 14.4 physical points. In this case, the provided contrast ratio of 4.15:1 violates the required

[1] In October 2016, the European Parliament approved the directive 2016/2102 that requires websites and mobile applications of public sector bodies to conform to WCAG 2.0 Level AA. New websites must comply from September 23, 2019, old websites from September 23, 2020, and mobile applications from June 23, 2021, onwards.

[2] One device-independent point is defined by 1/160 inch.

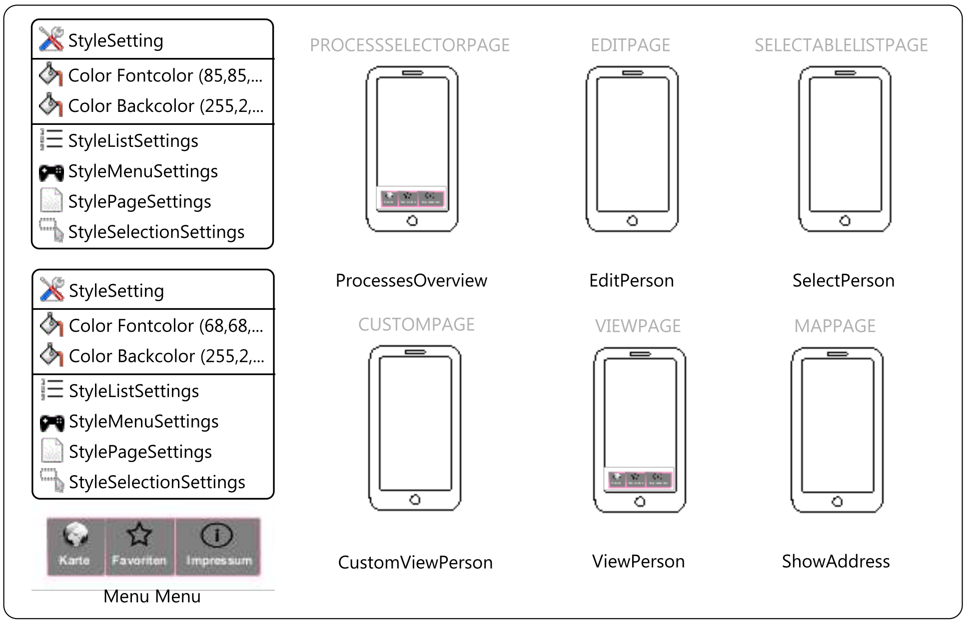
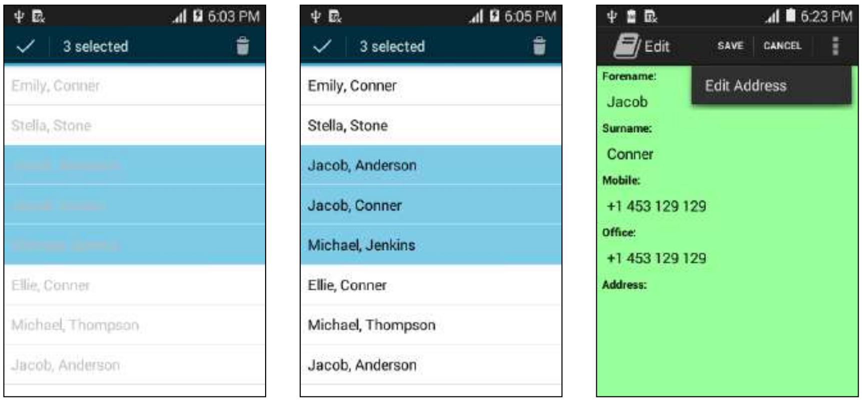


FIGURE 13.6: Style model of the simple phone book application



(A) Low contrast of selected list elements (exaggerated)

(B) Increased contrast due to the runtime style model

(C) General adaptation of style setting at runtime

FIGURE 13.7: Device-specific runtime adaptation of the mobile application's style

contrast ratio of 4.5:1. A style model can be used to fix the insufficient initialization of the mobile application and provide a sufficient color scheme at runtime. Figure 13.6 shows a style model with additional style settings which fixes the WCAG guideline violations and provides a more contrasted mobile application (Figure 13.7a and Figure 13.7b) by using the font color (`#444444`; RGB[68,68,68]) on certain pages (*ListablePages*). The layout shown in Figure 13.7a is not device compatible. Only the runtime adaptation shown in Figure 13.7b leads to a device compatible layout, i.e., style. Besides, the style model could also be used to configure the style of the mobile application in general (Figure 13.7c). □

13.2.1 Runtime Adaptation Implementation

13.22 The runtime adaptation of provider models, i.e., *process instance models* and *style models*, is implemented as follows: the generated mobile application attempts to load a *process instance model* or a *style model* at the start time. If the device's specific context adaptation is realized with device-specific process instantiation, a *process instance model* will be processed as already described in Section 12.2.1. If the device-specific context adaptation is realized with a device-specific style adaptation, then a *style model* holds device-specific styles for the pages of the generated mobile application. Every generated page (e.g., Activity in Android) evaluates the runtime information and changes the interface according the modeled information given in the style model.

13.3 Demonstration

The previously shown examples deal only with devices that are primarily designed for the Android operating system. However, an increasingly number of devices from other application domains also use this operating system nowadays. For example, Android is increasingly being used for TV boxes, cameras, game consoles, and eBook readers. An ongoing trend is the so-called *E-Ink Optimization* of mobile applications to enable them to run on eBooks. Major eBook vendors provide offline application installation or access different app stores for online installation. Hence, the following case example will show how the *word trainer* application can be made compatible with an eBook reader (e.g., Icarus E654BK Android 4.2.2). Please note that the modifications follow the one-fits-all concept. Particular, *one* mobile application is suitable for different devices (e.g., Tablet, eBook reader) based on runtime adaptation. Different device types (e.g., Smartphones and eBooks) can share the same app store, instead of using individual app stores. **13.23**

In order to evaluate the feature of device context support, we ask: how easy is it to adapt a real-world mobile application to a specific device? Since we already showed (cf. Table 13.1) that a few changes to the app model affect many files and thus lines of program code on the implementation level, we will not compare the modeling level and the implementation level in a quantitative way within the case example. However, an adaptation of the codebase by handwritten modification would not be effective compared to the model-based transformation approach. **13.24**

13.3.1 Case Example 6 (Word Trainer Application – eBook)

To make the *word trainer* application additionally compatible with an eBook reader, the processes and widgets of the graphical user interface which are affected by the desired device and their hardware facilities must be analyzed. Incompatible parts of the mobile application must be re-modeled (or *re-implemented* when applying a traditional development approach). Moreover, parts of the original application and the re-modeled parts must be instantiated depending on the used device type. **13.25**

Since eBook readers are very homogeneous in terms of hardware facilities, we assume that a standard Android-operated device (e.g., Android 4.2.2) has an E-Ink display (4-Bit Grayscale, 200-300 dpi, low refresh rate), touch screen, memory card or internal memory, Wi-Fi access, no camera, no microphone and no other sensors. **13.26**

13.3.1.1 Device-Specific Processes

Given this hardware setting, the processes *CreatePicture* (cf. Section B.3.3.3) and *CreateAudio* (cf. Section B.3.3.4) cannot be executed because an eBook reader offers neither a built-in camera nor a microphone is available on an eBook reader. Although these processes could simply be disabled to provide a variant of the mobile application for eBook readers, we are heading towards a more elegant solution. We would provide a variant of the mentioned processes that must be modeled manually by the application developer. The original process *CreatePicture* creates a **13.27**

Picture object and associates this object with a picture taken subsequently. Hence, the process returns a *Picture* object with a corresponding picture file. Since the eBook provides a file system in which picture files can be stored, a process variant for eBooks might select existing picture files and create associated *Picture* objects. Figure 13.8 shows the device-specific process variant for the eBook. Figure 13.9 shows the device-specific variant (eBook variant) of the *CreatePicture* process from the mobile user perspective. The process *CreateAudio* is re-modeled in an analogous way. The mobile application can now run on an eBook providing the re-modeled process is used.

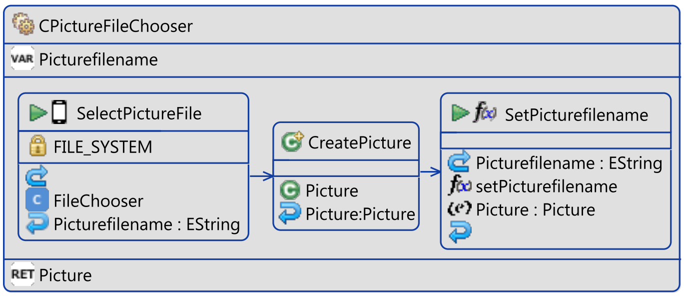


FIGURE 13.8: Device-specific variant of the process *CreatePicture*

(A) Main menu with device-specific processes (*SelectAudio/SelectPicture*)

(B) Device-specific implementation of the process *CreatePicture* (using a file selection dialog due to lack of a built-in camera)

FIGURE 13.9: eBook variant of the word trainer application

13.3.1.2 Device-Specific Style Configuration

13.28 However, the text inside the mobile application might be barely readable due to the style and color settings. Moreover, animations – i.e., the visual feedback to a mobile user while interacting with the mobile application (e.g., highlighting one or more selected list elements) – are hardly feasible since the E-Ink displays need several refresh cycles to change a pixel within a large contrast range (e.g., from black to white and vice versa)[3]. To circumvent these problems, the style of the mobile

[3] This problem is called *ghosting* because the so-called ghost pixels from prior content may remain on the screen.

application can be instantiated with a style that is optimized for an E-Ink display. Particularly, the style model uses only high-contrast values. Widget animations are avoided by using animation colors that are identical to the background. Figure 13.10 shows the effects of an E-Ink optimized style model in comparison to a non-optimized mobile application.

(A) Ghosting effect due to the changing focus of edit widgets

(B) Disabled selection animation while changing the focus of edit widgets

FIGURE 13.10: Using device-specific styles to avoid ghosting effects on eBooks

Chapter 14

System Contexts

The *system context* encompasses every context that a mobile application might be able to detect in the real world. While many system contexts can be used in an advantageous way, i.e., to realize smart behavior or innovative services, changing technical contexts (e.g., connectivity, energy supply) often cause problems and are currently handled insufficiently. While mobile devices and mobile applications could be used in numerous situations and locations, one serious disadvantage compared to traditional devices (e.g., stationary personal computers) is that physical mobility limits access to resources like the network or power supply. Hence, we will demonstrate in this chapter how mobile applications can deal with changing technical contexts, particularly with respect to network connection. **14.1**

The impact of an interrupted network link ranges widely in terms of its severity. Suppose the mobile application is used to transmit/receive a real-time audio- or video stream to/from another mobile client. In such a scenario, the broken network link makes the mobile application inoperable. Generally, real-time communication needs a network link with a low latency and sufficient bandwidth. However, when the network link is being used to access a remote database or invoke a processor-intensive remote service, a broken link might be tolerable. This is particularly true when the architecture of the mobile application can compensate for the interrupted network link by taking over the functionality of the service that could not be reached. For example, a mobile application can maintain local replicates of the application-specific data or compute processor-intensive tasks locally. Of course, there are drawbacks, such as conflicts of locally-performed transactions that might occur during data synchronization or bad performance of the mobile application due to the use of the local and limited processor facilities. Varshney and Vetter listed the network requirements of several kinds of mobile applications in [VV01] and described the effect of a broken network link. **14.2**

To construct a mobile application that is both online- and offline-capable, additional architectural components are needed for replication, synchronization, and local transaction management. If data can be modified locally, mobile applications shall avoid or prevent the execution of potentially conflicting transactions [Gra+96], and, thus, ensure conflict-free reintegration of modified data. Currently, mobile application developers have to acquire a lot of knowledge about online- and offline-capable data and transaction management before they can design such a mobile application. While initiating a mobile application development project, teams often have to make an architectural and platform choice [Puv+16] [Bre+14], which dominates or limits the development process thereafter. Forced to decide whether to realize an online-only or offline-only mobile application or an online- and offline-capable variant, mobile application developers must first thoroughly understand the use cases, i.e., the processes and the accessed data of the planned mobile application. Currently, there is no conceptual tooling for the evaluation of mobile application designs regarding their online and offline capability. Mobile application developers do not get any support to assess whether the effort to implement such an online- and offline-capable mobile application is reasonable, i.e., whether the number of successfully executable transactions (called throughput) increases considerably in **14.3** comparison to an online-only application variant. Moreover, the improper application of an offline-capable architecture can also drastically reduce the throughput. Besides, once a mobile application developer has decided to realize an online- and offline-capable mobile application, further issues emerge. For example, mobile information and transaction systems may support different consistency and conflict levels. Accepting lower conflict levels usually increases the throughput of transactions but may lead to data inconsistencies. Thus, mobile application developers are often unsure which conflict level should be favored, because the throughput for a certain conflict level is hard to predict.

14.4 In this chapter, we present four main contributions. First, we propose to model the core data structures and the behavior of a mobile application using a domain-specific modeling language. Based on this concise formulation of application data and behavior, a static conflict analysis can be carried out. This analysis identifies all potentially conflicting transactions, which must be managed in an online- and offline-capable mobile application. The model analysis shows the overall capability of a mobile application to be operated in an online and offline context.

14.5 Second, potentially conflicting processes may be tolerable in certain scenarios. For example, if a few mobile users act on a huge set of data objects, conflicts might occur rarely and could be tolerated. Using the results of the static conflict analysis and a simulation configuration, a model-based simulation system can predict the number of actual conflicts for an individual app model. Mobile application developers can test different (expected) operational conditions without implementing the planned mobile application beforehand.

14.6 Third, the design process allows the generation of prototypical online- and offline-capable applications. The goal of this software prototype generation is to provide an evolutionary prototype [Cri92] that can be studied and extended manually. The generated prototype follows the notion of horizontal prototyping [Bac+12, Sec. 4.3], i.e., specific layers of offline-capable applications are built for reuse. More precisely, the generated prototype focuses on the architectural components (e.g., application logic, replication and synchronization functionality) that are required [TG95] to realize online- and offline-capable applications, while, for example, the generated user interface is simple and may be replaced in further iterations. The generation step builds upon an existing framework for the model-driven development of mobile applications [Vau+18b] as presented in the first part of this thesis.

14.7 Fourth, the proposed design process is evaluated following different research questions. We will show how diverse the results of the analysis of different app models can be and how different contexts (e.g., the number of data objects, and mobile users) affect the throughput. Furthermore, we demonstrate the applicability of the design process by re-engineering a real-world application.

14.1 Design Process Overview

14.8 The starting point of our design process can vary: a new online- and offline-capable mobile application may be developed from scratch, or there may already be a mobile application that shall be made online- and offline-capable. In both cases, we want to find out under which conditions the introduction of online- and offline-capable data and transaction management is better than a pure online-only or offline-only solution. Figure 14.1 shows the main activities of our design process:

14.1.1 Modeling

14.9 The first step deals with the creation of an app model that describes the data structure and the behavior of the mobile application. It is sufficient to model

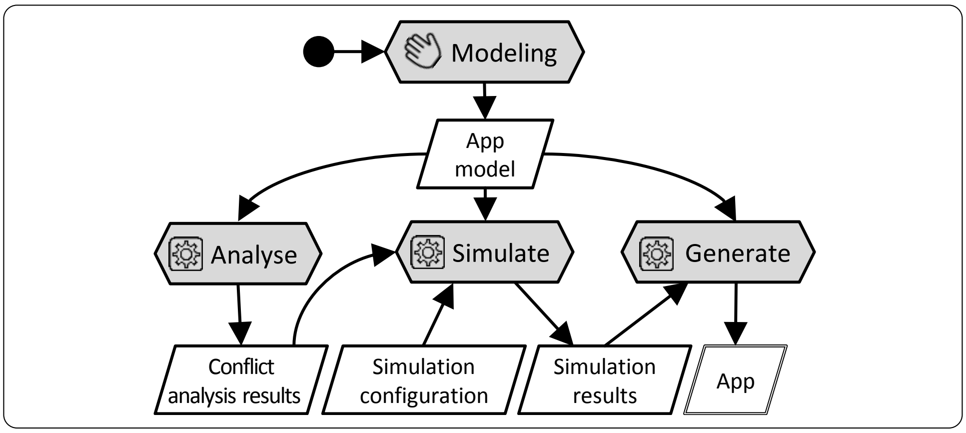


FIGURE 14.1: Design process of online- and offline-capable mobile applications

only that part that shall be available both online and offline. The app model follows the domain-specific modeling language presented in Chapter 6. A few new language features are introduced to facilitate the online and offline capability. The modeling step is carried out manually, but model editors support mobile application developers in creating models.

14.1.2 Model-Based Analysis

The first automated step of the design process takes the app model and identifies the conflicting processes using a static conflict analysis. An appropriate conflict definition will be introduced later. The result of this analysis is a conflict matrix, showing which processes are in conflict. Besides, this step delivers one or more conflict-free subsets of processes. **14.10**

14.1.3 Model-Based Simulation

Given the conflict-free subsets delivered by the analysis step, the simulation takes these subsets and determines how many transactions (instantiated processes) can be executed in certain usage scenarios. Mobile application developers mainly have to specify how many mobile users and data objects are expected in the existing or planned system. Finally, the simulation results denote the expected throughput for different subsets of processes and certain simulation conditions. **14.11**

14.1.4 Model-Driven Generation

Once mobile application developers have identified a beneficial design, they can generate a prototype of the online- and offline-capable mobile application. The code generator processes the app model to generate the basic functionality of the mobile application. The core components of the generated application, namely the data and application logic layer, the runtime part (enables/disables processes according to the connection state), and the local transaction manager, shall be reused because they are mature. The parts for replication and synchronization are also mature for mobile applications that process a medium-sized set of data objects. They may be manually optimized in case the mobile application processes a significant higher number of data objects. The user interface is robust but simple. It can be modified easily as the generated application separates the application logic and the user interface strictly. **14.12**

We will introduce an example to illustrate the used domain-specific modeling language as well as to exemplify the steps that are presented in the next sections.

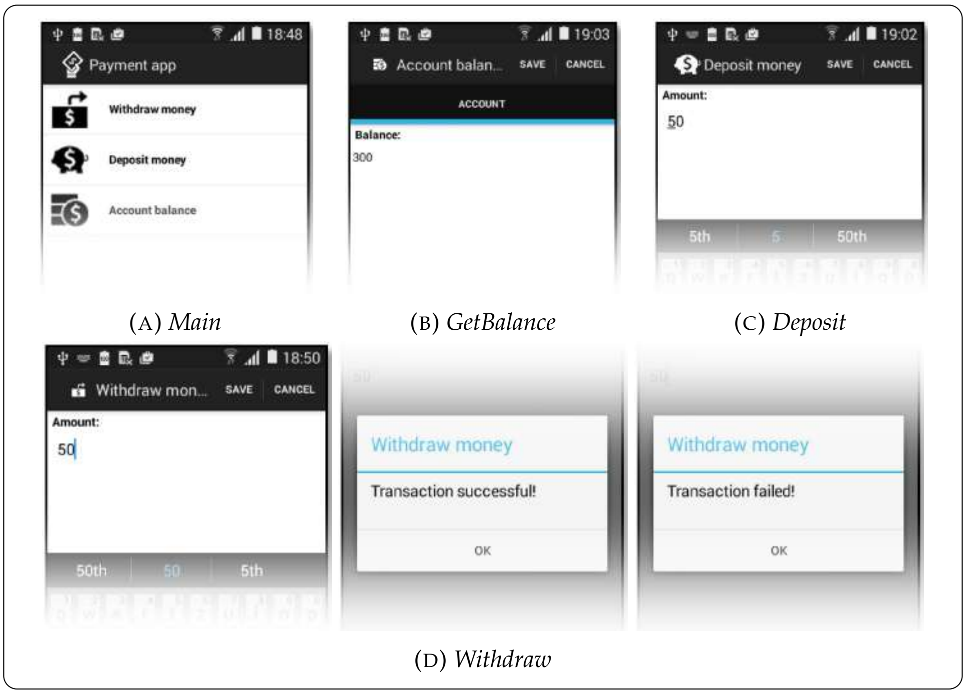
14.1.5 Running Example (Simple Payment Application)

14.13 Mobile applications that support mobile payments, such as Apple Pay [07], Google Wallet [14], PayPal [42], or in-app payment such as in-app Billing [13] do not yet support offline execution. The main obstacle may be the high-level of transaction security that is required by payment systems since conflicts can result in loss of money. Although no corresponding mobile application is yet available [SK12], the discussion about offline capability of mobile payment has started (read e.g., [29]). We choose mobile payment as running example since it illustrates very well the facilities for online- and offline-capable but conflict-free data and transaction management as offered by our design process. Note that our example is reduced to its core functionality. Hence, the data model and the application logic are kept rather simple. The generated mobile application shall function as a software prototype to explore if and under which conditions online- and offline-capable data and transaction management is useful for mobile payment.

14.14 **Example** (A simple payment application). A banking account is administered on a server at the banking site. It may be accessed by the server as well as from one or more clients. We assume to have a credit account, i.e., it is never in the debit state (which would give rise to a conflict).

FIGURE 14.2: App model of a simple payment application

14.15 This simple payment application is modeled as follows: the *data model* in Figure 14.2a mainly contains a class *Account* modeling a banking account. It comprises an attribute representing the current balance. It can be checked by using *getBalance()* and modified by operations *deposit()* and *withdraw()*. Both modifying operations need an amount that should be added to or subtracted from the balance. The operation *withdraw()* requires that sufficient funds are available to cover the required amount. Its return variable is set to *true* if covered and to *false* if not. The main process of the *process model* in Figure 14.2b is a process selector which refers to all the available processes. The user may invoke processes *GetBalance*, *Withdraw*, or *Deposit*. The process *GetBalance* invokes the operation *getBalance()* and displays the value. The process *Deposit* requests the amount of money that should be added and then invokes *deposit()*. The process *Withdraw* also requires the amount of money to be withdrawn, invokes *withdraw()*, and displays whether the transaction can be carried out or not. A simple graphical user interface can also be modeled and generated (not shown).



(A) *Main* (B) *GetBalance* (C) *Deposit*

(D) *Withdraw*

FIGURE 14.3: The simple payment application working online-only

The mobile application shown in Figure 14.3 is generated by the already existing code generators, presented in Chapter 7. The architecture of this mobile application follows a traditional client-server architecture, i.e., the mobile front end is not operable without a network connection. □ **14.16**

14.2 Model-Based Analysis

The purpose of the model analysis is to evaluate how many processes are potentially in conflict and which conflict-free configurations exist. For example, an online- and offline-capable mobile application requires replicated data objects on the mobile device which it is running on. The replicated data objects may be modified locally while the mobile device is disconnected. In parallel, another mobile client would do the same on its replicated data objects. This is a concurrent access to shared data objects, but traditional online conflict detection cannot be applied due to the network partition of the mobile clients [Ber+87]. Hence, the question arises: which conflicts may occur while all the mobile clients try to synchronize their modified copies and how can these conflicts be prevented or solved? Thus, we first introduce an appropriate conflict definition. Since this conflict definition is based on read and **14.17** write access to data objects, the data model must be equipped with corresponding annotations. Finally, the conflict analysis can run automatically; it delivers a conflict matrix as well as conflict-free subsets of processes.

14.2.1 Conflict Definition and Conflict Levels

14.18 Transactions that are performed offline on replicated data objects, must be *repeatable* on the primary copy (located at the server) in any order because the mobile clients will reconnect and synchronize in an arbitrary order. An offline transaction is repeatable only if it always returns the same value, i.e., fulfills the *return value commutativity*, and has the same effect, i.e., satisfies the *state commutativity*, when being re-executed on the primary copy [WV02] [Ouz+09]. This separates the set of transactions into potentially conflicting ones and conflict-free transactions. It should be noted that we consider only offline transactions on different mobile devices (users) to be synchronized with the server-sided copy, i.e., repeatability is not required for offline transactions performed on the same device.

14.19 To achieve a well-accepted online- and offline-capable mobile application, it seems to be more suitable (e.g., in order to increase the throughput) to use a fine-grained set of requirements for conflict-freeness in style of the ANSI/ISO SQL (Structured Query Language) [Ber+95] (cf. [Ady+00]) transaction isolation levels. This fine-grained conflict definition is beyond the state-of-the-art conflict definition for transaction systems, as it requires a more detailed description of data object access. In order to get a fine-grained conflict definition, we propose *weak* and *strict* versions of the *return value commutativity* and the *state commutativity*, respectively. Two offline transactions are called *weakly return value commutative* if their return values may differ when being re-processed on the primary copy. This happens if other clients have changed the primary copy in the interim period. Otherwise, they are called *strictly return value commutative*.

14.20 **Example** (Return value commutativity conflict). We assume that two clients access a banking account with an initial balance of \$50. One of them requests the account balance while being offline. The operation *getBalance()* delivers the last consistent value of *balance* being \$50. In the meantime, the other client deposits \$50 in an offline context. The operation *deposit()* returns no value and is therefore not affected by other transactions. But the return value of the operation *getBalance()* is not the same on the primary copy (assuming an arbitrary order of synchronization), because it may deliver \$100 instead of the locally returned value of \$50. Thus, there may be a return value commutativity conflict between the operations *getBalance()* and *deposit()*. □

14.21 Two offline transactions are *weakly state commutative* if the transformation of one state to another one may fail. This happens if other clients have changed the primary copy in the interim period and hence the execution condition of the failed transaction is no longer satisfied. They are *strictly state commutative* otherwise.

14.22 **Example** (State commutativity conflict). In this scenario, we assume that a bank account is accessed offline by a mobile client and online from an ATM (Automatic Teller Machine). Its initial balance is \$50. The client withdraws \$30 through the offline transaction *Withdraw* (e.g., transferring money via NFC). This transaction is performed on a local copy. It can be carried out locally because the paid amount is covered by the local balance of \$50. The local copy of the balance is \$20 now. In the meantime, the user withdraws \$30 from an ATM. This is an online transaction. It is checked whether the account is covered based on the primary copy (which is still \$50) and debits the requested amount. Finally, the primary copy has a value of \$20. While synchronizing the offline transaction, a conflict occurs because the amount has to be covered. In particular, the state condition ($balance - amount >= 0$)

is violated. Thus, there is a state commutativity conflict between the *Withdraw* processes. □

The resilience to conflicts is highly application-dependent. We define three relevant conflict levels to be able to define conflict strategies. The conflict strategy is a global property, i.e., all offline transactions have to follow the same conflict level: **14.23**

Level C1 (Conflict allowing) requires weak return value commutativity and weak state commutativity. **14.24**

Level C2 (Conflict avoiding) requires weak return value commutativity and strict state commutativity. **14.25**

Level C3 (Conflict prohibiting) requires strict return value commutativity. Note that weakness or strictness of state commutativity does not matter because state-changing operations, i.e., write operations, are not allowed requiring strict return values. **14.26**

Example (Conflict levels). Considering our running example, all processes fulfill C1. Level C2 dismisses the process *Withdraw* because the state condition is not fulfilled for this process. Hence, *Withdraw* is self-conflicting. Level C3 dismisses processes *Withdraw* and *Deposit* or *GetBalance*. □ **14.27**

If a mobile application has either read-only processes or processes with a write access albeit without a conditional read access, it may profit most from the connectivity awareness as its processes are conflict-free. All other mobile applications have to accept or resolve conflicts. We can identify different kinds of mobile applications for which the different conflict levels can be employed beneficially. The conflict allowing level C1 is recommended for mobile applications that realize an *information system*, where the information flow mostly is from the server to mobile clients. Conflict levels C2 and C3 are rather suitable for *transaction systems*, where the information flow is bidirectional between the server and mobile clients. However, the appropriateness of a conflict level is mostly determined by the number of users and data objects, i.e., by the conflicts that actually occur. **14.28**

Input of the analysis step: According to Figure 14.1, the input of the analysis step is an app model. Since conflicts are based on read or write access to shared data objects, such access has to be recognized by analyzing the modeled processes, namely process steps (called tasks). For example, processes may contain *CRUD* tasks that act on a specific data type. The model analysis can automatically recognize potential conflicts for all completely modeled tasks. For customized operations that are invoked inside modeled processes, mobile application modelers must explicate the data access of these operations manually by using additional language features. In the running example, 3 of 4 processes use customized operations. This is not usual because most data-oriented applications use rather standard tasks (e.g., CRUD) than custom functionality, as we will see in the evaluation. We chose this example to have the possibility of discussing the explication of data access as presented below. **14.29**

14.2.2 Explicating Data Object Access

In order to apply a conflict analysis, determining whether an operation is potentially involved in a conflict or not, all the operations must declare their data accesses. The following language features denote the data access of operations: **14.30**
i) An operation has a `Condition:Read` relation to an attribute or object if it reads this attribute or object as part of a condition.
ii) An operation has an `Action:Read` (`Action:Write`) relation to an attribute or object if it returns (writes) this attribute or object.

Example (Explicating data access of operations). Listing 14.1 shows the operation **14.31**

withdraw() of the class *Account*. This operation reads the attribute *balance* to check whether the account is covered.

LISTING 14.1: Operation *withdraw()*

```
public boolean withdraw(int amount){
  if ( balance - amount>=0) {
       Condition:Read
     balance = balance - amount;
     Action:Write
     return true; } else { return false; }}
```

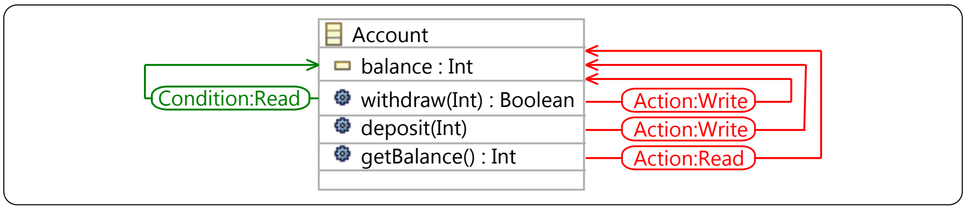


FIGURE 14.4: Access modes of the class *Account*

14.32 Hence, it has a `Condition:Read` relation to *balance*. It writes the attribute *balance* inside the body of the conditional statement, leading to an `Action:Write` relation to *balance*. Figure 14.4 shows the data access declarations for all the operations of *Account* by using newly introduced language features. □

14.33 Conflicting operations are identified as follows: two operations are C2-conflicting if one has a `Condition:Read` and the other one has an `Action:Write` relation to the same attribute. They are C3-conflicting if one has an `Action:Read` or a `Condition:Read` and the other one has an `Action:Write` relation to the same attribute.

14.2.3 Running the Model-Based Conflict Analysis

14.34 Based on the conflict definition, an app model equipped with read and write annotations can be analyzed. As shown in Figure 14.4, only the data model is equipped with annotations, but the whole app model (with process model) is taken into account. The tasks of all processes are analyzed for conflicts. In the second step, conflicting processes are identified based on their sets of conflicting tasks.

14.35 **Example** (Conflict analysis). For our payment application, the result of the conflict analysis is shown in Table 14.1. It is a conflict matrix where ✓ marks conflict-free processes and × marks conflicting ones for the corresponding levels.

TABLE 14.1: Conflict matrix (C1, C2, C3)

Processes	Withdraw	Deposit	GetBalance
Withdraw	(✓, ×, ×)	(✓, ×, ×)	(✓, ✓, ×)
Deposit	(✓, ×, ×)	(✓, ✓, ✓)	(✓, ✓, ×)
GetBalance	(✓, ✓, ×)	(✓, ✓, ×)	(✓, ✓, ✓)

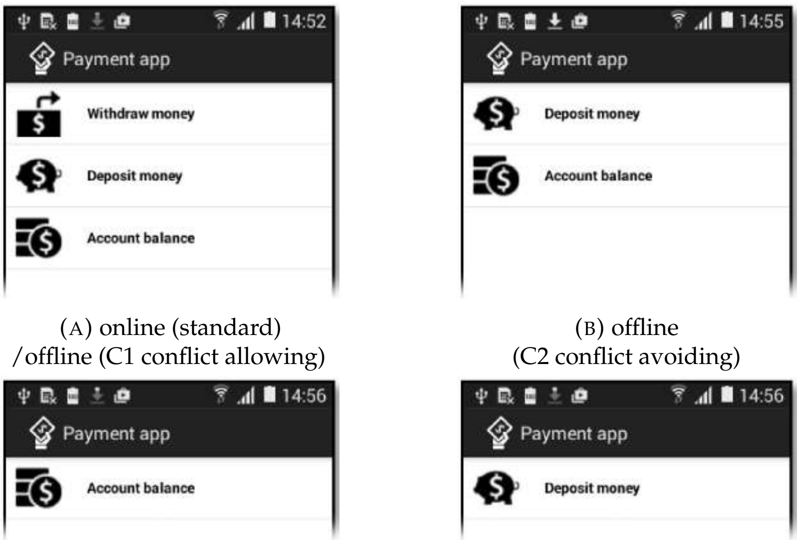
TABLE 14.2: C3 configuration variants

Processes	~~Withdraw~~	Deposit	GetBalance
~~Withdraw~~	~~×~~	~~×~~	~~×~~
Deposit	~~×~~	✓	×
GetBalance	~~×~~	×	✓

A conflict-free configuration can be found by removing (cf. Table 14.2 - crossed-out processes) i) self-conflicting processes and then ii) processes involved in a conflict until the set is conflict-free. Table 14.2 shows the removed self-conflicting process *Withdraw* (case i)). The remaining processes *Deposit* and *GetBalance* are still C3-conflicting. Hence, the inhibition of either the process *Deposit* or the process *GetBalance* delivers a conflict-free application variant. □ 14.36

Output of the analysis step: As shown in Table 14.1, the analysis step delivers a conflict matrix that shows, at first, all potentially conflicting processes. From this conflict matrix, a conflict-free configuration for a particular conflict level can be derived (cf. Table 14.2), which will be part of the input for the simulation step. 14.37

Example (Online- and Offline-capable payment application). Figure 14.5 shows the generated online- and offline-capable payment application that applies the results of the analysis step at runtime. 14.38



(A) online (standard) /offline (C1 conflict allowing)

(B) offline (C2 conflict avoiding)

(C) offline (C3 conflict prohibiting)

(D) offline (C3 conflict prohibiting) alternative configuration

FIGURE 14.5: The generated online- and offline-capable payment application

The mobile application provides the available processes at runtime according to the connection context and the selected conflict level. Choosing the conflict allowing level C1, all processes can be offered (as in online mode). Conflict level C2, however, requires to remove at least one conflicting process, here *Withdraw*. To reach conflict level C3, even two processes have to be removed to prohibit conflicts, here *Withdraw* and *Deposit* or *Withdraw* and *GetBalance*. We recommend the conflict-avoiding level C2 for the payment application. It ensures that the balance is always funded because money cannot be withdrawn without checking that sufficient funds are available. In turn, mobile users can check the balance and deposit money while being offline. □ 14.39

However, the conflict level is not only driven by the requirements of an application (w.r.t. conflict-freeness), but it is also dependent on the number of actual conflicts occurring in the system. Mobile application developers may be interested in understanding how the throughput improves if such conflicts were accepted. Hence, a model-based simulator can support mobile application developers by estimating the throughput for different conflict levels. 14.40

14.3 Model-Based Simulation

14.41 As Tables 14.1 and 14.2 show, the conflict analysis delivers conflict-free subsets of processes by discarding conflicting processes (cf. Figure 14.5). The inhibition of conflicting processes is effective, but the question that may arise is: how many conflicts will there be if potentially conflicting processes are not inhibited? For example, an online-only architecture might be generally non-operable in an offline context, but an online- and offline-capable mobile application might be too restrictive and hence non-operable due to process-starvation.

14.42 The actual throughput of mobile applications depends on several factors such as the number of mobile users and data objects. Further factors are the activity, connectivity, and behavior of mobile users. The structure of the data objects (e.g., loosely coupled or strongly related) also affects the throughput. We observed that these runtime factors can considerably influence the number of actual conflicts both positively and negatively. For example, an app model with high conflict potential, i.e., many potentially conflicting processes, might be uncritical in the case that only a few users are involved who even show low activity. In turn, an app model with less conflict potential might become critical in the case that a lot of highly active users act on a small set of shared data objects.

14.3.1 Dynamic Conflict Analysis by Simulation

14.43 To better support mobile application developers in evaluating the results of the static conflict analysis, the proposed design process contains a simulation step. Based on the app model and different sets of processes, mobile application developers can predict and compare the throughput of their mobile application, based on a dynamic conflict analysis.

14.44 **Input of the simulation step:** A mobile application developer designs an app model equipped with access declarations (if needed). The simulation system additional requires a conflict-free configuration, determined by the prior analysis step (cf. Table 14.2). Furthermore, values for all the independent simulation variables have to be set. They are shown in Table 14.3.

TABLE 14.3: Independent simulation variables

Name	Description	Domain
#Users	Number of mobile clients.	1 .. 10,000
Activity	Average activity level of users.	0 .. 100
Connectivity	Average connectivity level of users.	0 .. 100
App model		DSML
#Objects	Number of objects or aggregate size for each class.	1..5,000 or custom instance

14.45 *#Users* denotes the number of simulated mobile clients. *Activity* denotes the average activity level of all users. A level of 50 means that, on average, half the users are active in each iteration. *Connectivity* denotes the average connectivity of all users, again in each iteration. By setting the number of objects (*#Objects*) of each class, the simulation system constructs a corresponding instance of the data model used in the simulation. Mobile application developers can also load an existing instance of the data model in order to simulate on the basis of data taken from a running system. This is indicated by the phrase “custom instance” in Table 14.3.

14.46 **Output of the simulation step:** In turn, the simulation system delivers the so-called dependent simulation variables as output, which is shown in Table 14.4. The number of transactions processed by Client i is given by *#Processed_Client_i*. The

TABLE 14.4: Dependent simulation variables

Name	Description
#Processed_Client_i	Number of transactions processed by Client i.
Throughput	$\sum_{i=1}^{\#Users}$ #Processed_Client_i

overall system throughput is the sum of these values. These dependent simulation variables are determined for all conflict levels.

14.3.2 Running the Model-Based Simulation

Based on the app model and the input parameters, i.e., the independent variables, the simulation system determines the throughput of the desired mobile application. With the exception of the app model, not all input parameters must be set. For example, in order to investigate how the throughput varies due to different connectivity levels of the mobile clients, this parameter may be left uninitialized. The simulation system selects values for the uninitialized parameters according to their corresponding domain (cf. Table 14.3). By default, every run passes 10,000 simulation iterations to get stable results. **14.47**

Example (Predicting the throughput of the online- and offline-capable payment application). We apply the simulation system to the model of our simple payment application shown in Figures 14.2 and 14.5. Figure 14.6 shows the results of the simulation with 500 mobile clients and 100 accounts. The results are normalized with respect to the maximum theoretical throughput. The *Online* graph represents the throughput for the online-only architecture while using a traditional transaction model and conflict definition (cf. Weikum und Vossen [WV02, Def. 2.2]). At the best connection level (100%), the overall system reaches a throughput of 50%. The throughput is not higher than that due to the occurrence of conflicts. It decreases as the average connection level of the clients (users) declines. At the worst (0% connectivity), there is no throughput (0%). The *Offline (C1/C2/C2 MTM/C3)* graphs show the throughput at different conflict levels. **14.48**

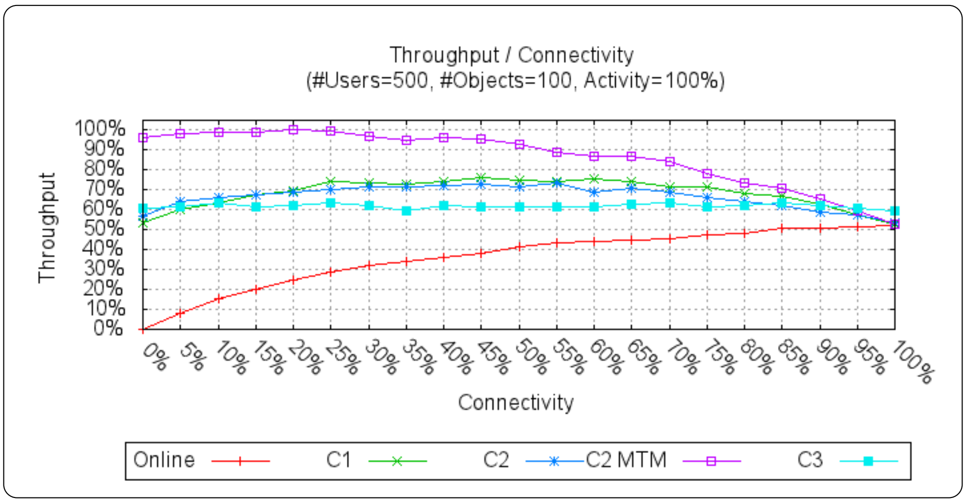


FIGURE 14.6: Throughput of the payment application (Configuration 1)

While even the most restrictive level C3 provides a higher throughput than the online variant, the best variant is actually shown by graph C2 MTM which denotes level C2 using mobile transaction models (cf. Section 9.4.2.4) in addition. This variant is explained in the next section. The throughputs of the C1, C2, and C3 variants are almost equal for the simulated configuration. **14.49**

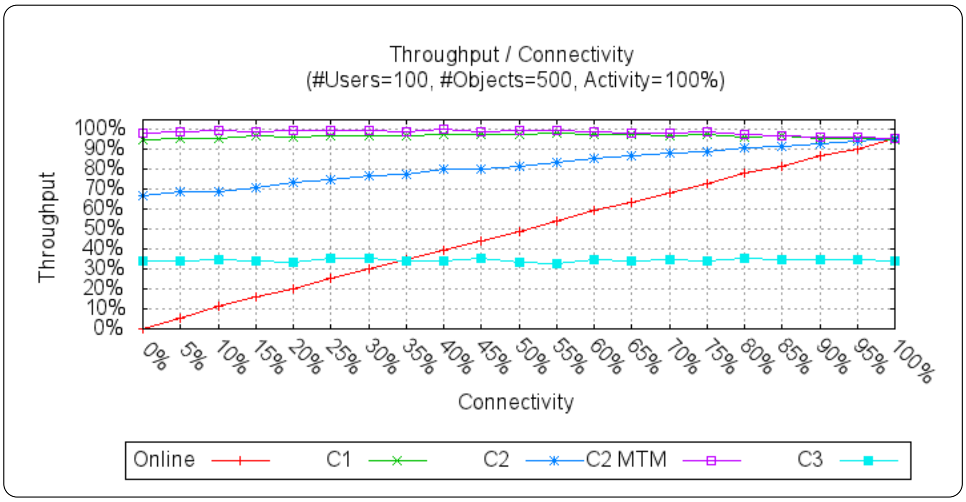


FIGURE 14.7: Throughput of the payment application (Configuration 2)

14.50 Another tested configuration, shown in Figure 14.7, demonstrates that a conflict-prohibiting conflict level (C3) drastically decreases the throughput in case that conflicts occur only rarely. In this case, it is advisable to select the conflict-allowing level C1. Besides, the use of a mobile transaction model will provide only marginal improvements. □

14.3.3 Restricting Data Object Access

14.51 Considering the conflict-avoiding level C2, processes are discarded if they do not preserve the state conditions of other processes. Hence, online and offline capability can heavily restrict the behavior of mobile applications. It is possible to improve this situation by introducing mobile transaction models (MTM) that allow potentially conflicting processes if they do not operate on the same data. To this end, the collection of potentially concurrently accessed objects may be split in such a way that the conflicting processes work on disjointed sets of data (Keypool method [54] [56] [55]). Furthermore, object attributes containing single numeric values (called aggregates) may be fragmented to preserve a global state condition, e.g., not negative balances of a banking account (Escrow method [O'N86] [LL09]). For example, an *Integer* value might be an aggregate that can be split into fragments. Depending on whether mobile application developers declare objects as summable objects (aggregates) or individual objects (collections), the corresponding mobile transaction models are used in the simulation as well as in the generated application. The simulation and generation automatically select an appropriate mobile transaction model, i.e., the Escrow method is used if attributes are declared as aggregates, while the Keypool method is used when collections are declared as online- and offline-capable. Although the Keypool and the Escrow methods are independent approaches, our simulation and generation approach supports the combination of these methods in a single architecture. Similarly to the read and write access introduced in Section 14.2.2, the data model can be equipped with annotations that declare the previously mentioned restrictions of data accesses. As a result, processes also become available for the conflict level C2. Although mobile application users can utilize these additionally gained processes, they are limited in terms of argument values or data objects. While the online-only variant of a process can act on the whole set of a collection or an aggregate, the online- and offline-capable variant of this process is limited to a subset of a collection or a fragment of an aggregate.

14.52 **Example** (Application of mobile transaction models). Figure 14.8 shows a snippet of an annotated data model for our simple payment application. The annotation

<<ca>> (connectivity-aware) of the attribute *balance* indicates that this object may be split and allocated to mobile clients. Thus, mobile clients may share *Account* objects, particularly its attribute *balance*, but only in fragments. The parameter *#AllocObjects* denotes the number of objects/fragments that should remain at the server (i.e., not distributed among clients).

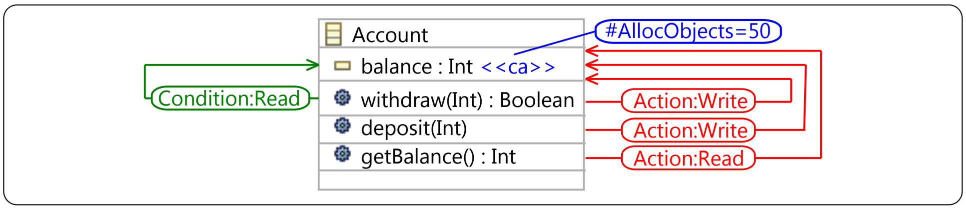


FIGURE 14.8: Access modes and restrictions of the class *Account*

In the generated online- and offline-capable payment application that uses mobile transaction models, the process *Withdraw* is now also available at conflict level C2. However, the process *Withdraw* is limited: assume that there are five mobile clients sharing one account and the online value of *balance* is $300. According to the *#AllocObjects* annotation, $50 remains at the server, while the remaining amount of $250 is distributed among the clients applying the Escrow method. Hence, every client can withdraw $50 while being offline. Graph C2 MTM in Figure 14.6 shows the simulation results for using a mobile transaction model. Its throughput is above those of all other variants; hence, this conflict level is optimal. □ **14.53**

14.3.4 Design and Implementation of the Simulation System

The purpose of the simulation system is to predict how many conflicts would occur if the designed application was implemented. Thus, the simulation system generates a set of mobile users, a functional implementation of the data model and process model, and a conforming set of data objects. During a simulation run, mobile users execute processes (transactions) generated by a customizable transaction generator. The simulation system logs every transaction (even if they fail) that is executed offline or online and finally provides a global history. This history is analyzed and the throughput is computed. For an example, see Figures 14.6 and 14.7. **14.54**

Finally, the Eclipse plugin *pimar.simulation.modelbased* implements the simulation system. The simulation system reuses the model-based analysis form the plugin project *pimar.generator.frontend.android* (cf. Table 14.5). **14.55**

TABLE 14.5: Fifth set of plugins shaping the MDD infrastructure

Plugin project name	**Content**	**Type**
pimar.simulation.modelbased	Simulation system	manual
pimar.generator.frontend.android	Model analysis	manual

To conclude, the simulation system ca be used to decide if an online- and offline-capable architecture is preferable to purely online- or offline-capable architectures. Moreover, the simulation system shows which throughput could be achieved by accepting different conflict levels. Finally, the simulation system predicts the throughput for using mobile transaction models. **14.56**

14.4 Model-Driven Generation

While the first steps of the design process focus on the analysis of an application **14.57** design based on its online and offline capability, the final step is applied to generate a software prototype of the designed mobile application. However, the results of the analysis and the simulation considerably affect the architecture of the prototype that is generated. The framework used is, in its present form, able to generate native online-only or offline-only applications, with a traditional rich-client or client-server architecture respectively (cf. Figure 7.3). A client-server design should be favored if the predicted throughput for such an online-only architecture is higher than for the online- and offline-capable architecture. In turn, the existing code generator for Android was extended – one of our contributions – to produce mobile applications with an online- and offline-capable architecture (cf. Figure 9.5).

14.58 **Input of the generation step:** The generation step requires an app model. To generate an online- and offline-capable mobile application, the app model must be equipped with annotations that declare the data access (cf. Section 14.2.2) and, optionally, restrict it (cf. Section 14.3.3). The simulation results are not directly processed by the generator because it seems more appropriate to let the mobile application developers decide the conflict level to be applied in the resulting prototype. They can select between Online, C1, C2 (MTM), and C3.

14.59 **Output of the generation step:** The output of the generation step is a native mobile Android application that either has a simple client-server/rich-client architecture or an online- and offline-capable architecture.

14.60 The model-driven generation completely conceals the architectural design from mobile applications developers. Mobile application developers control the resulting architecture only implicitly by the modeled processes, the annotations they made inside the app model, and the selected conflict level. For example, an app model that comprises solely read-only operations results in a mobile application architecture without components for transaction logging or synchronization because no data objects will be changed or needed to be synchronized.

14.4.1 Introducing Online and Offline Capability

14.61 Given an app model as presented in Section 14.1.5, the code generator produces a connectivity-aware Android application with a generic extended client-server architecture (see client *GEC* in Figure 9.5). Hence, the existing generator [Vau+18b] was extended to generate applications containing a local TM and a local database. The following functions were realized based on the working model of the TM (cf. Figure 9.8):

14.62 *Initial setup*: Given the data model, the model-object mapping framework Teneo [71] and the object-relational mapping framework Hibernate [LM10] allow setting up a relational data base scheme and to persist model instances to a server-located database.

14.63 *Online transaction processing*: Since Hibernate is a certified Java Persistence API (JPA) Provider [KS13] [53], it includes transaction session management that can be reused to handle the online transaction processing.

14.64 *Replication*: Data replication is realized by loading the model instances from the database (server), detaching them from the online session, and storing them locally on the mobile devices.

14.65 *Offline transaction processing*: Offline transaction processing follows the selected conflict level. All transactions are logged for later synchronization and will be executed on the server when the mobile devices are online again.

14.66 *Synchronization and reintegration*: Since we use a data model that contains the definition of operations in terms of attached platform-specific program code, the mobile applications can realize a transaction-based synchronization (cf. [Shi+12]). By logging the object identifiers of accessed objects and performed operations (including parameters) in the offline context, a transaction can be repeated on the server in the online mode.

Finally, the Eclipse plugin *pimar.generator.frontend.android* is modified in order to generate online- and offline-capable mobile applications. However, mobile application developers can also still use the generator without this functionality. **14.67**

To sum up, model-driven generation allows the creation of online- and offline-capable mobile applications. By understanding the concepts presented here, mobile application developers can generate online- and offline-capable applications that might serve as prototypes for further development. **14.68**

14.5 Evaluation

In this section, we aim to answer the following question: is our design process for online- and offline-capable mobile applications useful and applicable? To address this research question, we investigate two sub-questions: given the app model of an existing or planned app, does our design process help to evaluate it with respect to online and offline capability? (RQ1) Can mobile application developers easily apply our design process to create software prototypes with an online- and offline-capable architecture? (RQ2) **14.69**

14.5.1 Usefulness of the Design Process (RQ1)

As claimed previously, the design process helps mobile application developers to evaluate any given app model regarding the online and offline capability of a remodeled or planned mobile application. It should be considered that both static and dynamic conflict analysis may reveal that some mobile applications may not be suitable to operate online and offline. Thus, the design process could be considered as useful if suitable and unsuitable app models can be identified. **14.70**

TABLE 14.6: Key data of considered mobile applications

Application name	Model elements	Processes	Classes	LOC
Conference Guide	168	15	8	35137
Word Trainer	843	24	15	43076
Phone Book	142	9	3	16241
App Shop	161	9	9	19691
Air quality application	118	2	7	13348

We will discuss analysis and simulation results of selected app models. Table 14.6 shows the key data (i.e., the number of model elements and lines of code) of five data-oriented mobile applications that are considered within this evaluation. Three mobile applications have already been published, namely the phone book [45] (cf. Section B.1), conference guide [44] (cf. Section B.2), and word trainer [43] (cf. Section B.3). The app shop and the air quality application have been newly developed. All these mobile applications have been modeled, analyzed, simulated, and generated. These app models contain only standard tasks, except the AppShop model, which includes custom operations. The app models, generated code and simulation results are available at [46]. **14.71**

The static conflict analysis (cf. Table 14.7) determines the available processes according to the required conflict level. Only the app shop has processes with `Condition:Read` annotations in such a way that two more processes can become active by using mobile transaction models (denoted in column C2 MTM). The other mobile applications perform non-conditional read and write operations only. **14.72**

TABLE 14.7: Analysis results of considered mobile applications

Application name	Available processes			
	C1	C2	C2 MTM	C3
Conference Guide	15	15	-	7
Word Trainer	24	24	-	11
Phone Book	9	9	-	5
App Shop	9	7	9	2
Air quality application	2	2	-	2

TABLE 14.8: Simulation results of considered mobile applications

Application name	Average improvement of the throughput (compared to online-only)				#Users	#Objects
	C1	C2	C2 MTM	C3		
Conference Guide	49.27%	49.66%	-	51.31%	500	390
Word Trainer	50.01%	50.79%	-	4.15%	300	2601
Phone Book	51.49%	51.41%	-	29.58%	40	40
App Shop	50.58%	34.19%	40.46%	1.12%	200	10609
Air quality application	54.96%	54.96%	-	54.96%	50	118

14.73 Table 14.8 shows the simulation results of the considered mobile applications. *#Users* and *#Objects* denote the number of simulated users and used objects, respectively. Now, we want to know how the throughput may be improved (or worsened) by using an online- and offline-capable architecture at a specific conflict level compared to the online-only variant. These differences are considered for all levels of average connectivity and then summed up. The average of this sum is shown in the columns below *Average improvement*.

14.74 The considered mobile applications show an improvement of the throughput at the conflict level C1. The reason for this is that both the replication and transaction-based synchronization generally provide a higher throughput compared to a traditional transaction model. Concerning the conference guide, the throughput can be further improved by selecting the conflict level C3, because seven among 15 processes are conflict-prohibiting and conflicts will occur frequently (500 users act on 390 objects). In turn, the app shop has a low improvement of about 1% at the level C3. This improvement is so small, as only two processes are available at the C3 level. The execution of all other processes is not successful. In contrast, the throughput of the online-only variant is pretty high, since the ratio of users to objects is about 53. However, the application of mobile transaction models for the App Shop shows an improvement compared to the conflict avoiding level C2.

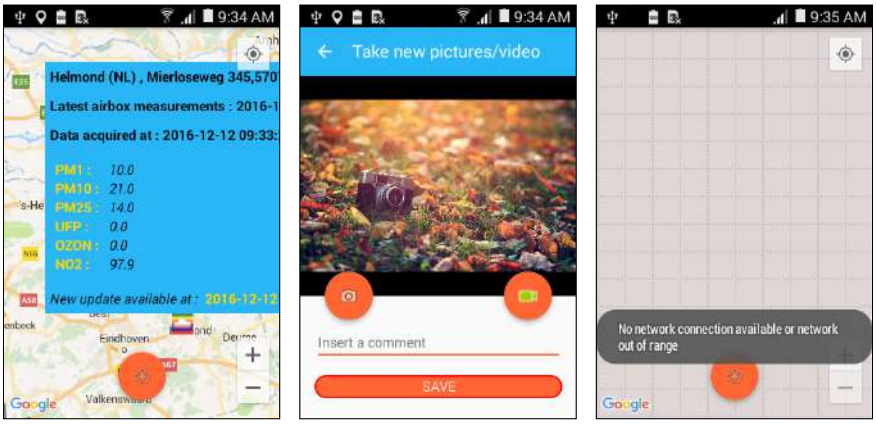
14.75 The simulation of the presented and other mobile applications leads us to the realization that the static analysis – the ratio of conflict-free and conflicting processes – is a non-robust indicator of the expected throughput. Only a dynamic simulation helps to determine whether a mobile application could be used advantageously in an online and offline context and what the thresholds are in terms of the number of users and objects.

14.5.2 Usability of the Design Process (RQ2)

14.76 The usability of the design process is evaluated using a real-world development project. This includes both an observation as to how a mobile application developer applies the proposed model-driven design process as opposed to a traditional development approach, and how powerful the software prototype will be in terms of online- and offline capability. The goal of the demonstration is to show that the proposed design process requires less effort (i.e., development time) to obtain better results, i.e., mobile application with a higher transactional throughput.

14.5.2.1 Case Example 7 (Air Quality Application)

This case example presents the model-driven re-engineering of an air quality ap- **14.77**
plication (cf. Figure 14.9) developed by the *Technical University of Eindhoven (TU/e)* in Netherlands and the *Istituto di Informatica e Telematica* in Italy. The air quality application provides real-time air pollution data for the cities Eindhoven, Breda, and Helmond. The mobile application relies on the Aireas network[1]. The Aireas network provides measurements of *particulate matter emission*, *ozone*, *nitrogendioxide*, *temperature*, and *humidity*. This data is made available via a JSON-based (JavaScript Object Notation) application programming interface (API) in real time. The Aireas measurement infrastructure forms the back-end service of the air quality app. Mobile application users can get real-time information (see Figure 14.9a) of the air pollution in their surroundings. In addition, mobile application users can share their comments, pictures, or videos on critical air quality values (see Figure 14.9b) to provide additional information to the user community. Hence, *WeSense* is not only an environmental information system but rather a social platform that helps to create cities that are more attractive and healthier, appealing to individual responsibility and collaboration (cf. Jean-Paul Close et al. [Ham+16]). *WeSense* could rather be classified as information system (cf. Section 7.1.1) than a transaction system (cf. Section 7.1.2), because the air pollution data is static, and the user comments can always be added conflict-free. Concurrent modification of data does not occur in this mobile application.



(A) Air pollution data (B) Comment function (C) Offline context

FIGURE 14.9: Air quality application *WeSense*

In its current version, the mobile application has some limitations. Due to the **14.78**
external Aireas measurement infrastructure, the architectural design of the mobile application requires that it stay permanently online to acquire the latest measurement values. Although the measurement infrastructure provides only a 10-minute interval as the shortest update period, the application cannot operate temporarily offline during these periods and remains stuck if the network connection is interrupted (cf. Figure 14.9c). Acquired and displayed data is not cached during a session and will be lost after disposing of an activity. Besides, the mobile application does not save the data permanently on the device. All data is lost every time a user terminates the mobile application. Hence, at first, the re-engineered air quality application should be online- and offline-capable and, by this, provide a

[1] www.aireas.com/welcome-to-aireas/

higher throughput of transactions. Second, the re-engineering of an already existing application additionally provides the opportunity to compare concurrently used development approaches, i.e. the model-driven development approach vs. the traditional development approach.

Model-Driven Implementation

14.79 According to the modeling approach, the model-driven implementation requires an app model comprising a data-, process-, and GUI model. The app model of the air quality application is similar to the app models shown before. The developed app model requires no additional model elements to realize the online- and offline-capable application features because all of the required architectural features will be realized by the code generator. Hence, app models developed with the model-driven development infrastructure might be used to create (i) only online, (ii) only offline, and (iii) online- and offline-capable mobile applications. Solely the configuration of the code generator or the configuration set at the first start-up of the mobile application (cf. Section 7.3.3.1) defines the instantiation of the different mobile application variants. The model-driven re-engineered version of the air quality application provides the use cases *show air pollution data on map* and *create comment*, which are comparable to their implementations in the original applications.

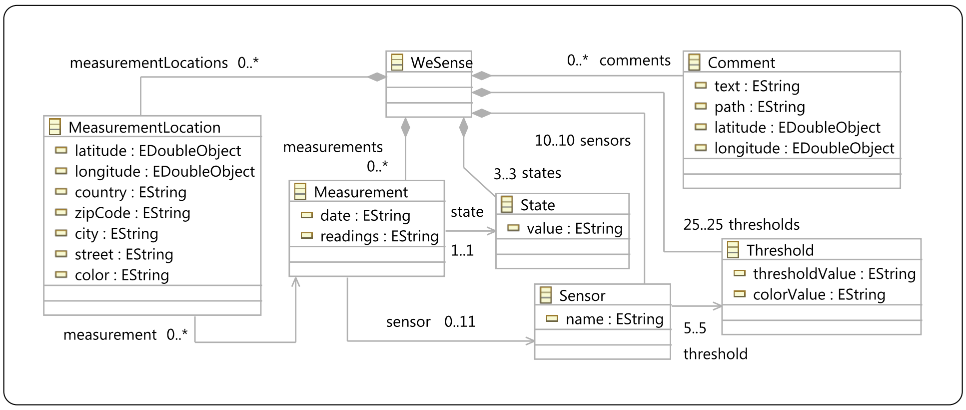
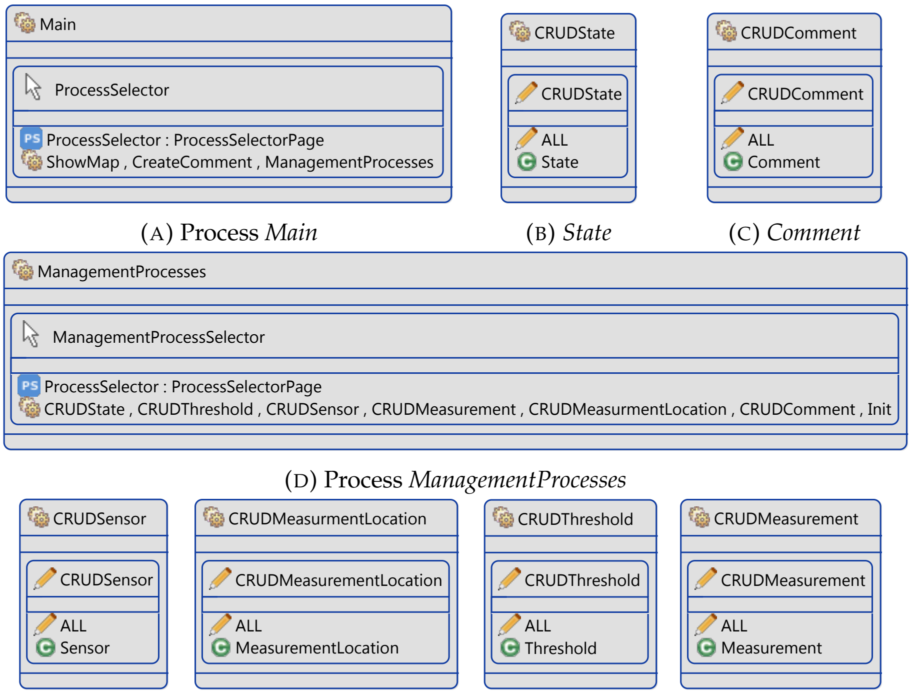


FIGURE 14.10: Data model of the air quality application (without class methods)

14.80 **Data model**: The data model was derived from the data format provided by the JSON-based application programming interface (API). The data model (see Figure 14.10) consists of the five main entities *Measurement*, *MeasurementLocation*, *State*, *Sensor*, and *Threshold*. Each measurement is attached to a measurement location. Measurements are ordered by date and contain the air quality values, characterized by the attribute readings. They make use of several sensors to arrange the measured values. Each sensor value is associated with a set of thresholds. If a threshold value has been exceeded, a specific color code is set as the attribute color in the entity *MeasurementLocation*. Each measurement location is associated with GPS coordinates characterized by the attribute's longitude and latitude and suitable address data. The entity *Comment* is used for storing user comments containing the GPS position data, a text value, and the path of an optional photo. User comments are displayed along with the air pollution data on the map.

14.81 **Process model**: The obligatory *Main* process (cf. Figure 14.11a) refers to three processes namely *ShowMap* (cf. Figure 14.12), *CreateComment* (cf. Figure 14.13), and *ManagementProcesses* (cf. Figure 14.11d). While the first two processes (i.e., *ShowMap* and *CreateComment*) are implementations of the given use cases, the third process *ManagementProcesses* might be used only by administrators to manage the application data. The process *ManagementProcesses* contains several CRUD processes (cf. Figures 14.11b, 14.11c, 14.11e, 14.11f, 14.11g, and 14.11h), providing the creation, modification, and deletion of objects (e.g., *MeasurementLocation*, *Measurement*, *State*, *Sensor*, *Threshold*, and *Comment*). Additionally, the process *Init* (cf. Figure 14.14) provides a hook mechanism to acquire data from an existing custom back end. Such a process is only necessary if a custom back end should be used rather than the automatically generated default back end.



(A) Process *Main* (B) *State* (C) *Comment*

(D) Process *ManagementProcesses*

(E) *Sensor* (F) *MeasurementLocation* (G) *Threshold* (H) *Measurement*

FIGURE 14.11: Air quality application process structure and CRUD processes

The process *ShowMap* (cf. Figure 14.12) first gathers the current spatial position **14.82**
of the device by invoking the sub-process *GetPositionAndColor* (not shown). The second task of the process *ShowMap* uses the sub-process *GetAllPositionsAndColors* (not shown) to retrieve the measurements of the measurement locations including coloring to visualize the level of air pollution. Subsequently, the sub-process *GetUserPositionCommentAndPath* delivers all user comments including the geographic position where they are created. The next three tasks are being used to join the results of the prior steps in a comprehensive list of data. Finally, the task *ShowGoogleMap* invokes a MapPage that renders the collected data within the previously processed tasks.

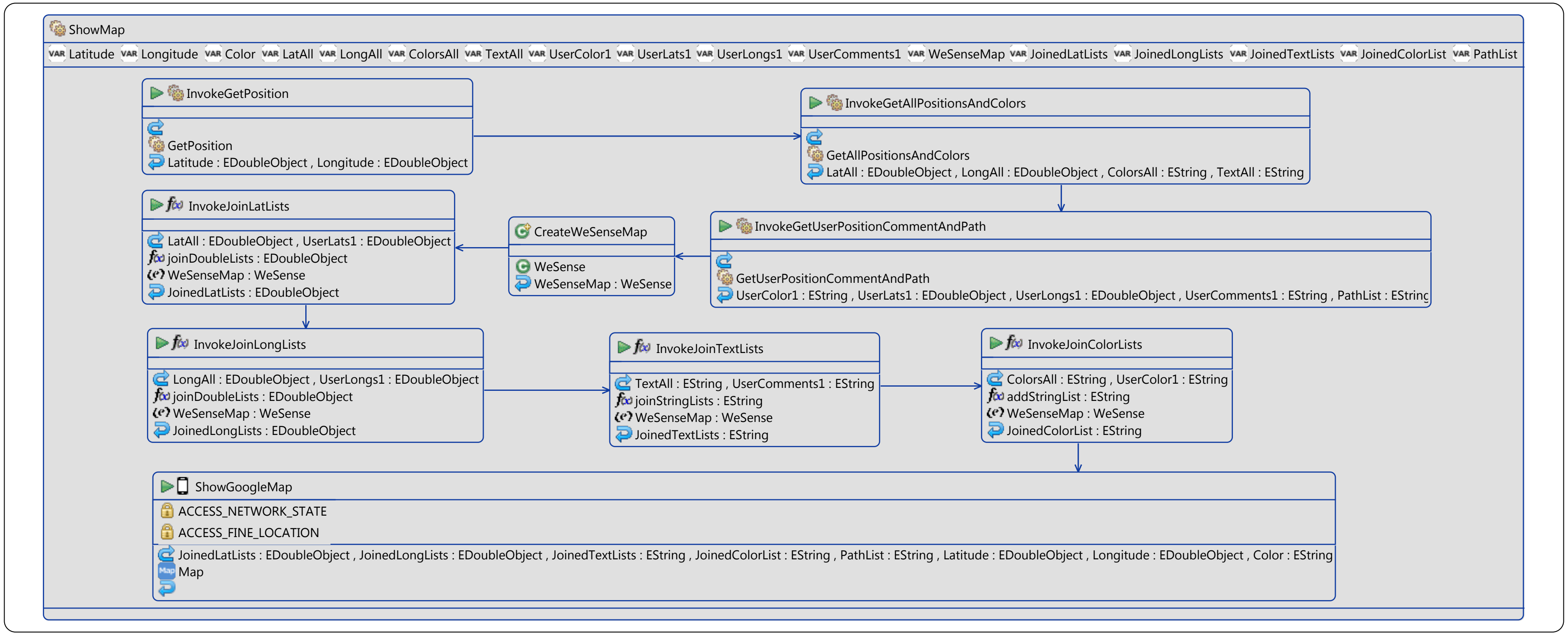


FIGURE 14.12: Process *ShowMap*

The process *CreateComment* (cf. Figure 14.13) provides the creation of a comment including the current geographic position and a photo. The first task of the process acquires the current spatial position of the device via the sub-process *GetPosition*. The following task invokes a *TakePicturePage*, which takes a picture and returns the picture path and file name under which the picture was saved. A new comment is created. Then an *EditPage* for setting the user comment appears and the user is allowed to enter the text of the comment. Finally, a task of the type *InvokeOperation* joins the geographic position, the path information, and the comment text in a comprehensible comment object, which will be stored on the mobile device. **14.83**

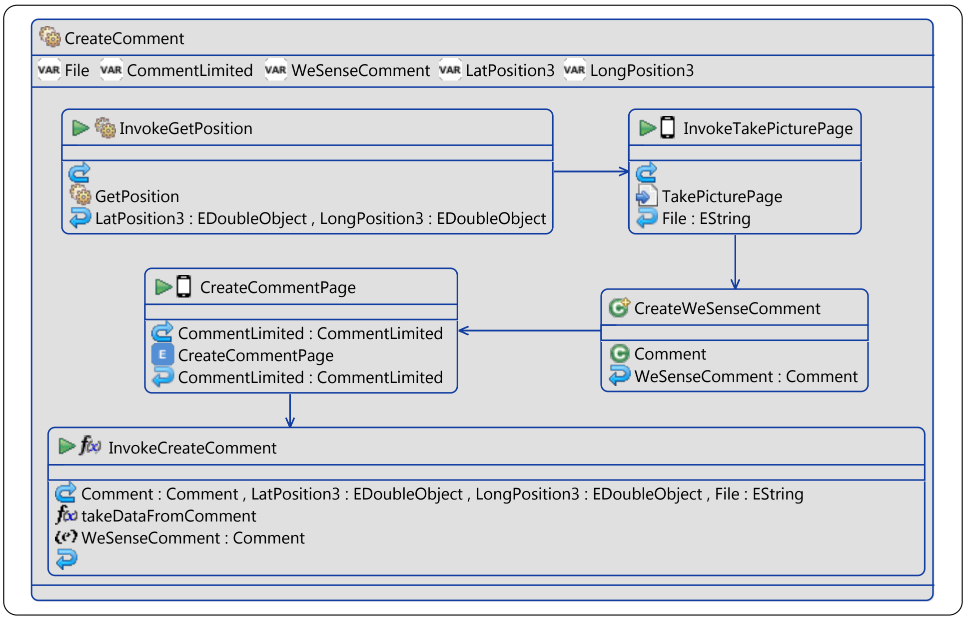


FIGURE 14.13: Process *CreateComment*

The process *Init* (cf. Figure 14.14) is used to initialize the data of the air quality application. The first task invokes the method *init* of the data model, which, in turn, uses an external library to access the back-end services. This external library contains a handwritten JSON-to-EMF adapter. This adapter acquires and converts the data from the back end to model instances used internally by the generated mobile applications. Thus, the generated mobile application can deal with various back-end systems, as well as a default EMF-based back-end infrastructure. The subsequent tasks of the *Init* process show a notification after the successful replication and conversion of data. **14.84**

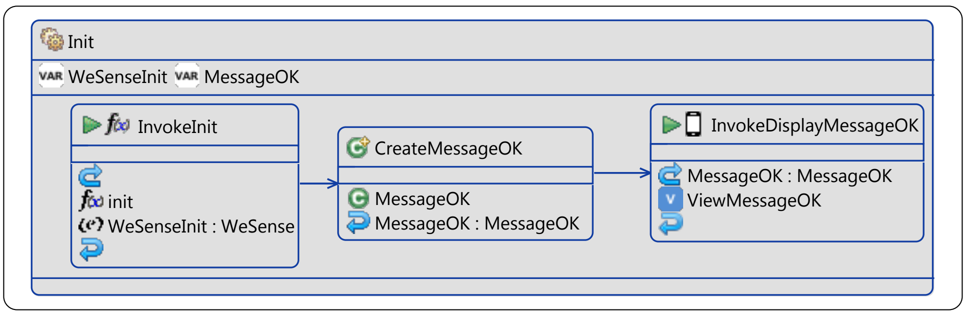


FIGURE 14.14: Process *Init*

GUI model: Finally, the GUI model (cf. Figure 14.15) defines the pages and styles that are used inside the air quality app. The *ProcessSelector* will be used by the processes *Main* (cf. Figure 14.11a) and *ManagementProcesses* (cf. Figure 14.11d). **14.85**

The *Map* is referred to by the process *ShowMap* (cf. Figure 14.12). The process *CreateComment* (cf. Figure 14.13) uses the pages *TakePicture* and *CreateComment*. Finally, the page *ViewMessageOK* displays the result of the process *Init* (cf. Figure 14.14).

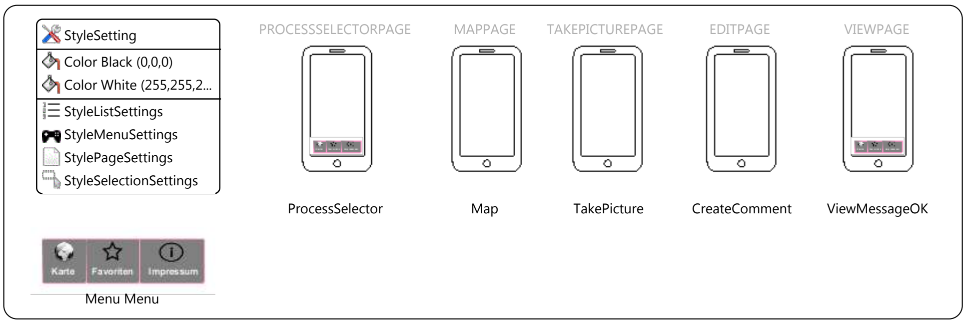


FIGURE 14.15: Pages and style of the air quality application (without attributes)

14.86 **Back end interoperability and compatibility:** As described earlier in this chapter, the model-driven development infrastructure aids in the generation of both a front end (mobile application) and a compatible default back end (cf. Figure 1.4). However, since the presented air quality application uses the Aireas platform as an already existing back end, the re-engineered mobile application should be compatible to this back end. Hence, the re-engineered air quality application exploits the presented mechanism to inject custom code into the mobile application developed in a model-driven way (cf. Section 7.3.5.2).

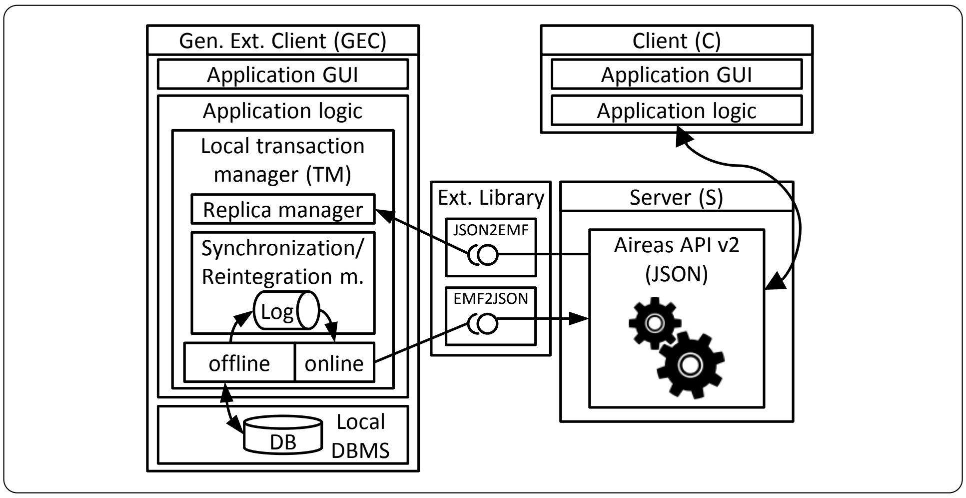
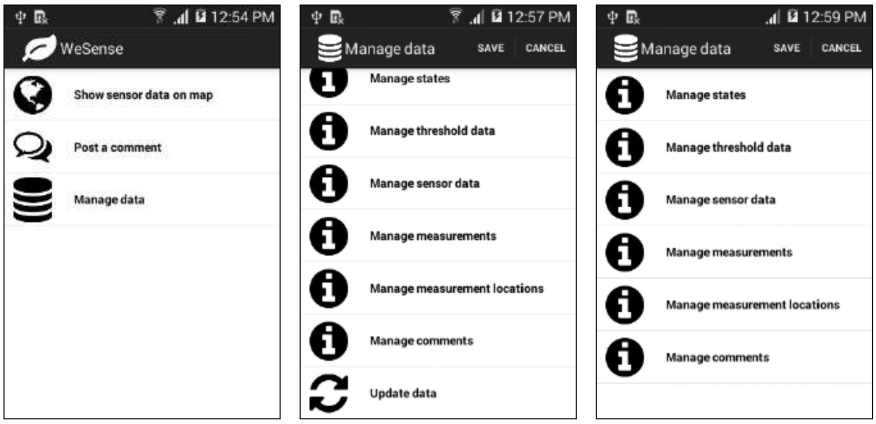


FIGURE 14.16: Generic extended client-server architecture with custom back end

14.87 This external library bridges the Aireas platform, which is using JSON, and the generated mobile application, which uses EMF. The external library creates and updates the runtime instance of the data model with the data from the custom back end. Figure 14.16 shows the architecture and the interaction of the re-engineered air quality application. The re-engineered air quality application, represented by the generic extended mobile client *GEC* (left-hand side), provides additional architectural components to store data and synchronize transactions. The existing air quality application (mobile client *C* right upper corner) might further operate unaffected by the unmodified back-end system.

Online- and Offline-capable Air Quality Application

14.88 Based on the presented app model a code generator generates an online- and offline-capable mobile application. Figure 14.17a shows the main screen of the mobile application (cf. its model in Figure 14.11a). The mobile application was initially started in an online context, recognizable by the Wi-Fi-symbol (ᯤ). The mobile application automatically triggers the process *Init* every time the mobile application starts. The local instance data model will be created during the first start-up, or else it will be updated. An update can also be triggered manually by the menu item *Update data* (cf. Figure 14.17b) that triggers the process *Init* (cf. its model in Figure 14.14). The process *Update data* is context-sensitive and will not be available in an offline context (cf. Figure 14.17c).



(A) Main menu of the air quality application

(B) Management processes (online)

(C) Management processes (offline)

FIGURE 14.17: Re-engineered air quality application (Part I/II)

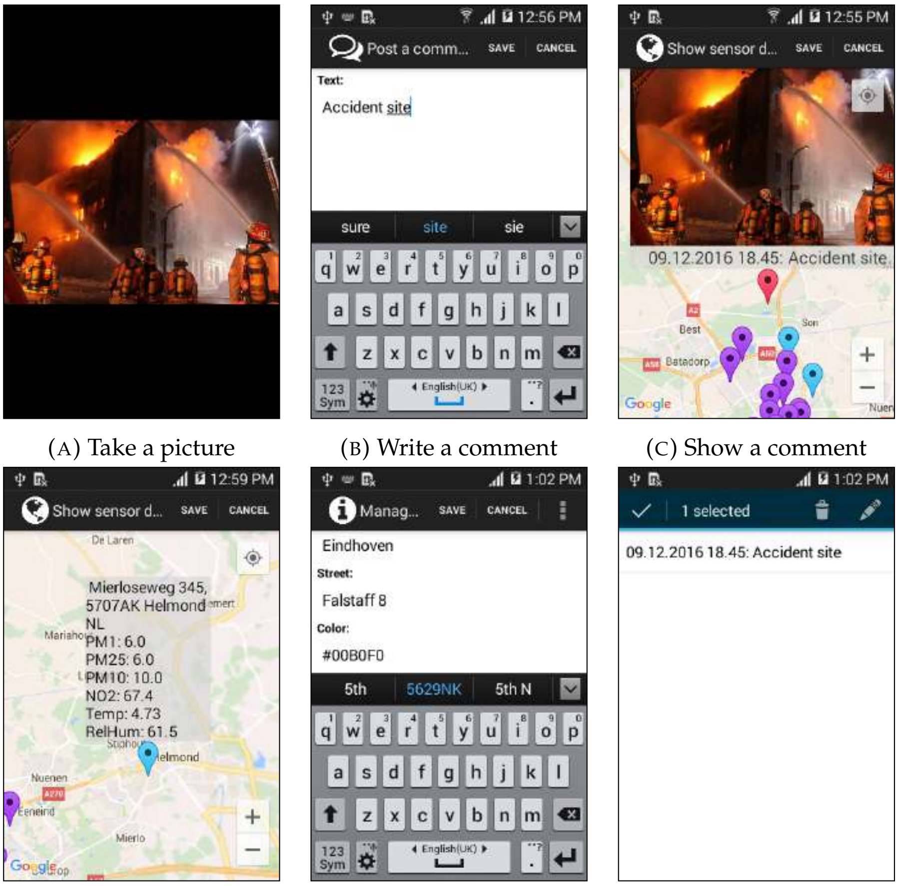
Figures 14.18a, 14.18b, and 14.18c show the use case *Post a comment* (cf. its model in **14.89**
Figure 14.13). Assuming the mobile application is now disconnected, Figure 14.18d shows the use case *show air pollution data on map*. It does not matter if the connection is interrupted while the application runs or the mobile application starts without a network connection. Finally, Figures 14.18e and 14.18f show the selected CRUD operations (cf. the model shown in Figures 14.11f and 14.11c respectively). Having demonstrated that the re-engineered application can work both online and offline, we will go on to show how this online- and offline-capable architecture affects the overall throughput at different connection situations and how beneficial such an architecture could be.

Comparing the Online-only and the Online-and-Offline Solution

When comparing the versions of the air quality application, we are interested in **14.90**
the transactional throughput (number of successful transactions) that they can achieve. The presented simulation system (cf. Section 14.3) was used to predict the transactional throughput of an online-only architecture (as implemented by the original air quality app) and an online- and offline-capable architecture (as implemented by the re-engineered air quality application). The simulation considers different stages of device connectivity.

Figure 14.19 shows the simulation results, i.e., the transactional throughput of the **14.91**
application versions. The simulation considers 50 mobile application users, which are fully active (100%). They operate on a set of 118 data objects. This simulation data was taken from the already running back-end system.

The first plot (*Online*) shows the throughput of the online-only architecture. As **14.92**
in almost all mobile applications that implement an online-only architecture, the throughput increases when the connectivity increases. Mobile clients with low



(A) Take a picture (B) Write a comment (C) Show a comment

(D) Show measurement values (offline) (E) Edit measurement location (offline) (F) Delete a comment (offline)

FIGURE 14.18: Re-engineered air quality application (Part II/II)

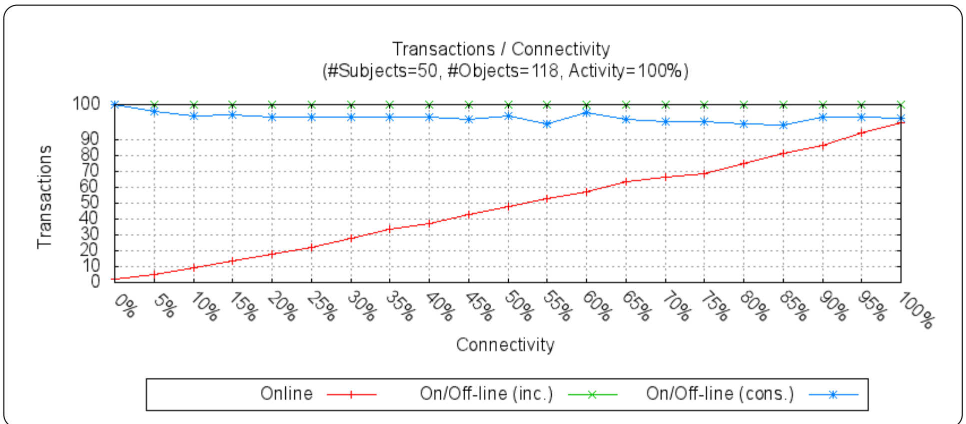


FIGURE 14.19: Transactional throughput of the application versions

bandwidth or connectivity cannot commit many transactions because they cannot reach the primary copies located at the back end. A better connectivity usually enables a higher throughput, but it may also lead to higher conflict rates. Hence, an excellent connection of the mobile device does not necessarily lead to a 100% success rate of the performed transactions.

14.93 The second plot (*On/Offline (inconsistent)*) shows the throughput of an online- and offline-capable architecture such as implemented by the re-engineered mobile application. Due to the read-only character of the air quality application (except the create comment use case) it is not necessary to synchronize the locally performed transaction with the back end because shared data records will never be modified by the mobile clients. Only comments must be reintegrated at the back end, but they are typically just added, i.e., cannot be edited in a concurrent manner. Operating with read-only transactions on a local copy of the data enables a success rate of 100%. Weakly connected or disconnected mobile clients can increase the throughput considerably compared to the online-only architecture. However, one may argue that the locally performed transaction might show outdated, i.e., inconsistent values due to changes performed intermediately on the primary copies.

Hence the third plot (*On-/Offline (consistent)*) shows the simulation results while considering strictly consistent return values. Locally performed transactions which deliver inconsistent return values were not counted as performed successfully. Consequently, a difference of 5% on average between the consistently and inconsistently performed transactions could be recognized. In the case that the mobile application contains several modifying processes, the gap between consistently and inconsistently performed transactions might be higher. **14.94**

Comparing the Development Approaches

Furthermore, we compared the development effort using the proposed model-driven development approach to the development effort when using a traditional development approach. The re-engineered mobile application was developed by a graduate assistant. The assistant had already gained knowledge by using the model-driven development infrastructure before. He spent 128 hours on this project, which included the review of the existing mobile air quality application, the back end, including its application programming interface (8 hours), the construction of the back-end adapter (cf. center of Figure 14.16) library (24 hours), the modeling of the data-, process-, and GUI models (56 hours), the indoor and outdoor tests of the mobile application (16 hours), and the corresponding documentation (8 hours). Besides, the project needs some minor changes and extensions on the model-driven development infrastructure (e.g., showing pictures on the map page), which needed an effort of 16 hours. Compared with the effort of using a traditional development approach by *TU/e* and the *Istituto di Informatica e Telematica*, our model-driven development approach requires less development effort and provides more features at the same time.[2] **14.95**

14.5.3 Threats to Validity

An external threat to validity is our choice of case examples: the result of our studies may not be generalized in the context of other cases. We attempted to mitigate this threat by choosing at least three real-life applications from different application domains – the conference guide, word trainer, and the air quality application WeSense. Moreover, we have not considered large app models, but both simulation and prototype generation scale well, as tested with artificially generated models (e.g., up to 7500 processes). Since simulations and generation are performed at development time, these activities are not too performance-critical. **14.96**

Our focus is on improving the throughput of transactions. However, we have to admit that replication of data may deteriorate it. In most cases, the standard replication is sufficient, for example, for mobile applications with read-only processes. For other cases (e.g., location-aware data), a more efficient replication strategy should **14.97**

[2] Since the development times in the original project were not logged, we compare the times based on the size of the codebase. The original online-only version of the mobile application is a result of about 14.788 LOC. We assume that the creation of such a codebase needs much more time than 128 hours.

be considered. Since the architecture of a generated prototype application is highly modular, components may be exchanged by more suitable ones to overcome this threat.

14.98 Finally, we have to admit that we have not evaluated the validity of simulation results in practice, except in our case study. We are convinced that they are valid due to the following reasons: the simulation system executes exactly the generated code during simulation since parts of the simulation system are generated from the app model. Hence, the mobile application is likely to behave exactly in the same manner as in the simulation. The used simulation data and parameters, however, may deviate from the runtime environment. In that case, the throughput will deviate from the predicted one. To deal with this threat, the deployed application can be simulated again using real-world datasets and parameters. Besides, comparing the simulation with generated data objects to the simulation with data objects taken from existing applications has shown little deviation of the throughput.

14.99 Sargent's [Sar14] guidelines for verifying and validating simulation models, which states that the real-world entity (e.g., generated mobile applications), the conceptual model, and the implemented simulation system must be consistent in order to get accurate simulation results. Our simulation system meets these requirements. The only drawback is the *data validity* since the simulation data may deviate from the data used in the runtime environment. However, Sargent also states that data which is used only for experiments (e.g., simulation runs) and not for the simulation model construction can be omitted during validation process of a simulation model.

Chapter 15

Related Work: Model-Driven Development of Context-Aware Mobile Applications

In the following sections, the approaches and software engineering toolsets for context-aware software development are presented and compared with our model-driven development approach and its context support facilities. Since we have already discussed the work related to the model-driven development part of mobile application engineering in Chapter 8, this chapter will focus on the work related to context-aware mobile software systems. **15.1**

Since there is a wide variety of work related to context-aware software development, we cluster them into three groups, and, finally discuss only the last group in more detail. The first cluster contains the middleware-based context-aware systems. These are state of the art for the development of traditional context-aware systems. The second cluster consists of several model-driven techniques and methods for the development of context-aware systems that are proposed in the literature. Finally, in the last cluster, the available frameworks for the model-driven development of context-aware systems are presented and discussed. **15.2**

15.1 Middleware-Based Context-Aware Systems

Early context-aware mobile applications realized a direct sensor access, mostly to physical sensors, but not limited to those ([IS03, Sec. 2]). Consequently, the processing of sensor data, context recognition, and context processing was implemented inside a single mobile application. However, mobile applications with such architecture are very specific to a software platform or a device. Hence, they could not be easily installed on other devices or platforms. Besides, recurring functionality, which deals with context processing, could not be reused in other context-aware mobile applications. Thus, the first architectural improvement was the introduction of a middleware. As shown in Figure 15.1, the abstract layered architecture [Ail+02] [Dey+01] for context-aware applications uses a *sensor* layer to access physical or virtual sensors. This layer is device- as well as platform-specific. The *raw data retrieval* layer acquires sensor data. The acquired data can be very complex (e.g., a series of records) and extensive. The next layer, i.e., the *preprocessing/reasoning* layer, is responsible for reasoning and interpreting raw data. This layer might aggregate or join data from different sensors to recognize more complex contexts. The fourth layer – *storage and management* – saves the recognized contexts (sensor fusion), provides querying functionality, and sends the events to the subscribing application(s). This layer acts as a public programming interface that can be accessed by top-level applications with a *programming toolkit*. An essential component of the *storage and management* layer is the *context model*. This is needed to define and store context data. Different competing approaches, such as ontology and graphical models, **15.3** are discussed [SLP04] [Bet+10] as the methodology for context modeling. We will see later that the elements of the context model can be mapped to the elements of an app model to specify the context-aware functionality. A distributed variant of such a middleware framework is broker architecture [Che04], which is also called a *context server*. Broker architecture is a centralized service that gathers sensor information from several mobile devices, allows performance-intensive data analysis, and provides remote sensors. Hence, contexts that occur through the interaction of different mobile devices and mobile users could be detected with such architecture.

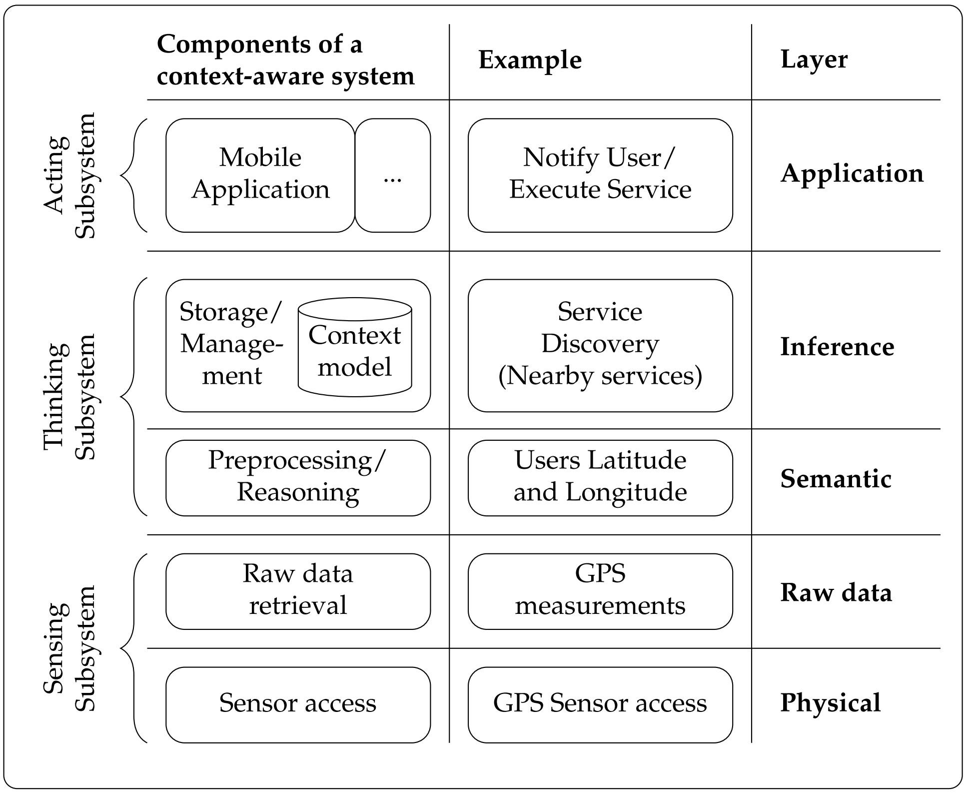


FIGURE 15.1: The abstract layered architecture for context-aware applications

15.4 Some of the most popular context-aware frameworks that follow the layered architecture before mentioned are the *Context Managing Framework* [Kor+03], the *Service-Oriented Context Aware Middleware – SOCAM* [Gu+04], the *Context-Awareness Sub-Structure – CASS* [FC04], the *Context Broker Architecture – CoBrA* [Che+03], the *Context Toolkit* [Sal+99] [DA00] [Dey01], the *Hydrogen Approach* [Hof+03], the *CORTEX* system which base on the *Sentient Object Model* [BC04], and the *Gaia* project [Rom+02] [74]. Hong et al. [Hon+09] give a more comprehensive overview of context-aware frameworks and applications.

15.5 A drawback of the middleware approach is that the *applications* layer is out of the methodological focus. More precisely, it is up to the mobile application developers to specify the actions that a mobile application shall perform if the middleware detects a particular context. For example, the detection of an offline context might be very simple, but the adequate adaptation of the mobile application needs a lot of changes at the application layer. Another drawback is that the middleware approach focuses merely on runtime contexts. Many contexts, such as the platform or the device type, are known at the design time. Hence, contexts are also relevant at design time, which is often out of focus of the mentioned frameworks.

15.6 The middleware-based architecture is useful if a complex context can be recognized at runtime, and only minor runtime adaptations of the application layer are required.

Hence, the focus of the middleware-based approaches is on context recognition rather than on context processing, which is a part of the acting subsystem. Since the work presented in the previous chapters focuses on the *acting subsystem*, i.e., the mobile application, we will move to the techniques and methods for context-aware system design which are more focused on the application layer.

15.2 MDD Techniques for Context-Aware System Design

In the following section, the existing model-driven development techniques for the design to context-aware systems are presented. These approaches are mostly general and independent to a specific domain. Moreover, these methods are often not implemented or tool-supported because they have a conceptual character. We will position our approach after each of the presented approaches: **15.7**

Vale and Hammoudi [VH08] proposed a model-driven development method that exploits parameterized model transformations. The parameterization of model transformation rules promotes the reusability, adaptability, and interoperability of model artifacts within a context-aware application development process. At first, a mobile application modeler must annotate a platform independent model (PIM) with *markers* that declare which types are context-dependent. A parameterized transformation rule gets an annotated *PIM* and *context argument*s. The transformation rule binds the given *context arguments* to the annotated *PIM*, particularly the marked variability point. The authors called this step *contextualization*. The output of the parameterized transformation is a *Context PIM* (CPIM). Depending on the number of the provided context arguments, several CPIMs can be created. Subsequently, the CPIMs will be translated to platform specific models (PSMs[1]). The approach provides context-awareness at the design time. The *contextualization* and the subsequent generation of an application code provide a set of applications that are tailored to different contexts. The specification of a context-aware application happens at a very low level since only the types of modeling elements can be *contextualized*. **15.8**

Our approach is very similar to the parameterized model transformation approach but has a larger scope in terms of the affected modeling levels (e.g., PIM, PSM, and Code). As shown in Figure 15.2 (left-hand side), the approach proposed by Vale and Hammoudi focuses mainly on endogenous model transformation at the level of the PIM. In turn, our approach provides *contextualization* both by an endogenous model to model transformation (cf. Section 13.1) and by an exogenous model to code transformation (cf. Section 12.1), as well as through runtime-contextualization. Hence, our approach provides *contextualization* at every stage of development, even at runtime. **15.9**

Ayed et al. [Aye+07b] (cf. also [Aye+07a]) presented a model-driven development approach for context-aware applications. Mobile application developers specify several models (context model, context collection model, application variability model, and application adaptation model) based on the UML and the UML profiles. The final step of their approach requires the definition of the target platform and model-to-model transformations, which are used to translate these models to program code of the desired platform. In contrast to the approach of Vale and Hammoudi, the approach of Ayed et al. focuses on runtime adaptations of applications instead of design time adaptation. The specification of application variability supports structural, architectural, and behavioral adaptation. This variability is **15.10**

[1] The authors named a transformed CPIM a CPSM instead of a PSM, but their approach provides no contextual arguments while transforming the CPIM to a PSM.

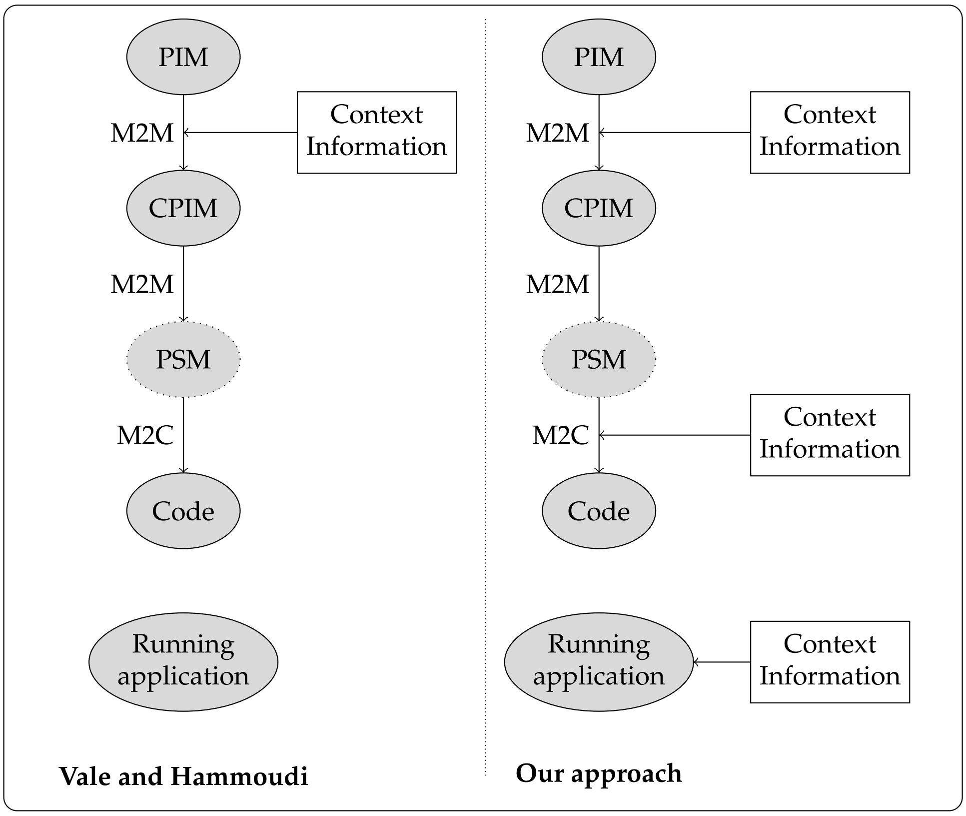


FIGURE 15.2: Comparison of the approaches

supported by overriding functionality through the inheritance of objects, the optional instantiation of classes (class diagrams), and the context-sensitive execution of functionality modeled in sequence diagrams.

15.11 Similar to the approach of Ayed et al., our approach supports runtime adaptability. However, our approach is limited in terms of the supported contextual dimensions. The approach of Ayed et al. requires mobile application developers to specify both the context collection and the context adaptation for potentially any contextual dimension, as well as the model-to-model transformation required for the implementation of this contextual awareness. In contrast, our approach provides an implicit context-collection and context adaptation mechanism for the mentioned contextual dimensions. Additionally, our approach considers contexts that are known at design time.

15.12 Ou et al. [Ou+06] (cf. also Georgalas et al. [Geo+07]) proposed a model-driven integration architecture for ontology-based context modeling and context-aware application (CAA) development. They proposed to extend the development process of the OMG's Model-Driven Architecture by a *Context Ontology Model* (COM). The authors named the resulting architecture *Model-Driven Integration Architecture* (MDIA). This architecture contains a *Process metamodel*, a *Component metamodel*, a *GUI metamodel*, and a *Data metamodel* for application specification. The *Context Ontology Model* defines context ontology data in the architecture. The essential *Integration metamodel* defines the way all these models associate and integrate. More metamodels – for example, *J2MEMetaModel*, *CSharpMetaModel*, and *XMLMetaModel* – are used to generate technology-specific context-aware applications. Model transformations provide the transformation of multiple technology-neutral metamodels into a single technology-specific metamodel. Similar to the approach of Ayed et al., runtime adaptation of mobile applications is in the focus of this approach.

15.13 Similar to our approach, Ou et al. provides several metamodels for the specification of a mobile application. Additionally, some contexts, such as the platform context, are implicitly processed at the design time by applying the corresponding platform-specific transformation rules. The reuse of the existing context ontologies by context ontology models is beyond our approach, since we deal only with a limited set of contexts. The approach of Ou et al. requires the specification of *integrated model transformation*s, which transform platform-independent modeling elements to the modeling elements of platform-specific models (e.g., *J2MEMetaModel*). This does not seem to be very practical and is rather untypical. Compared to our model-to-code transformation (e.g., template-based generation), a direct model-to-model transformation is very complex in terms of required transformation rules and rule maintenance.

Finally, we also want to refer to further model-driven development techniques for context-aware system design, those offered by Almeida et al [Alm+06], Sindico and Grassi [SG09], Tesoriero et al. [Tes+10], Guo and Heckel [GH04], and Jaouadi et al. [Jao+16]. **15.14**

15.3 MDD Frameworks for Context-Aware System Creation

Since the previously mentioned approaches are more focused on the conceptual design and the techniques to create context-aware systems and applications, this cluster focuses on the frameworks that can be used practically to create context-aware systems. Before the examination of the existing frameworks and tool sets for the model-driven development of context-aware systems, we present some evaluation criteria to classify the existing approaches, as well as our own approach. **15.15**

Domain: While most of the conceptual approaches named in the previous section are designed for general-purpose usage in different domains, the more practical frameworks are often tailored to a specific domain. The presumption of a domain allows putting more abstraction in the model-driven development infrastructure, i.e., mobile application developers do not have to model the technical functionality of context-awareness in detail. Instead, they model context-awareness with high-level modeling elements. Hence, the targeted domain is a relevant piece of information for the mobile application developers to find and select an appropriate framework. **15.16**

Implicit/Explicit definition of contextual dimensions and adaptation mechanisms: The existing frameworks for context-aware system development can either provide an implicit or an explicit definition of contextual dimension and adaptation mechanism. An implicit definition has the advantage that mobile application developers do not have to specify the context and the context adaptation. Instead they use a prepared modeling element. A disadvantage is that the number of these well-supported contexts is limited. Furthermore, facilities for easily adapting the mobile application may not exist or only work superficially. However, frameworks that support an explicit definition of context and application context adaptation provide more flexibility. In order to provide this flexibility, however, they need more elaborate specification of context-aware behavior by the mobile application developers. If a framework provides an implicit context definition and application context adaptation, the following two criteria are relevant: **15.17**

Supported Contextual Dimensions: The different model-driven development approaches for developing context-aware mobile applications can be classified according to the supported contextual dimensions such as *platform*, *device*, *user*, and *system* context (cf. Figure 1.2). Moreover, these contextual dimensions might be evaluated at *design time* or *runtime*. **15.18**

Adapted concerns: Assuming that mobile applications have separate concerns, such as *data*, *behavior*, and *GUI*, we evaluate which of these concerns are adaptable at runtime by the different frameworks. **15.19**

The following model-driven development frameworks with context support are **15.20** the most relevant to our work. We have already considered these frameworks in Chapter 8 (cf. Table 8.1) during the discussion of the related work to model-driven development infrastructures for mobile applications. Now, we reconsider some frameworks that are additionally able to deal with contexts:

15.21 Serral et al. [Ser+10] [Muñ+06] presented a model-driven development method for developing context-aware systems. The authors demonstrated their development method and a resulting context-aware mobile application in a home automation scenario. The development method exploits a domain-specific modeling language named *Pervasive Modeling Language* (PervML). PervML contains sub-models to address different concerns. These are the *Service model*, the *Structural model*, the *Interaction model*, the *User model*, the *Functional model*, the *Component structure model*, and the *Bind providers model*. The last three are maintained by developers who are called *Architects*. The other models are created by developers who are called *Analysts*. The *Analysts'* models are used for a complete generation of a mobile application. The *Analysts* are not obliged to implement the program code manually. On the contrary, the *Architect*s' models are used only for the partial generation of stubs. *Architect*s manually complete the hardware-related part of a context-aware application. The approach of Serral et al. contains the creation of an OWL (Web Ontology Language) specification and program code that maintains a context model at runtime. The framework supports the evaluation of *platform*, *user*, and *location* contexts. The change of a context causes a corresponding event, which will invoke several services as a reaction to an event. This event-driven behavior is specified in the *interaction model*. The framework provides a context-specific behavior. The other concerns (e.g., GUI and Data) are not affected by the context-awareness.

15.22 The approach of Serral et al. shows many similarities to our model-driven development infrastructure. Both approaches use a domain-specific modeling language to model data and behavior. The facilities for the modeling of a graphical user interface are not well provided by the approach of Serral et al. On the other hand, the management of a runtime context model is not provided in our model-driven development infrastructure, since we deal only with simple contexts. Moreover, our model-driven development infrastructure provides no facilities to introduce new context definitions and definitions of context adaptations.

15.23 Ceri et al. [Cer+07] presented a model-driven development method for developing context-aware web applications. Thus, the proposed solution is highly tailored to the domain of web applications. The development framework is based on a domain-specific visual modeling language named *Web Modeling Language* (WebML) (cf. [Cer+03], [Cer+02], and [Cer+00]). WebML provides a model-driven specification and the generation of web-based and data-driven mobile applications. A WebML model primarily describes the organization and presentation of data content in one or more hypertext views. Similar to other approaches, Ceri et al. provided different sub-schemas or sub-models. To handle contextual information, an extended, context-aware version of WebML contains a *user profile sub-schema*, a *personalization sub-schema*, and a *context model sub-schema*. These schemes contain meta-information, such as the used device, the current user location, and the user role and group. At runtime, context-aware hypertext pages filter the data records based on this modeled meta-information in different context-related criteria. The approach evaluates *user* and *location* contexts at runtime and adapts the data to these contexts. The behavior or the graphical user interface is not adapted by the appearance of different contexts.

15.24 The approach of Ceri et al. is different in many ways from our model-driven development infrastructure. While Ceri et al. focused on the generation of mobile web applications, our approach focuses on the generation of native mobile applications. Thus, we have to additionally deal with platform contexts at design time. Moreover, web applications are usually not offline-capable. Therefore, some system contexts are not relevant for mobile web applications because they cannot support these due to architectural limitations. The location-aware filtering of application data seems to be state of the art for mobile applications, as Baldauf et al. argued [Bal+07, Sec. 3.1]. The fixed and limited modeling of contextual dimensions is a similarity between the framework of Ceri et al. and ours. None of the frameworks provide any facility to introduce new context definitions and definitions of context adaptations.

Escolar et al. [Esc+14] proposed a model-based approach to generate online- and offline-capable mobile applications. The framework focuses on the domain of web applications. It is based on the open-source project *MyMobileWeb* (MMW) [30]. MMW's modeling facilities comprise a data model that captures the application data as well as the contextual data, an IDEAL2 [AI10, pp. 166] model for the user-interface specification, and a *State Chart XML* model (SCXML) [75] to specify the navigation through the mobile application. The authors extended the SCXML model of the existing MMW framework to provide additional online- and offline-capable behavior of the generated mobile applications. For example, the added attributes denote where the data elements should be located (client-server) and how the data should be synchronized. Mobile applications that are generated based on this extended models can evaluate device and connection contexts at runtime and adapt the behavior (navigation) and the graphical user interface to these contexts. However, the mobile application developers cannot define custom contexts or application adaptation. **15.25**

Comparing our model-driven development infrastructure with the framework of Escolar et al., we recognize a lot of commonalities. The evaluated contexts and the adapted concerns of the generated mobile applications overlap, i.e., both support the evaluation of the system and the device context and can adapt the behavior (navigation) and the graphical user interface of the mobile application. Concerning the online and offline capability, both solutions show a lot of similarities, i.e., both realize architectural components like replication and synchronization. The solution of Escolar et al. provides a detailed configuration on how data entities should be managed. However, it provides no model analysis and cannot deal with conflicting processes, which appear in multi-user transaction systems. The solution of Escolar et al. could be used if a mobile application needs neither complex hardware access to the device's sensors nor contains conflicting multi-user access to data objects. **15.26**

Finally, Kapitsaki et al. [Kap+09] (cf. also [Kap+08]) presented a model-driven development method for developing context-aware web applications based on context-aware web services. This approach also focuses on the domain of mobile web applications. The modeling language used is UML, which is extended with profiles. A *web services profile* describes the used web services of a mobile application. A *context profile* provides the specification of contexts. Finally, a *presentation profile* is used to model the graphical user interface presentation, its flow, and the navigation properties. The web application runs on an application server, i.e., the architecture is client-server-based. We find particularly interesting that the proposed architecture provides the concept of a *context server* (cf. Section 15.1). Both context sensing and business services are provided through web services. Thus, this server also provides context information that may come from the requesting client or from any other remote source. For example, a context sensor could be a web service providing weather data. The provided demonstration scenario shows a ticket-booking system for a cinema. The evaluated contexts are the location and the user context. The mobile application adapts the data shown according to these contexts. **15.27**

The approach of Kapitsaki et al. shows the least amount of similarity with our model-driven development infrastructure. The contradictory nature of both approaches reveals that the *context server* based mobile applications cannot be online- and offline-capable since they heavily rely on services. On the other hand, our model-driven development infrastructure can hardly implement a context-server architecture while being online- and offline-capable at the same time. Once more, the proper selection of an appropriate architecture becomes important when starting a mobile application development project. **15.28**

TABLE 15.1: Model-driven development approaches for context-aware mobile applications

Name	**Domain (Example[1])**	**Implicit definition of context and context adaptation**	**Evaluated contextual dimensions ...**	**Automatic context adaptation**		**Adapted concerns**
				... at design time	**... at runtime**	
Serral et al.	Native application (Home automation)	×	Platform	✓	×	Behavior
			Location	×	✓	
			User	×	✓	
Ceri et al.	Data-driven web applications	✓	Location	×	✓	Data
			User	×	✓	
Escolar et al.	Data-driven web applications	✓	System	×	✓	Behavior GUI
			Device	×	✓	
Kapitsaki et al.	Data-driven web applications (Cinema ticket system)	×	Location	×	✓	Data
			User	×	✓	
Our Framework	Data-driven native applications	✓	Platform	✓	×	Behavior GUI Data
			User	✓	✓	
			Device	✓	✓	
			System	✓	✓	

15.29 Table 15.1 shows the model-driven development approaches and the software engineering toolsets for context-aware software development and the classification of their main characteristics. A ✓-symbol indicates that the corresponding feature is supported, whereas an ×-symbol indicates that it is not supported.

15.30 To sum up, as shown in Table 15.1, most of the model-driven development approaches for context-aware software development do not consider the contextual dimension of system contexts, particularly the connectivity context. Only one of the evaluated frameworks provides the generation of online- and offline-capable mobile applications. Although nearly half the frameworks provide custom context definitions and custom definitions of application adaptation, the relevance of these mechanisms for further potential contextual dimensions is difficult to determine. Moreover, the frameworks without a customizable definition and adaptation mechanism often implement the adaptation of only the data layer, instead of the application behavior or the graphical user interface.

15.31 Our framework for the model-driven development of mobile application supports most of the identified features. A major feature that is intentionally not supported by our solution is the custom definition of contexts and application adaptation.

[1] If any given.

Chapter 16

Summary and Outlook

This concluding chapter summarizes the main contributions of this thesis. In the outlook section, we point to limitations of the framework for model-driven development of mobile applications with context support and discuss how further work could remove these limitations. Moreover, we show how the presented work can be extended. We also outline possible directions for future research. **16.1**

16.1 Summary

As shown in the introduction to this thesis, the software engineering of mobile application poses many challenges for mobile application developers, making it difficult to complete a development project successfully. Competing technical approaches (web-based, native, hybrid), different architectures (client-server, rich client, (generic) extended client-server), heterogeneity of software platforms and devices, and different types of mobile applications (standalone application, information system, multi-user transaction system) are only a few decision points where mobile application developers have to make the appropriate choices. Software engineering for mobile applications lacks a one-fits-all solution. Moreover, some types of mobile applications, such as multi-user transaction systems, are not covered by the existing approaches. Model-driven development seems to be a promising approach to tackle these problems. Models are an ideal abstraction from technical concerns. They allow mobile application developers to focus on certain aspects of mobile applications, such as data, behavior, or graphical user interface modeling. Subsequently, code generators can transform an app model into a partial or complete implementation on multiple platforms. Moreover, the model-driven development infrastructure can provide model analysis and simulation techniques to further support mobile application developers. **16.2**

This thesis introduced a framework for the model-driven development of mobile applications that is beyond many features of the existing model-driven development infrastructures in terms of *modeling aspects*, *modeling techniques*, *application implementation* approaches, and the supported *application architectures* (cf. Table 8.1). Moreover, the proposed model-driven development infrastructure provides the *processing of different contextual dimensions* and the *contextual adaptation* of the generated mobile application at design and runtime (cf. Table 15.1). Key elements of the framework are a two-level modeling approach that provides design time models and runtime models, and a generic extended client-server architecture of the generated mobile applications. **16.3**

The elementary contribution (Part I) of a model-driven development infrastructure was developed rigorously and following an agile bottom-up development process. The domain-specific modeling language and the corresponding editors provided an adequate toolset for the specification of mobile applications. The domain-specific modeling language and the model editors were validated with respect to the guidelines for domain-specific modeling languages and the guidelines **16.4** for user interface description languages. Moreover, a user experience evaluation showed the applicability of the model-driven development infrastructure.

16.5 The supplementary contribution (Part II) of context support demonstrates that the app models can be reused and extended easily to support static contexts, such as user contexts or device contexts. The two-leveled modeling approach also provides runtime adaptation of an already-deployed mobile application. Mobile applications can be adapted to dynamic contexts at runtime. The generic extended client-server architecture shows serious advantages creating online- and offline-capable mobile applications. In particular, online- and offline-capable multi-user transaction systems, which are beyond the capability of state-of-the-art systems, can be created. Based on the high abstraction level due to the application of model-driven development techniques, mobile application developers need fewer technical skills to realize context-aware mobile application with an appropriate architecture.

16.6 In addition to the tangible contributions listed in Section 1.5, we applied different methods for artifact evaluation and validation to follow a rigorous research methodology. Most of the designs are developed in a constructive way, such that we use a *persuasive* validation (cf. Shaw's research taxonomy [Sha01]). We use the defined functional and non-functional requirements as well as the feature model, to show that our contributions fulfill these requirements, i.e., features. The requirements and the domain analysis are based on a collaboration with experts from industry. Thus, we consider the set of requirements and the domain analysis to be credible and realistic. The domain-specific modeling language was validated by an evaluation taking existing guidelines for the design of domain-specific modeling languages into account. Different case examples (Case example 1-3) demonstrate that the generated mobile applications cover both the requirements and the desired features. A user experience evaluation shows the applicability and usability of the developed toolset. Moreover, technical evaluation of the contributions shows that model-driven development infrastructure scales well and works correctly i.e., generates correct program code of high quality. Further case examples (Case example 4-7) demonstrate that the context support that is enabled by our approach can be used in practice.

16.7 In this thesis, we emphasized that model-driven development of mobile applications is a promising approach to tackle the challenges of mobile application development, in particular, the realization of context-aware mobile applications. As a crucial part of the development process, app models can be transformed and adapted to support many potentially contextual dimensions. Moreover, the co-existence of design time and runtime models within a two-leveled modeling approach can be useful in dealing with static and dynamic contexts. A two-leveled modeling approach such as this is beyond the state of the art. Therefore, we consider this as a new concept in the area of model-driven development and we are convinced that this concept might be useful in model-driven development for other domains as well (e.g., embedded systems).

16.2 Outlook

16.8 The model-driven development infrastructure focuses on the generation of native applications, which are data-oriented, i.e., used merely in business environments. The domain-specific modeling language and the code generators are tailored to this application domain. Hence, neither complex algorithms nor sophisticated graphical user interfaces can be specified by the provided domain-specific modeling language.

16.9 Further work could provide additional modeling elements or even additional submodels to specify further concerns of mobile applications which are related to other domains (e.g., gaming, social and medical applications).

16.10 The model-driven development infrastructure intentionally provides a high abstraction level. Mobile application developers can not directly change the generated architecture of the mobile applications. The model-driven development infrastructure produces mobile applications that can be run directly. Mobile application developers do not have to complete the generated implementation manually (e.g., filling program code stubs). In turn, the model-driven development infrastructure is not designed for manual code completion or modification (e.g., providing protected regions). Like most model-driven development frameworks and environments, the manual extensibility of the generated code is limited.

In order to specify the architecture of the generated mobile application explicitly, architectural sub-models could be introduced. Domain experts may use the existing modeling facilities (e.g., data, behavior and GUI model) while technical experts may complete the app model with technically-oriented models, such as an architectural sub-model. Mobile application developers might use an architectural sub-model to more explicitly specify the design of the generated mobile apps. Currently, code generators make a lot of design decisions (e.g., full replication of data records for offline transaction processing), which can often be optimized (e.g., partial replication of declared data records). Moreover, an architectural sub-model allows the specification of interfaces to other services or mobile applications. **16.11**

Supporting the manual completion of program code will remain a major problem since only a round-trip engineering approach will provide a co-evolution of model and code artifacts. **16.12**

A further limitation lies in the fixed number of supported contextual dimensions. Although the author is convinced that the design of the model-driven development infrastructure, as well as its iterative bottom-up development approach, is beneficial for adding further contextual dimensions, the custom definition of the context and context adaptation mechanisms were not part of this thesis. Moreover, except of the connectivity context, the generated mobile application will not trigger the adaptation automatically (e.g., self-adaptive systems [Lem+13]). **16.13**

Thus, to be able to model and generate context-aware mobile applications for arbitrary contexts, the context sensing and reasoning sub-system (cf. Figure 15.1) must be developed in more detail. For example, the generated mobile applications currently provide neither an internal context model which stores appearing and disappearing contexts over time, nor do they provide reasoning about such data. **16.14**

In opposite to the future work that is motivated by theses limitations, we consider other obvious tasks as future work: **16.15**

Even if we focus on a graphical model editor with a graphical concrete syntax, a textual concrete syntax is also conceivable and has some advantages. The advantage of a textual concrete syntax is that the platform-specific parts (e.g., program code) could be embedded in the domain-specific textual concrete syntax. Many technical experts like the modeling approach, but dislike modeling with a graphical model editor. Writing the model in a domain-specific textual concrete syntax is more akin to programming, and many benefits are provided by state-of-the-art language tool kits (e.g., auto-completion and quick-fixes). **16.16**

Since we argued earlier that a model-driven development approach covers the cross-platform approach, we can demonstrate this by creating a cross-platform code generator (e.g., generating Cordova applications, web application). Moreover, other native platforms (e.g., Windows Phone) could be also supported by a corresponding code generator. **16.17**

Considering a setting more focused on industrial applications, future work encompass a comprehensive model-driven development environment which combines **16.18** model-driven development, model quality assurance, and model versioning. Moreover, since we generate a complete mobile application from the entire app model within each of the iterations, a partial generation might be more beneficial.

16.19 Finally, as any other framework that was designed for software developers, we could conduct further evaluations such as developers studies or more qualified case studies to compare the model-driven development approach with the traditional development approach.

The following research directions are related to model-driven development, mobile applications, and context-awareness but not directly focused on the designed framework:

16.20 *Empirical Evaluation of Design Variants:* Mostly, any model-driven development infrastructure offers the opportunity to modify the app models rapidly without considerable effort. As a result, many different variants of a system could be created by the code-generation facilities. These design variants are well-suited for further investigation concerning different criteria. For example, any two variants of a mobile application could be compared by the existing techniques and tools according to the usability, energy consumption, or security. The gained insight could be used to select the most effective app model or to optimize the code generators.

16.21 *Model-Driven Testing of Mobile Applications:* Our domain-specific modeling language for describing mobile applications is usually used as the design specification of mobile applications that should be generated. In contrast, the domain-specific modeling language, i.e., the app models, can also be used as implementation models to describe the existing mobile applications that are not necessarily developed in a model-driven way. Hence, the app models can describe an existing mobile application in an abstract manner. Such an abstract description of an already existing mobile application can be used for model-driven testing. A *test model* could be deduced from the app model, which describes test cases for the mobile application. Moreover, code generators can generate a test implementation, i.e., test cases, for several potential test frameworks and platforms. The different contexts in which mobile applications may operate could be taken into account during the test-case generation. Model-driven testing will ease and reduce the effort of testing, particularly if more than one platform must be maintained by the mobile application developers. The derivation of test models and the context-aware test case generation are future research directions.

16.22 *Model Extraction from Mobile Applications:* Although the model-driven testing of existing mobile applications seems to have tremendous potential since testing activities are often ignored in mobile application development projects, the extent of effort the mobile application developers must exert to construct a corresponding analysis model must be taken into account. The creation of an analysis app model from an existing mobile application can be considered as an inverse mobile application generation. The mobile application developers map the generated program code, or the known functionality and data structures, to the more abstract model elements of the domain-specific modeling language. In contrast, the model-driven development infrastructure contains these mapping rules between abstract model elements and program code inside its code templates. Hence, a further research direction could be the construction of semi-automated extraction tools based on the information provided by the model-driven development infrastructure.

16.23 *Context Description, Context Collection, and Context Adaptation:* As mentioned earlier, our proposed model-driven development framework is not a holistic context-aware framework. Mobile application developers can neither specify new contexts and their collection nor define the adaptation mechanism of the generated mobile application to newly defined contexts (e.g., security contexts, social contexts, or energy contexts). The presented work can be extended to realize these features. To do this, the domain-specific modeling language must be extended with an additional sub-model. This model captures the contextual information. The context collection and context-adaptation functionalities could be added by the injection of custom code, as shown in Section 7.3.5.

Design Time vs. Runtime Instantiation: The model-driven development framework proposed by us uses models at design time to generate native program code, and additional runtime models to configure the generated mobile applications at runtime. The infrastructure developers of the model-driven development framework decide which model elements are translated at design time to the native program code and which model elements are interpreted at runtime by the generated mobile application. For several reasons (e.g., security, privacy, and performance) it is desirable to maintain both variants, i.e., model synthesis (code generation) and model interpretation, and decide later which variant should be deployed. As a result, the mobile application developer could control whether the resulting mobile application is an interpreter or runs from static, compiled code. **16.24**

Appendices

Appendix A

Well-Formedness Rules

The declarative language definition of the app models by metamodels also includes a number of well-formedness rules. The following three sections specify the well-formedness rules of the domain-specific modeling language. A.1

A.1 Constraints for the Data Model

The constraints of the Ecore model have been completely adopted, which is why we refer only to the implementation. You will find the constraints or their respective implementations, in the class `org.eclipse.emf.ecore.util.EObjectValidator` and its subclass `org.eclipse.emf.ecore.util.EcoreValidator`. So far, additional constrains have not been formulated. A.2

A.2 Constraints for the Process Model

1. There is exactly one process with name *Main*. This process is the first one to be executed.

```
context ProcessContainer inv:
self.processes->select(p : Process | p.name = 'Main') ->
size() = 1
```

2. There is at least one task of type *ProcessSelector* in a *Main* process.

```
context Process inv:
self.name = 'Main' implies containProcessSelectorTask(
    self.startTask.oclAsType(wcore::Task))
```

```
def containProcessSelectorTask(t:Task):Boolean
if (t.oclIsTypeOf(wcore::If)) then
    containProcessSelectorTask(t.oclAsType(wcore::If)
    ._body.oclAsType(wcore::Task)) else
if (t.oclIsTypeOf(wcore::While)) then
    containProcessSelectorTask(t.oclAsType(wcore::While)
    ._body.oclAsType(wcore::Task)) else
if (t.oclIsTypeOf(wcore::IfElse)) then
    containProcessSelectorTask(t.oclAsType(wcore::IfElse)
    ._body.oclAsType(wcore::Task)) and
    containProcessSelectorTask(t.oclAsType(wcore::IfElse)
    ._elseBody.oclAsType(wcore::Task)) else
if (t.oclIsTypeOf(wcore::Sequence)) then
    containProcessSelectorTask(t.oclAsType(wcore::Sequence)
    ._body.oclAsType(wcore::Task)) or
    containProcessSelectorTask(t.oclAsType(wcore::Sequence)
    .follower.oclAsType(wcore::Task)) else
t.oclIsTypeOf(wcore::ProcessSelector)
endif endif endif endif
```

3. A *Process*, which is registered in a *ProcessSelector*, contains - potentially transitively - at least one task of type *InvokeGUI* or a *CrudGui* task.

```
context ProcessSelector inv:
self.oclAsType(wcore::ProcessSelector).processes ->
select(proc:wcore::Process|containInvokeGUITask(
    proc.startTask.oclAsType(wcore::Task))) ->
size() = (self.oclAsType(wcore::ProcessSelector).processes
-> size())
```

```
def containInvokeGUITask(t:Task):Boolean
if (t.oclIsTypeOf(wcore::If)) then
    containInvokeGUITask(t.oclAsType(wcore::If)
    ._body.oclAsType(wcore::Task)) else
if (t.oclIsTypeOf(wcore::While)) then
    containInvokeGUITask(t.oclAsType(wcore::While)
    ._body.oclAsType(wcore::Task)) else
if (t.oclIsTypeOf(wcore::IfElse)) then
    containInvokeGUITask(t.oclAsType(wcore::IfElse)
    ._body.oclAsType(wcore::Task)) and
    containInvokeGUITask(t.oclAsType(wcore::IfElse)
    ._elseBody.oclAsType(wcore::Task)) else
if (t.oclIsTypeOf(wcore::Sequence)) then
    containInvokeGUITask(t.oclAsType(wcore::Sequence)
    ._body.oclAsType(wcore::Task)) or
    containInvokeGUITask(t.oclAsType(wcore::Sequence)
    .follower.oclAsType(wcore::Task)) else
if (t.oclIsTypeOf(wcore::InvokeProcess)) then
    containInvokeGUITask(t.oclAsType(wcore::InvokeProcess)
    .subProcess.startTask.oclAsType(wcore::Task)) else
if (t.oclIsTypeOf(wcore::ProcessSelector)) then
    t.oclAsType(wcore::ProcessSelector).processes ->
    select(proc:wcore::Process|
    containInvokeGUITask(proc.startTask.oclAsType
    (wcore::Task))) -> size() = (t.oclAsType
    (wcore::ProcessSelector).processes -> size()) else
t.oclIsTypeOf(wcore::InvokeGUI) or
t.oclIsTypeOf(wcore::CrudGui)
endif endif endif endif endif endif
```

4. Invoking a process, the list of arguments has to be consistent to the list of parameters defined for that process w.r.t. number, ordering and types.

```
context InvokeProcess inv:
self.arguments -> size() = self.subProcess.variables ->
select(v:wcore::Variable|v.scope.oclAsType(wcore::Scope) =
wcore::Scope::GLOBAL) ->size() and
Sequence{1..self.arguments -> size()} ->
forAll(i:Integer | let var1:wcore::Variable =
self.arguments->at(i), var2:wcore::Variable =
self.subProcess.variables->select(v:wcore::Variable|
v.scope.oclAsType(wcore::Scope)=wcore::Scope::GLOBAL) ->
at(i) in var1.oclAsType(wcore::Variable).eType =
var2.oclAsType(wcore::Variable).eType)
```

5. Considering task *InvokeGUI*, number, ordering and types of input and output data as well as output actions have to be consistent with the type of page invoked. E.g., a *MapPage* gets two Double values as output data, but a *LoginPage* gets a Boolean value as output data representing the result of a login trial and two strings as input to show the user name and password.

MapPage (Output)

```
context InvokeGUI inv:
self.page.oclIsTypeOf(gcore::MapPage) implies
self.outputData -> size() = 2 and self.outputData ->
forAll(var:Variable|var.eType=ecore::EDouble)
```

LoginPage (Output)

```
context InvokeGUI inv:
self.page.oclIsTypeOf(gcore::LoginPage) implies
self.outputData -> size() = 1 and self.outputData ->
forAll(var:Variable|var.eType=ecore::EBoolean)
```

LoginPage (Input)

```
context InvokeGUI inv:
self.page.oclIsTypeOf(gcore::LoginPage) implies
self.input -> size() = 2 and self.input ->
forAll(var:Variable|var.eType=ecore::EString)
```

ViewPage (Output)

```
context InvokeGUI inv :
self.page.oclIsTypeOf(gcore::ViewPage) implies
self.outputData -> size() = 1 and self.outputData ->
forAll(var:Variable|var.eType.oclIsKindOf(ecore::EObject))
```

EditPage (Output/Input)

```
context InvokeGUI inv :
self.page.oclIsTypeOf(gcore::EditPage) implies
self.outputData -> size() = 1 and self.input ->
size() = 1 and self.input -> forAll(var:Variable|
var.eType=self.outputData -> at(1).eType)
```

SelectableListPage or *ListPage* (Output)

```
context InvokeGUI inv:
self.page.oclIsTypeOf(gcore::SelectableListPage) or
 self.page.oclIsTypeOf(gcore::ListPage) implies
self.outputData -> size() = 1 and self.outputData ->
forAll(var:Variable|var.upperBound=-1)
```

SelectablePage (Input)

```
context InvokeGUI inv:
self.page.oclIsTypeOf(gcore::SelectableListPage) implies
self.input -> size() = 1 and self.input ->
forAll(var:Variable|var.eType=self.outputData ->
at(1).eType and var.upperBound=1)
```

A.3 Constraints for the GUI Model

1. The attributes *blue*, *red*, *green* of the Class *RGBColor* must be in the interval from 0 to 255.

```
context RGBColor inv:
0 <= self.blue and self.blue <= 255 and
0 <= self.green and self.green <= 255 and
0 <= self.red and self.red <= 255
```

Appendix B

App Models

This appendix provides complete app models and excerpts of app models. Their notation follows the graphical concrete syntax presented in Section 6.6. Instance models are not shown. Neither are properties, which are only displayed in the properties view of a selected graphical concrete syntax element. B.1

B.1 Phone Book App Model

B.1.1 Data Model

As shown already within the example of Section 6.3.1 – Figure 6.4, the contact data is structured in the classes *Person* and *Address*. Moreover, Figure B.1 shows the modeled *EOperations* which provide further custom functionality. B.2

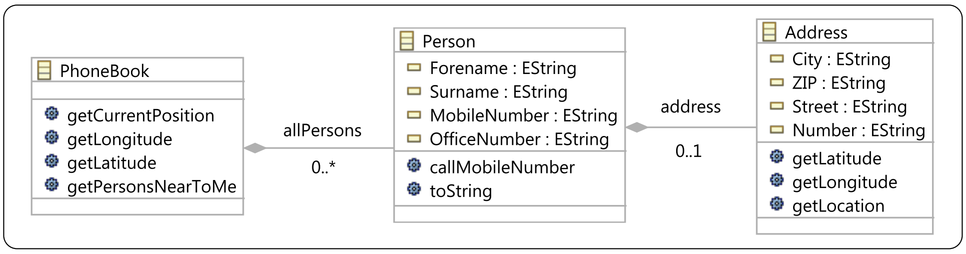


FIGURE B.1: Data model of the simple phone book application

The operations *getCurrentPosition*, *getLongitude*, and *getLatitude* of the container class *PhoneBook* return the current position of the mobile device, i.e., a geographical coordinate (longitude/latitude). The operations *getLocation*, *getLongitude*, and *getLatitude* of the class *Address* resolve the logical address, i.e., street and city, to a geographical location (longitude/latitude). The operation *getPersonsNearToMe* returns Persons whose address is near to the current position of the mobile application user, i.e., its device. The operation *toString* is implicitly used when displaying *Person* objects inside a listable page (cf. Section 7.3.2.1). B.3

B.1.2 GUI Model

B.4 The modeled pages are shown in Figure B.2 referring all to the default style setting, which is shown at the left-hand side of Figure B.2. The GUI model contains a *CustomPage*, named *CustomViewPerson*. An annotation (cf. Listing 7.2) with a custom layout (cf. Figure 7.7b) is attached to this page.

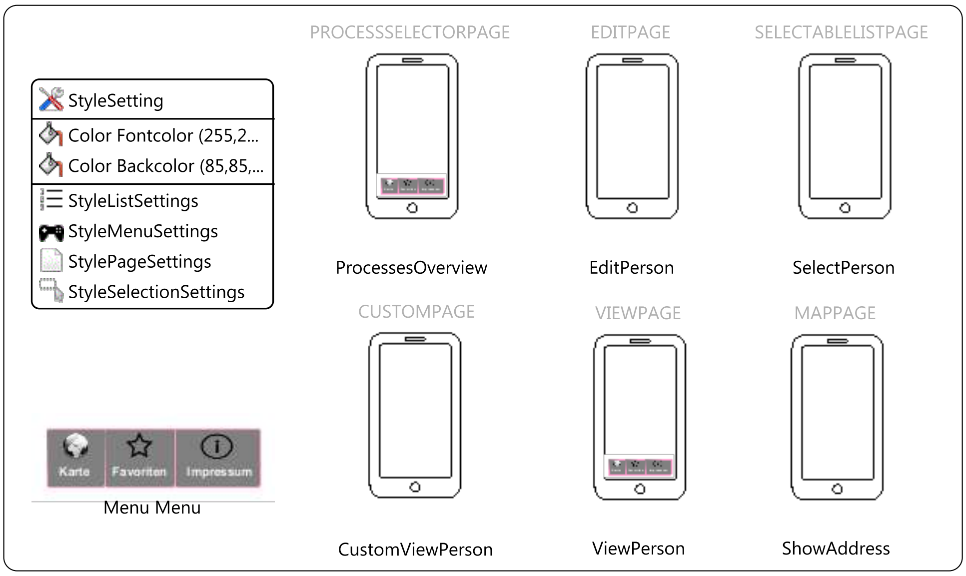


FIGURE B.2: GUI model of the simple phone book application

B.1.3 Process Model

The process model contains 13 processes overall. Two processes are auxiliary processes (Processes *SelectAndShowPersonAddressOnMap* – B.1.3.10 and *PersonsWithAddress* – B.1.3.11) which cannot be invoked directly by the mobile application users. However, the number of available processes depends on the configuration of the *ProcessSelector* inside the process *Main* (cf. Figure B.3). **B.5**

B.1.3.1 Process *Main*

The *Main* process is the first one to be executed. It contains all process that a mobile application user can reach from the main screen of a mobile application. The main process may be reconfigured to provide role-specific variants of the mobile application (cf. Figure 12.3). **B.6**

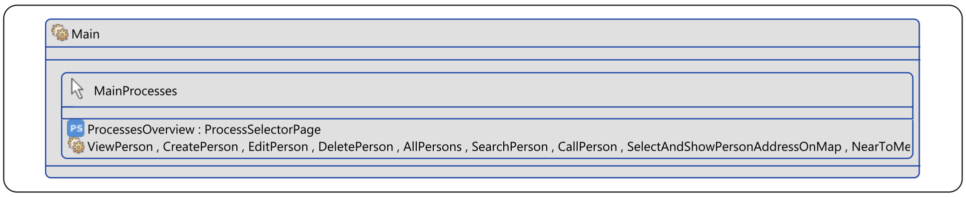


FIGURE B.3: Process *Main*

B.1.3.2 Process *ViewPerson*

This process allows viewing an existing *Person* object. In this case, a customized view, named *CustomPage*, is invoked instead of a standard *ViewPage* (cf. Process *AllPersons* – B.1.3.6). The first task creates an empty search pattern. The reading task *ReadAllPersons* delivers every *Person* object since these match the empty search pattern. The third task shows a list which contains the *Person* objects. A mobile user may select a list element, i.e., a Person, to get a detailed view, which is provided by the last task element of the process. **B.7**

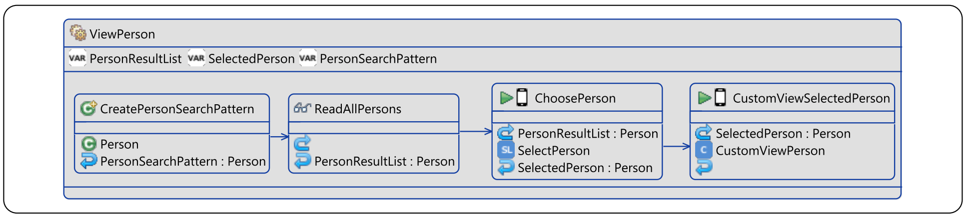


FIGURE B.4: Process *ViewPerson*

B.1.3.3 Process *CreatePerson*

B.8 This process creates a new *Person* object. The first appearing page (*EditPage*), named *EditPerson*, allows the user to enter the personal data of a person including the address. Subsequently, a non-editable page (*ViewPage*), named *ViewPerson*, displays the input data.

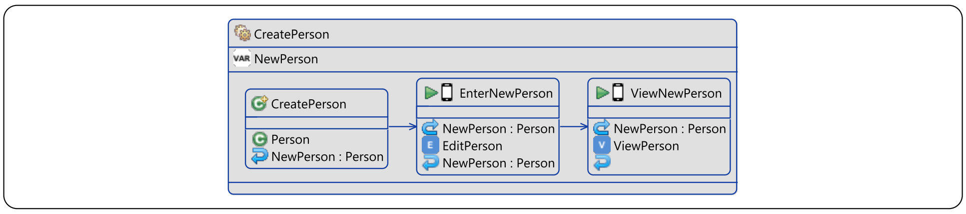


FIGURE B.5: Process *CreatePerson*

B.1.3.4 Process *EditPerson*

B.9 This process allows editing an existing *Person* object. Before editing, the user has to choose a person from a list (*SelectableListPage*). Subsequently, the edit page (*EditPage*), named *EditPerson*, and a non-editable page (*ViewPage*), named *ViewPage*, are shown.

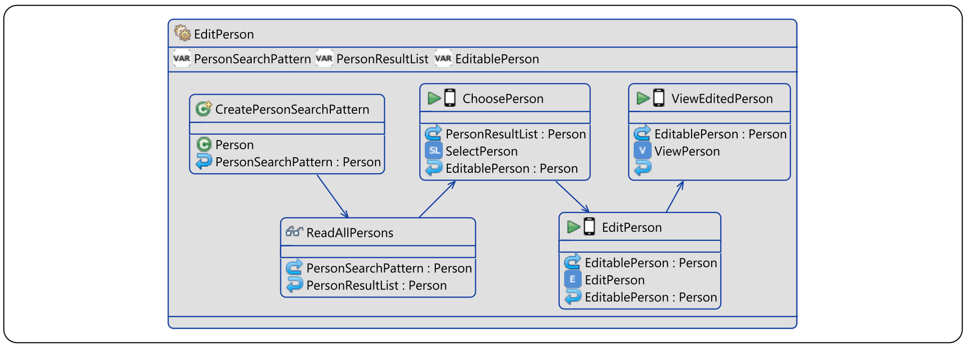


FIGURE B.6: Process *EditPerson*

B.1.3.5 Process *DeletePerson*

B.10 This process allows deleting an existing *Person* object. Before deleting, the user has to choose a person from a *SelectableListPage*, named *SelectPerson*.

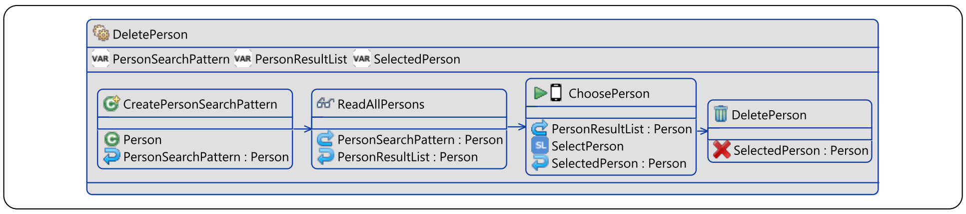


FIGURE B.7: Process *DeletePerson*

B.1.3.6 Process *AllPersons*

This process displays all existing persons in a selectable list (*SelectableListPage*). The B.11
user can select one *Person* object and see its details in a single view (*ViewPage*).

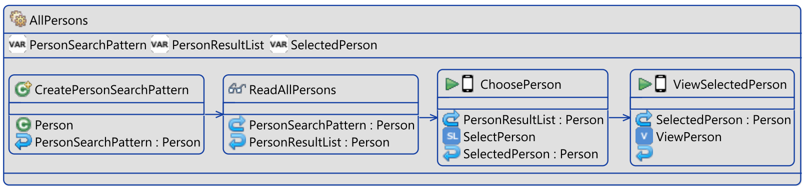


FIGURE B.8: Process *AllPersons*

B.1.3.7 Process *CRUDPerson*

This process replaces the before mentioned processes (*CreatePerson* – B.1.3.3, *EditPer-* B.12
son – B.1.3.4, *DeletePerson* – B.1.3.5, *AllPersons* – B.1.3.6) in a functionally equivalent and standardized way. Individual or customized styles (e.g., Process *ViewPerson* – B.1.3.2) are not provided. During preprocessing (cf. Section 7.3.2) the code generator expands this abstract model element to a process which contains standard model elements such as described before (e.g., *Read*, *InvokeGUI* task).

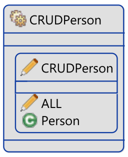


FIGURE B.9: Process *CRUDPerson*

B.1.3.8 Process *SearchPerson*

This process offers a search function. The first appearing page (*EditPage*), named B.13
EditPerson, gathers the search criteria from the user-provided search pattern. Subsequently to the search criteria page, a result list, named *SelectPerson* (*SelectableListPage*), appears and shows the *Person* objects that match the search pattern. The mobile application user can select one *Person* object and see its details in a separate view (*ViewPage*).

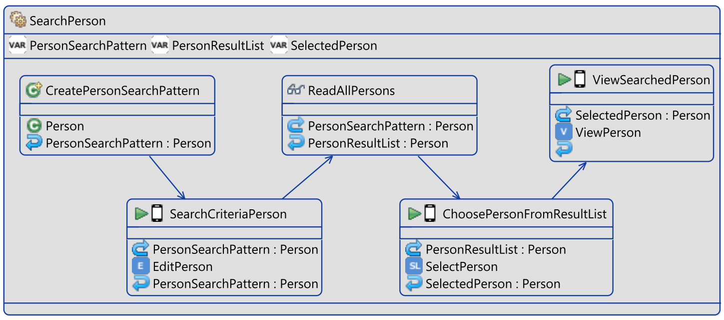


FIGURE B.10: Process *SearchPerson*

B.1.3.9 Process *CallPerson*

B.14 This process allows calling a person in the phone book. Before the call, the user has to choose a person from the list (*SelectableListPage*) of all persons. The task *CallPerson* requires a permission to indicate that the invoked operation uses the call service of the mobile phone.

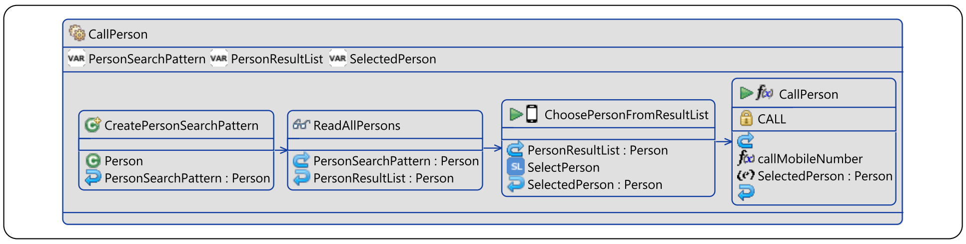


FIGURE B.11: Process *CallPerson*

B.1.3.10 Process *SelectAndShowPersonAddressOnMap*

B.15 By this process, all *Person* objects from the phone book that have an address are displayed in a list (*SelectableListPage*). The list of persons is given by a parameter variable. The mobile application user can select one *Person* object and see their address, i.e., the geographical location, on a map. Similar to the process *CallPerson*, the process *SelectAndShowPersonAddressOnMap* requires a permission to use the map service of the current mobile platform.

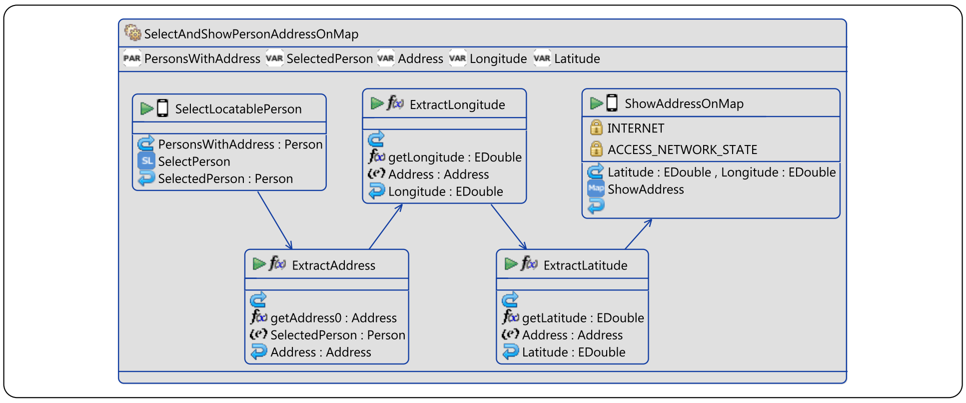


FIGURE B.12: Process *SelectAndShowPersonAddressOnMap*

B.1.3.11 Process *PersonsWithAddress*

This process creates two empty search patterns: i) an empty search pattern of the B.16
type *Person* and ii) and empty search pattern of the type *Address*. The third task combines the search patterns, i.e., add the empty address search pattern to the *Person* object. This is required because the following task should only read *Person* objects with a non-empty address reference. The read task delivers all *Person* objects that match the complex search pattern and thus persons who have an address. The process makes the result accessible for other processes by defining the variable *PersonsWithAddress* as a return variable.

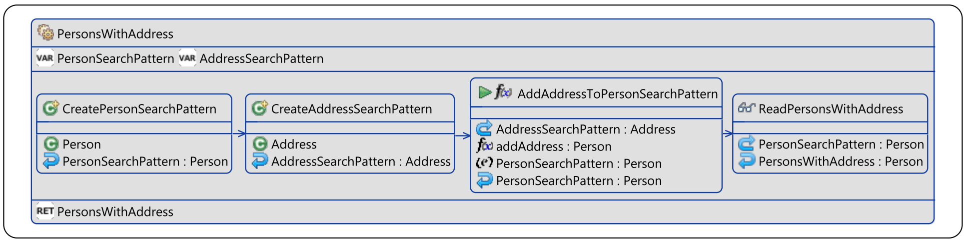


FIGURE B.13: Process *PersonsWithAddress*

B.1.3.12 Process *ShowPersonAddressOnMap*

The process *ShowPersonAddressOnMap* combines the two before mentioned process. B.17
The output of the process *PersonsWithAddress* is forwarded as input of the process *SelectAndShowPersonAddressOnMap*. Overall, the composed process provide the reading of *Person* objects that have an address, the selection of a *Person* object, and the view of the geographical location of the corresponding address on a map.

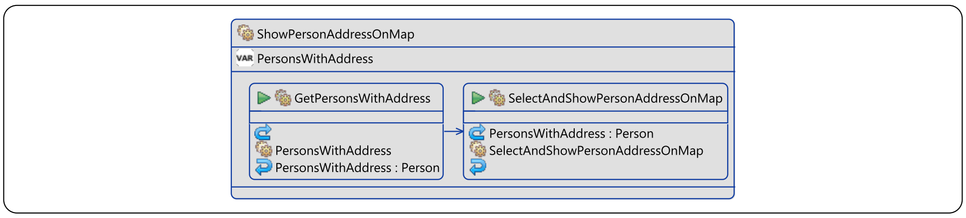


FIGURE B.14: Process *ShowPersonAddressOnMap*

B.1.3.13 Process *NearToMe*

By this process, all persons from the phone book with an address near to the current B.18
position of the mobile application user are displayed in a list (*SelectableListPage*). The user can again select one *Person* object and see their address on a map.

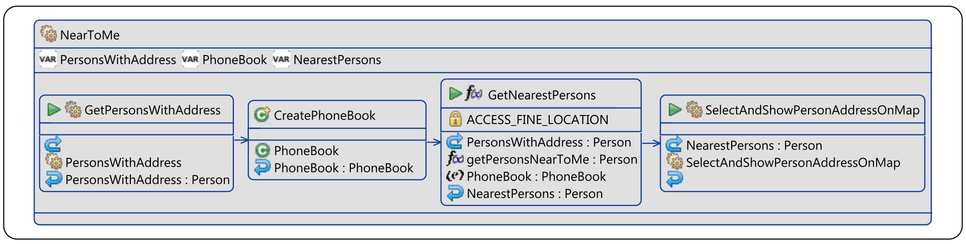


FIGURE B.15: Process *NearToMe*

B.2 Conference App Model

B.2.1 Data Model

B.19 The basic elements of a conference, i.e., sessions, presented papers and of course persons in their roles as authors or session chairs, are modeled straightforwardly in the Ecore data model. The class *Conference* is the overall container. It contains *Sessions*, *Persons*, *Papers*, *Rooms* and *Venue*. A *Session* is connected to the *Room* where it takes place and to all the *Papers* to be presented in that *Session*. Moreover, *Persons* are indicated as session chairs and authors. Note that there are several operations *getPlanFilenames*, *addRemoveToFavorites*, *initializeNotFavored*, *initializeFavored*, *toString*, *compareTo*, *getLatitude*, *getLongtitide*, and *getLocation* that are modeled as EOperations and have an Ecore annotation with platform-specific code. All other operations (e.g., getters and setters) are generated automatically.

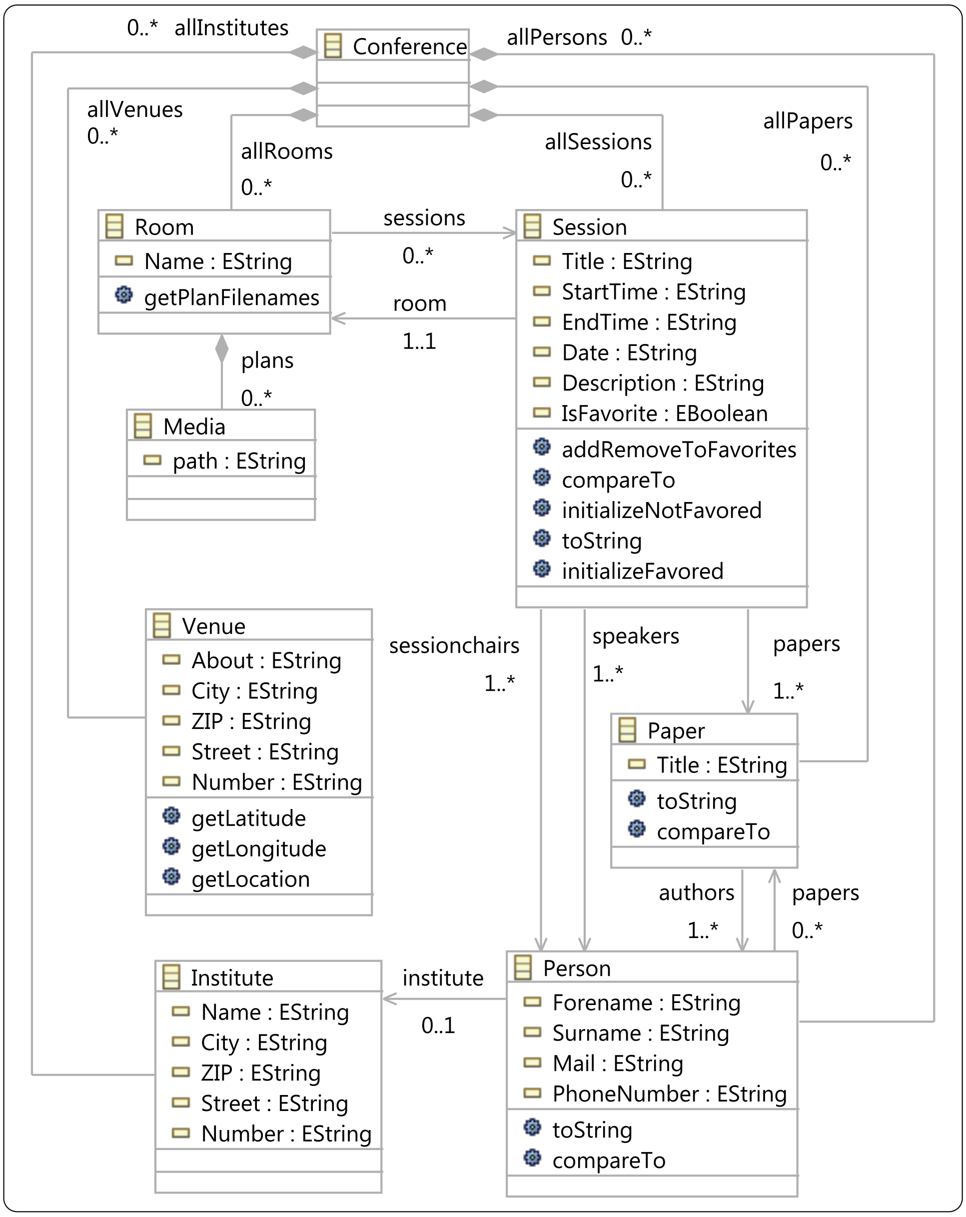


FIGURE B.16: Data model of the conference application

B.2.2 GUI Model

The GUI model contains only standard pages implementing a default layout. All views use a default style setting. The mobile application modelers use a separate *ViewPage* and *EditPage* for each entity (*Person* and *Paper*). Although all pages refers to the same style setting element, this redundant modeling allows the use of different style setting elements for each page. Besides, the code generator produces a separate code fragment (e.g., an Android-Activity) for each of the modeled pages, which might be customized manually. In turn, the *SelectableListPage* is used in different contexts (e.g., Processes *RRoom* – B.2.3.5 and *Add Favorite* – B.2.3.7), showing different kinds of entities (e.g., *Room* and *Session*). This is an example for generic use of pages with different invocation contexts. **B.20**

FIGURE B.17: GUI model of the conference application

B.2.3 Process Model

B.21 The process model captures the behavior for all processes, and consists 17 processes. For conference administrators, CRUD processes with full permission shall be available for every defined conference entity. For conference participants, read access shall be realized for every defined conference entity. Furthermore, it shall be possible to mark a session as favorite. The actual configuration of processes is done in the provider models for conference administrators and participants (cf. Figure 12.4)

B.2.3.1 Process *Main*

B.22 The *Main* process is the first one to be executed. It contains all process that a providing user or mobile end user can reach from the main screen of a mobile application. This configuration may be reconfigured by a provider model or directly (e.g., using design time instantiation).

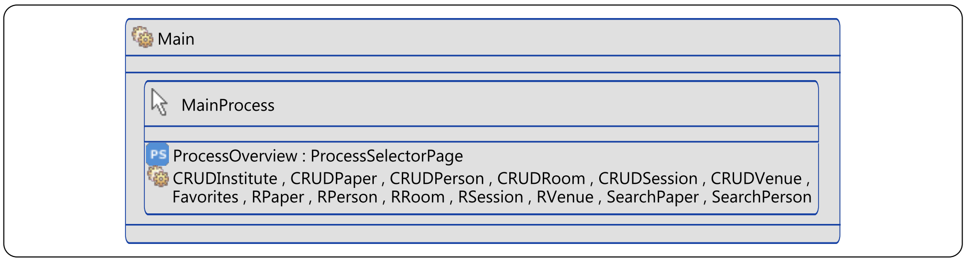


FIGURE B.18: Process *Main*

B.2.3.2 Process *Favorites*

B.23 Process *Favorites*, referenced by *Main*, contains the processes *AddFavorite* and *RemoveFavorite*, which exemplifies a nested menu structure.

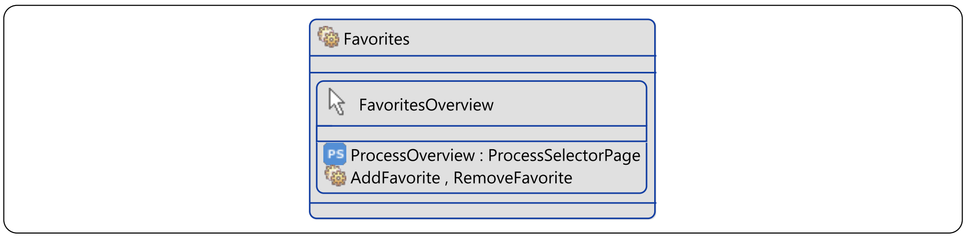
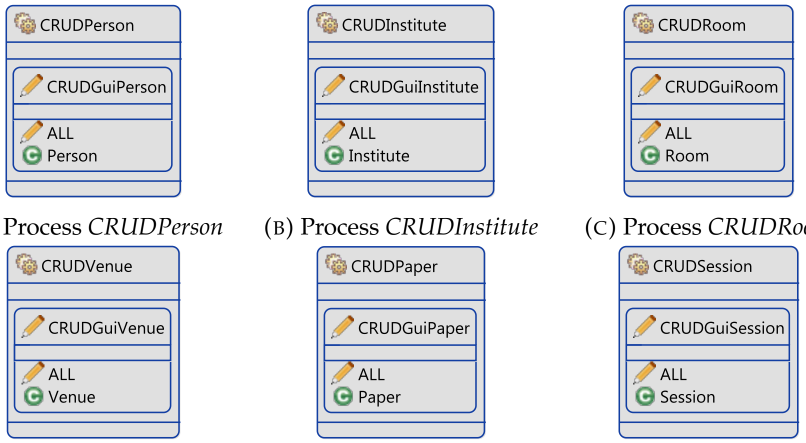


FIGURE B.19: Process *Favorites*

B.2.3.3 Standard CRUD Processes

CRUD processes for every entity of our data model are modeled. Each CRUD process contains only one task that is assigned to a class of the data model. E.g., *CRUDSession* contains a task that combines all CRUD activities w.r.t. entity class *Session*. Privileges are set to `ALL` here, enabling all CRUD operations. **B.24**



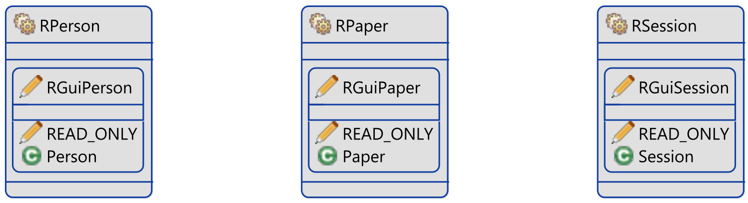
(A) Process *CRUDPerson* (B) Process *CRUDInstitute* (C) Process *CRUDRoom*

(D) Process *CRUDVenue* (E) Process *CRUDPaper* (F) Process *CRUDSession*

FIGURE B.20: Standard CRUD processes

B.2.3.4 Standard Read Processes

Full CRUD processes shall be available for conference administrators only. Therefore, corresponding read processes are provided for conference participants. A read process is very similar to a CRUD process, as described before. Of course, there are neither edit nor delete options according to the option `READ_ONLY`. **B.25**



(A) Process *RPerson* (B) Process *RPaper* (C) Process *RSession*

FIGURE B.21: Standard reading processes

B.2.3.5 Process *RRoom*

B.26 An instance of class *Room* may point to a plan which is a media dataset with a filename and path. This cannot be handled by the standard generation pattern and must therefore be handled explicitly in a separate operation that is manually coded. The custom process *RRoom* shows the allocated media files for the user-selected *Room* object.

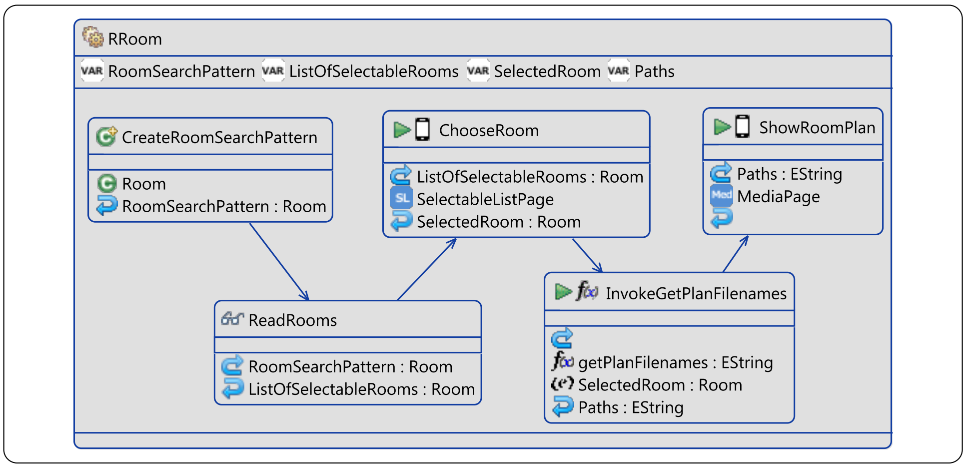


FIGURE B.22: Process *RRoom*

B.2.3.6 Process *RVenue*

B.27 An instance of class *Venue* holds address information. This information is to be displayed in Google Maps or with the Apple Map Service. This can also not be handled by the standard generation pattern and must be modeled as custom process.

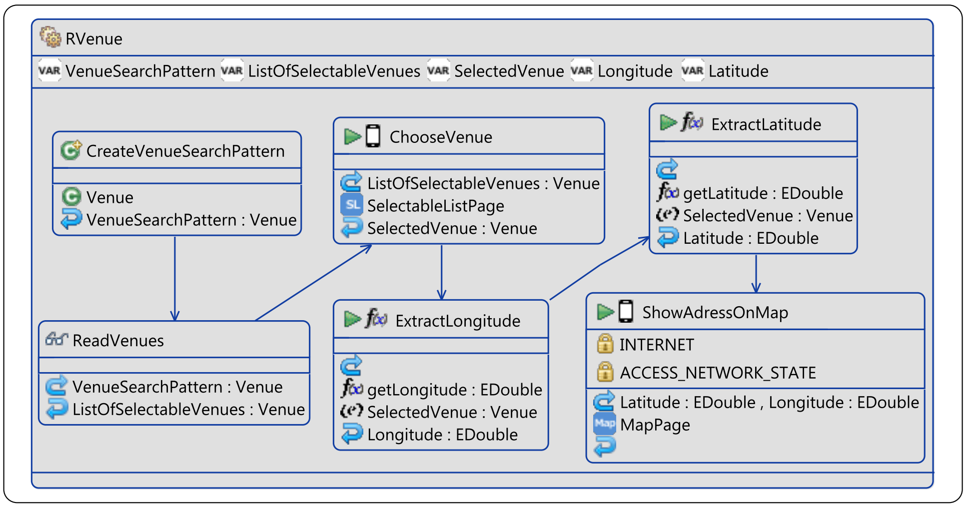


FIGURE B.23: Process *RVenue*

B.2.3.7 Process *AddFavorite*

The process *AddFavorite* uses a search pattern to find all *Session*s which are currently not select as favorite and display it on a *SelectableLisPage*. A mobile end user can select *Session* to set it as a Favorite. This invokes the method *addToFavorites* on the session object selected in *ChooseSession*. This method sets the attribute *isFavorite* to `true` and invokes an appointment in the user's calendar. Thus, the last task requires the access rights for the user's calendar. **B.28**

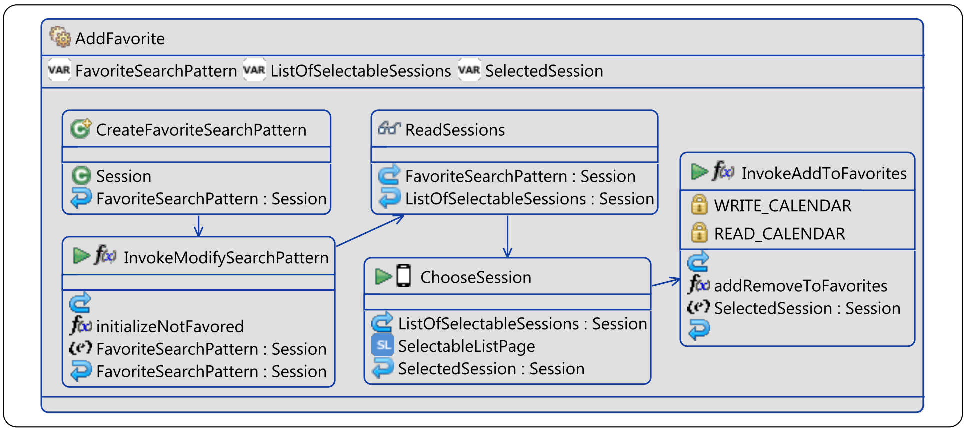


FIGURE B.24: Process *AddFavorite*

B.2.3.8 Process *RemoveFavorite*

The process *RemoveFavorite* works in an analogous way to the process *AddFavorite*. Favored *Session*s are display on a *SelectableListPage* and can be removed from the favorites as wells as their corresponding appointment in the calendar. **B.29**

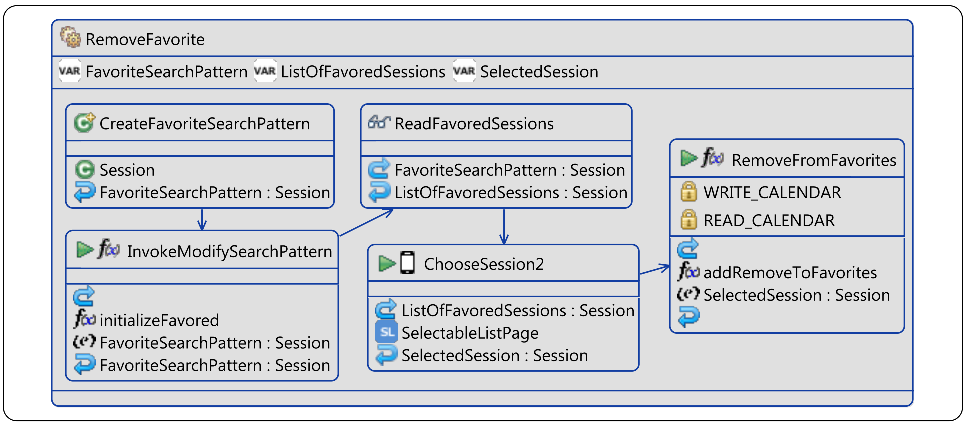


FIGURE B.25: Process *RemoveFavorite*

B.2.3.9 Process *SearchPaper*

B.30 The process *SearchPaper* provides functionality to find a paper. Hence, first the mobile end user edits a search pattern on a *EditPage*, then all matching paper objects are displayed on a *SelectableListPage*. Finally, the mobile end user can select one *Paper* object of the search result list for a detailed view on a *ViewPage*.

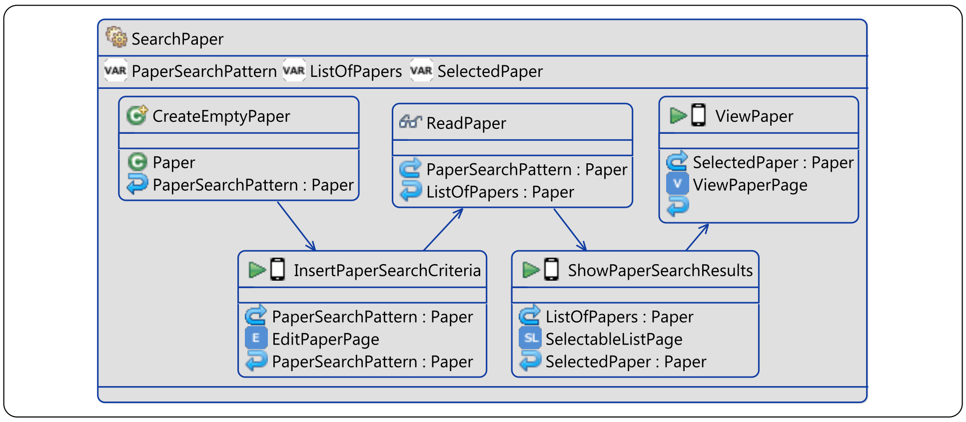


FIGURE B.26: Process *SearchPaper*

B.2.3.10 Process *SearchPerson*

B.31 The process *SearchPerson* works analogously to the process *SearchPaper*.

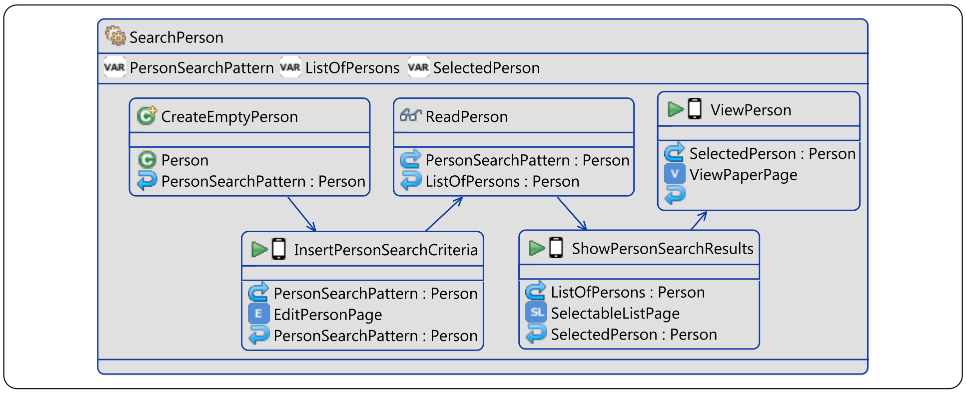


FIGURE B.27: Process *SearchPerson*

B.3 Word Trainer App Model

B.3.1 Data Model

The data model shown in Figure B.28 models a dictionary of words and related **B.32**
media files as well as additional grammatical and meta-information (e.g., part-of-speech, gender, topics, categories, priority). The *Word* class is the central class. It includes the word to be learned and an optional translation to the learner's mother language (given later by the learner). A word can be marked as favorite. It can have additional media. The classes *Picture*, *Audio* and *Video* store the file path, including the filename of the media. An optional imprint can be stored in the corresponding classes. The classes *Topic* and *Category* map a word to a topic or a category. In order to support test activities (especially multiple-choice tests), every word needs a number of choices or answers. The class *Answer* holds the answers, and at least two answers are required per design. The *Gender* class stores the gender of a word. The *PartofSpeech* class stores the part of speech for a word. The *Priority* class is an important meta-information. According to this value, a word occurs more or less in a particular test activity (e.g., writing, listening or reading).

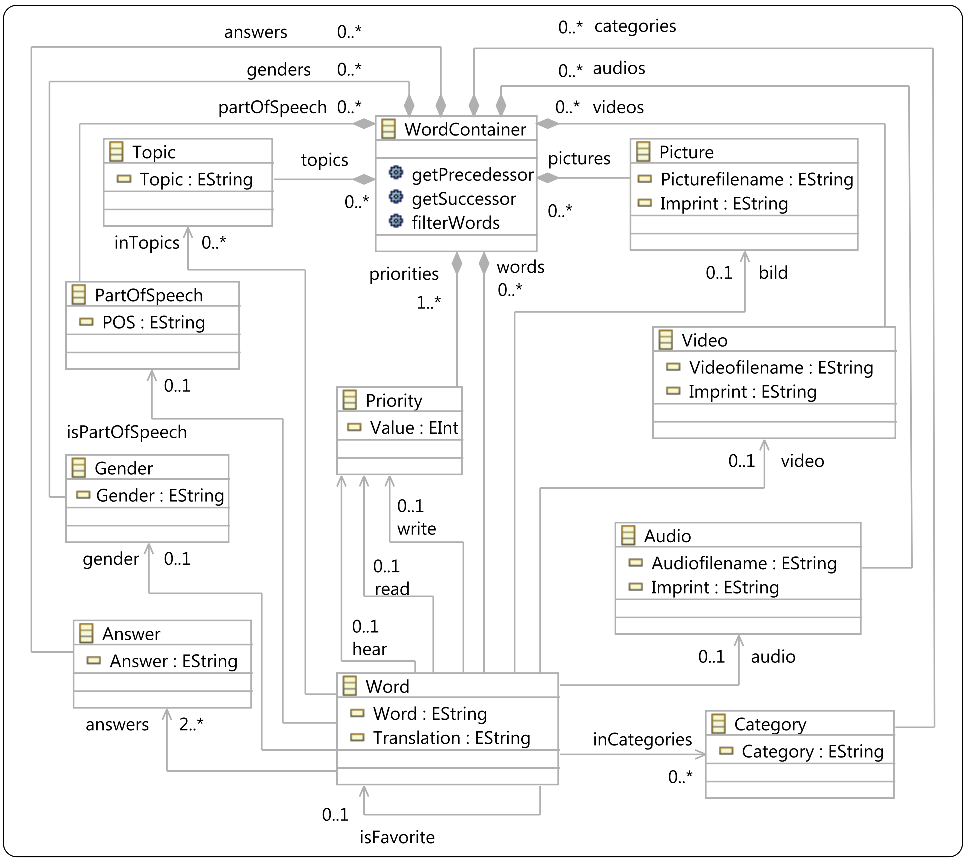


FIGURE B.28: Data model of word trainer application (excerpt)

B.3.2 GUI Model

B.33 The GUI model contains only standard pages. The page types *TakePicturePage* and *RecordAudioPage* provide access to the hardware facilities of a smartphone (e.g., built-in microphone and camera). The *eLearningPage* can be used in a generic way – i.e., both for learning and testing learning content. Similar as in the GUI model for the conference application (cf. Figure B.17), the GUI model for the word trainer app provides separate pages for each of the entities (e.g., *Word*, *Audio*, and *Answers*).

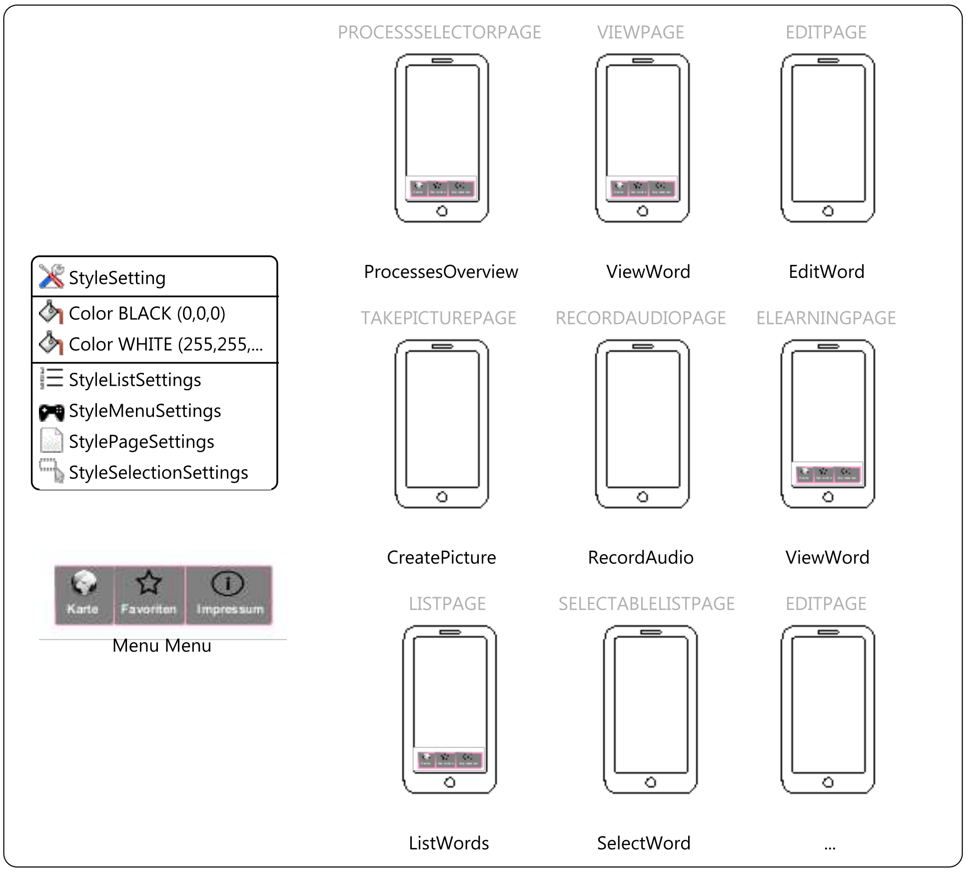


FIGURE B.29: GUI model of word trainer application (excerpt)

B.3.3 Process Model

The process model consists of 97 processes overall. Most of them are auxiliary processes which cannot be used directly. The standard CRUD processes are designated for the teacher variant of the mobile application. The learner variant provides processes for learning and testing learned content. Both variants share processes like the customized process for adding new words. **B.34**

B.3.3.1 Process *Main* and sub-processes

The *Main* process (Figure B.30) is the first one to be executed. It contains all process that a providing user or mobile end user can reach from the main screen of a mobile application. This configuration maybe reconfigured by a provider model or directly (e.g., using design time instantiation), as is shown in Figure 12.8. **B.35**

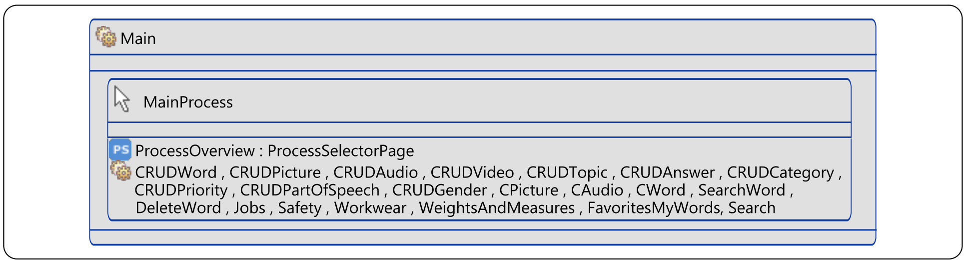
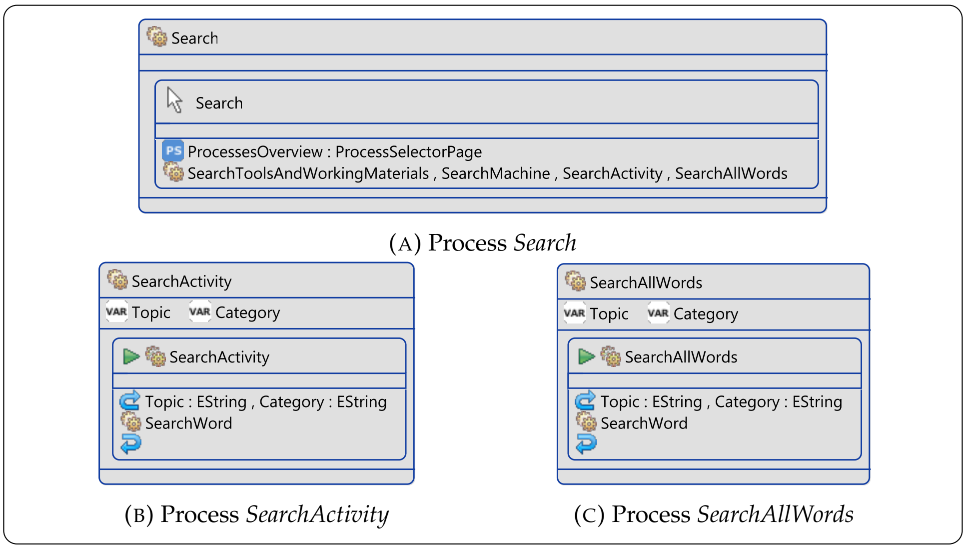


FIGURE B.30: Process *Main*

The process *Search* branches to the generic process *SearchWord* (B.3.3.7) but specifies the parameters *Topic* and *Category* before invoking the generic process. **B.36**



(A) Process *Search*

(B) Process *SearchActivity*

(C) Process *SearchAllWords*

FIGURE B.31: Process *Search* referring to category-specific/general search processes

As an example, Figure B.31 shows how a generic process for a job-specific process look like. A job-specific process (e.g., *Catering*) branches to sub-processes for learning and testing. Moreover, the testing sub-process branches to different test modes. The parameters *Testtype*, *Topic* and *Category* are used to instance the generic learning and testing processes (cf. B.3.3.10). **B.37**

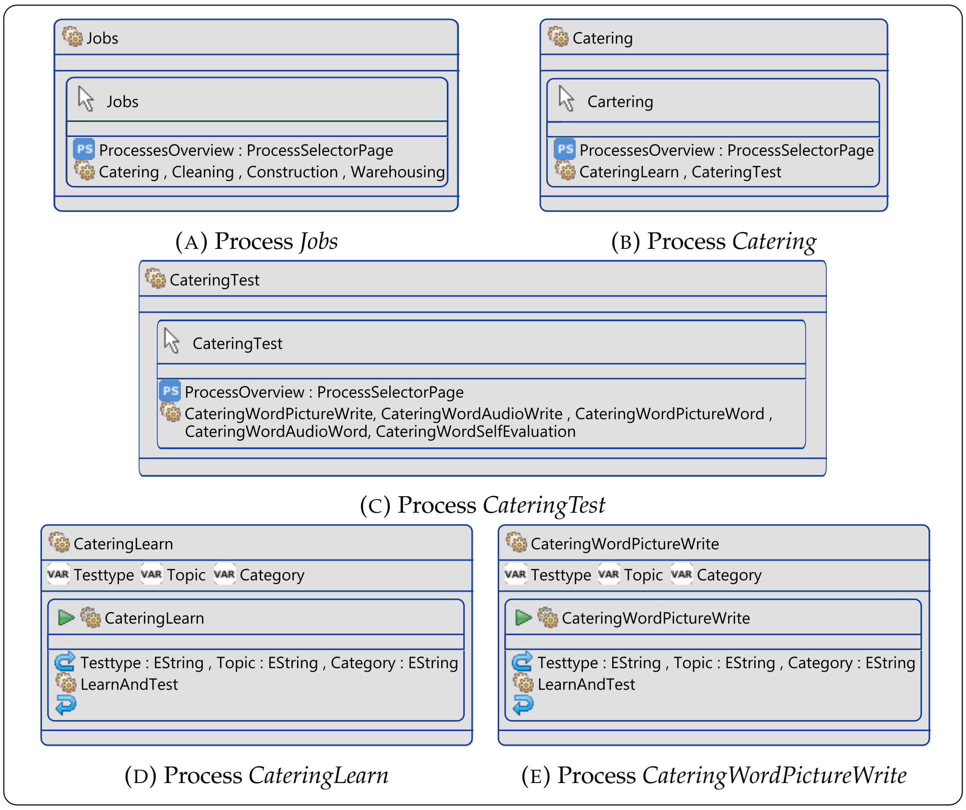
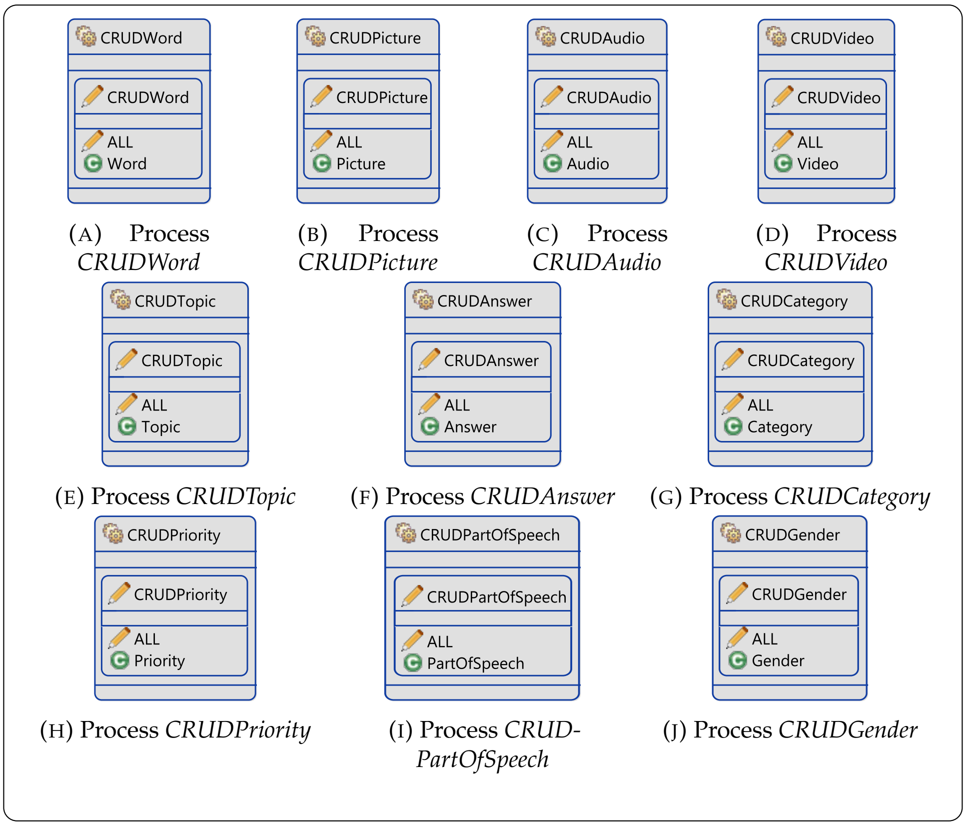


FIGURE B.32: Generic process structure *Jobs* (showing the *Catering* sub-processes)

B.3.3.2 Standard CRUD Processes

Full CRUD processes shall be available for the teacher variant of the word trainer application only. **B.38**



(A) Process *CRUDWord* (B) Process *CRUDPicture* (C) Process *CRUDAudio* (D) Process *CRUDVideo*

(E) Process *CRUDTopic* (F) Process *CRUDAnswer* (G) Process *CRUDCategory*

(H) Process *CRUDPriority* (I) Process *CRUDPartOfSpeech* (J) Process *CRUDGender*

FIGURE B.33: Standard reading processes

B.3.3.3 Process *CreatePicture*

The process *CreatePicture* creates a *Picture* object, delivers the chosen picture filename, and stores the taken picture at this location. The process requires different permissions to access the built-in camera as well as the file system. **B.39**

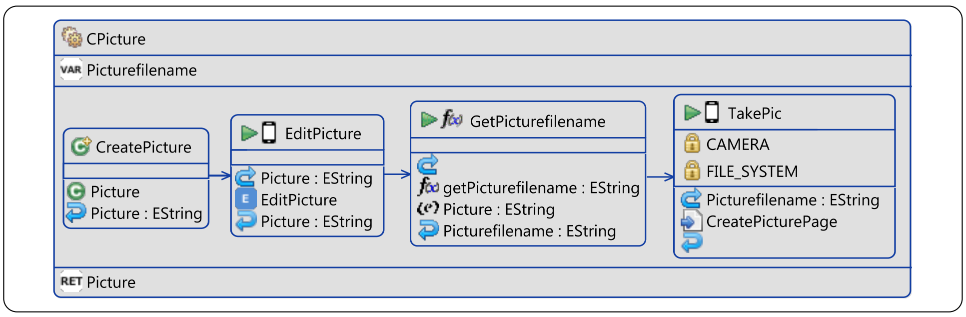


FIGURE B.34: Process *CreatePicture*

B.3.3.4 Process *CreateAudio*

B.40 The process *CreateAudio* creates an *Audio* object, delivers the chosen audio filename, and stores the recorded audio at this location. The process requires different permissions to access the built-in microphone as well as the file system.

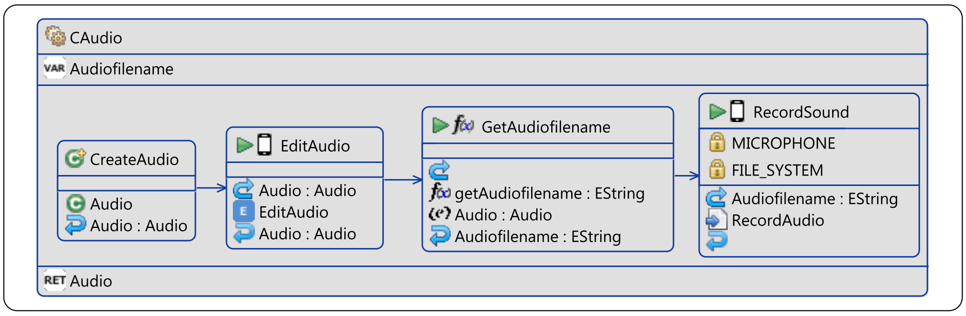


FIGURE B.35: Process *CreateAudio*

B.3.3.5 Process *CreateAnswer*

B.41 The process *CreateAnswer* is used inside the process *CreateWord* – B.3.3.6. It creates a new answer, provides user input by an *EditPage* named *EditAnswer* and finally adds the answer to the resulting *AnswerList*.

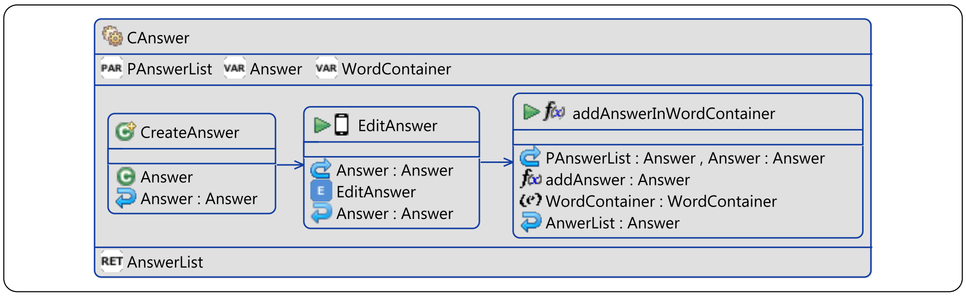


FIGURE B.36: Process *CAnswer*

B.3.3.6 Process *CreateWord*

B.42 The individual creation process *CreateWord* provides a more convenient version of the standard CRUD process for the corresponding entity *Word*. The process creates the mandatory *Word* object and add it to the user-specific vocabulary (Topic=OWN). Subsequently, the user decides whether a picture/audio should be taken/recorded for this word and whether answers should be added.

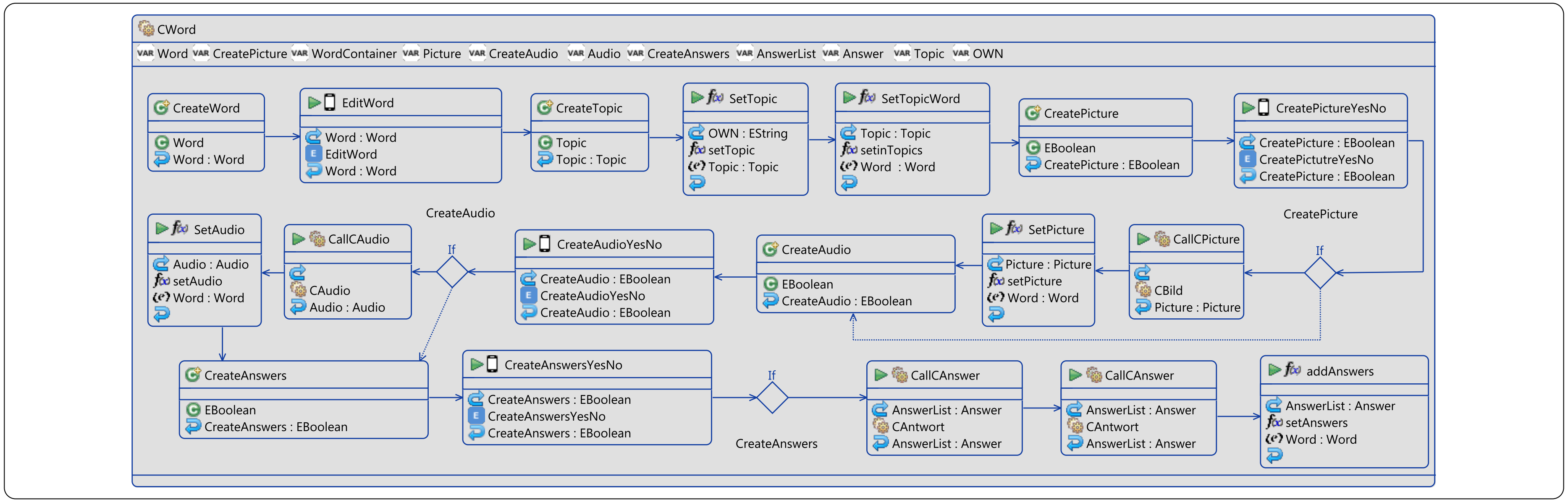


FIGURE B.37: Process *CWord*

B.3.3.7 Process *SearchWord*

B.43 The process *SearchWord* supports learners in finding a word in the vocabulary. The process creates an empty search pattern which subsequently can be modified by the mobile users, provides a selection from the search results and shows the selected word by calling the sub-process *ViewWord* – B.3.3.8.

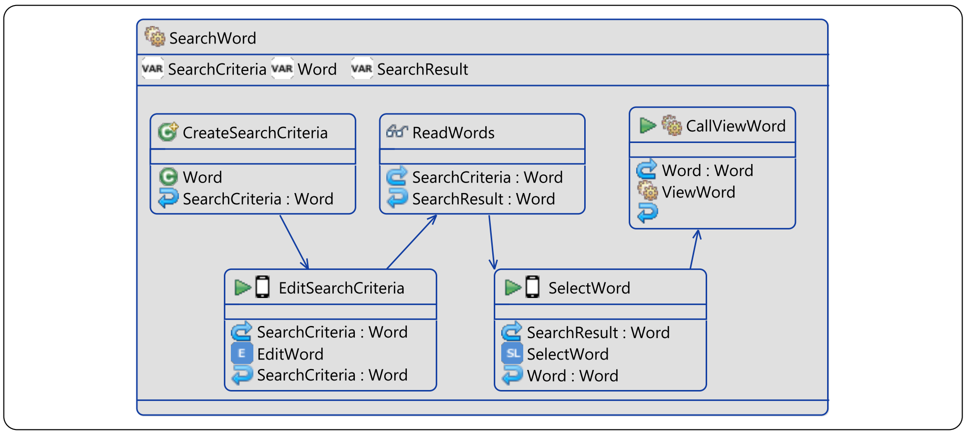


FIGURE B.38: Process *SearchWord*

B.3.3.8 Process *ViewWord*

B.44 The process *ViewWord* gets a *Word* object as parameter. Due to different context of usage, the process *ViewWord* decides whether the following word should be displayed or the viewing mode should be quit. The first task of the process extracts various attribute values. The *InvokeGUI* task, named *ViewWord*, displays the attribute values. The mobile user may add the displayed word to his/her favorites. Thus, the value of *Favorite* determines whether the word is set to a favorite or not.

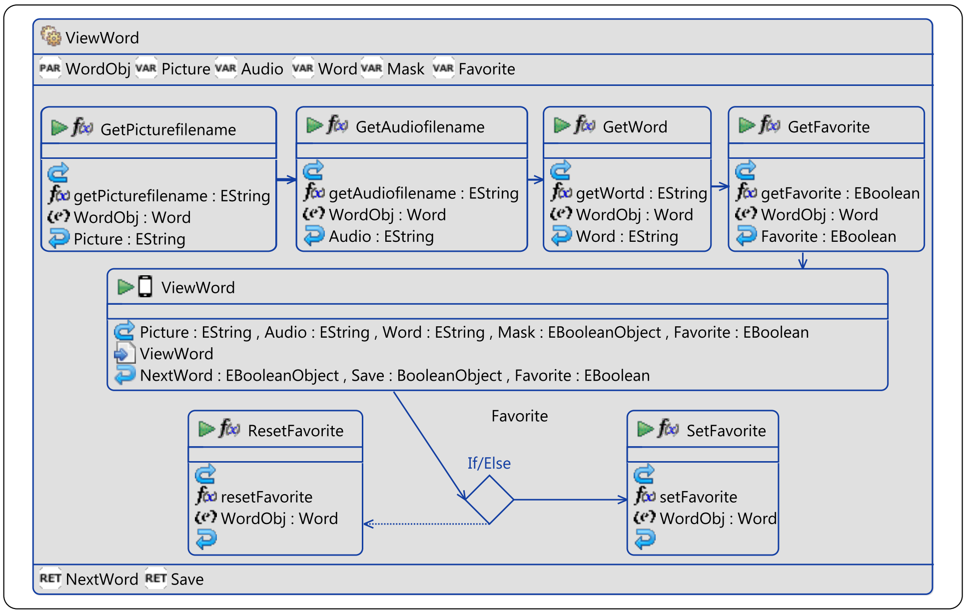


FIGURE B.39: Process *ViewWord*

B.3.3.9 Process *DeleteWord*

The process *DeleteWord* reads all *Word* objects from the vocabulary and filter the resulting set according to the parameters *Topic* and *Category*. The mobile application user selects a word from this filtered result list. Based on whether the word is from the personal or the preset vocabulary, the word will be deleted or only removed from the favorite list of the learner. **B.45**

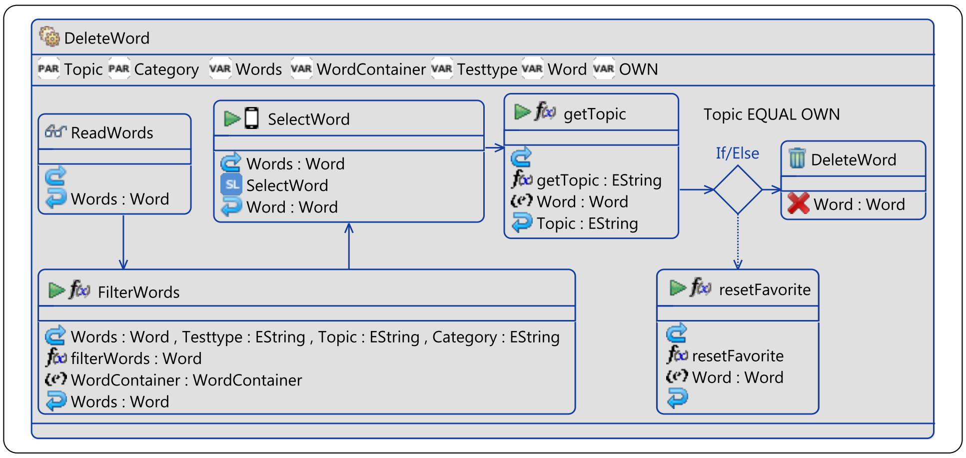


FIGURE B.40: Process *DeleteWord*

B.3.3.10 Processes *LearnAndTest*

B.46 At first, the generic learn and test process *LearnAndTest* filters the vocabulary according to the given parameters *Topic* and *Category*. When the mobile end user does not exit the process (Save=`true`), the successor or precessor of the filtered word list will be presented according to the selected presentation mode (i.e., Testtype).

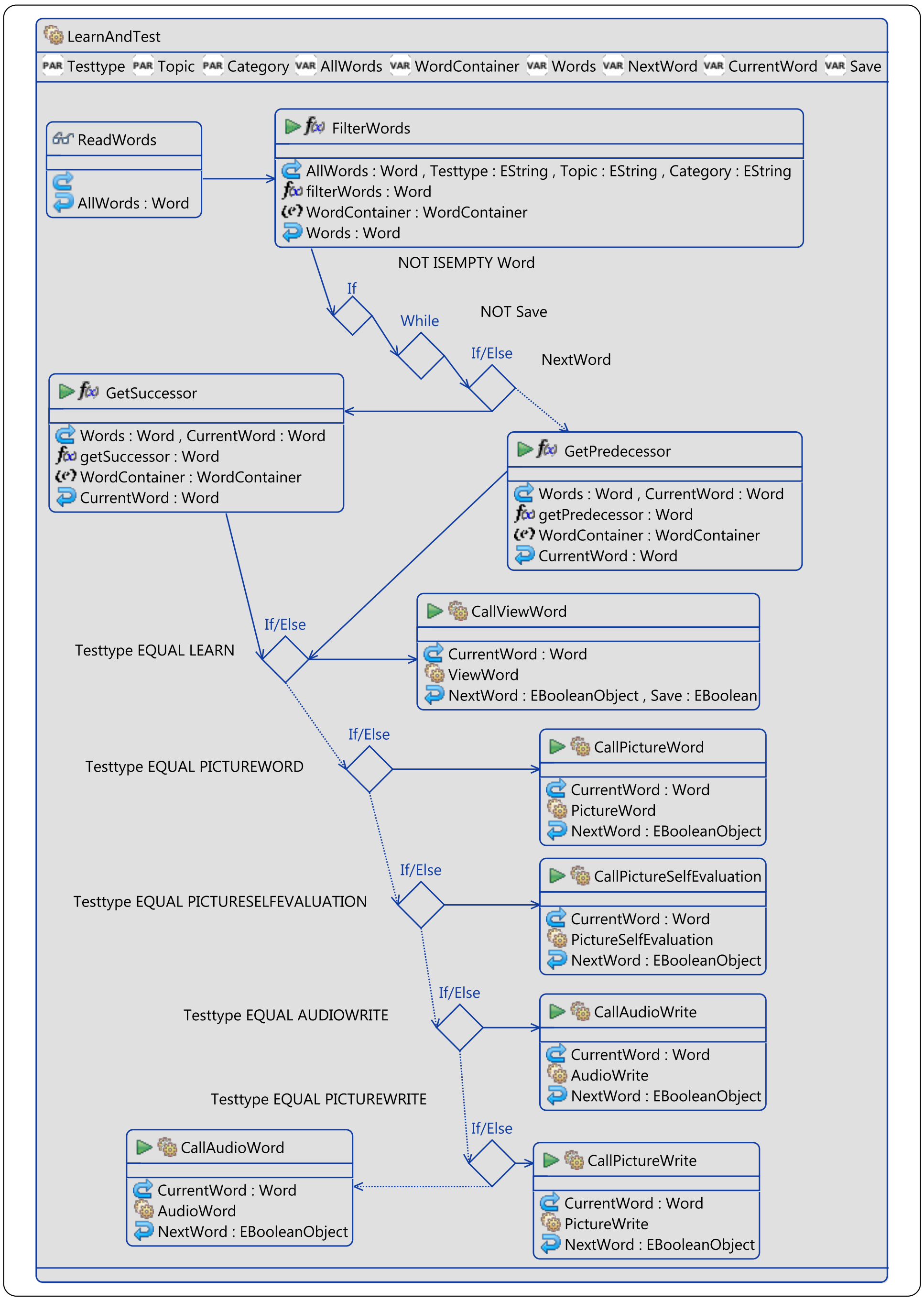
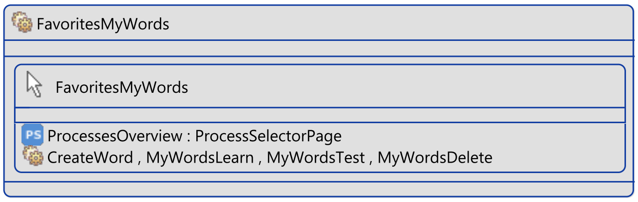


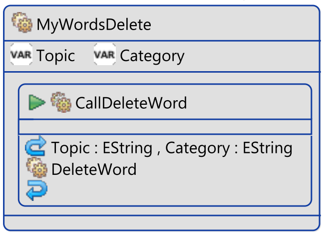
FIGURE B.41: Process *LearnAndTest*

B.3.3.11 Process *Favorites/MyWord*

The process *Favorites/MyWord* provides sub-processes to manage the Favorites and **B.47**
the individual vocabulary (Figure B.42a) of the learners. The process *CreateWord* offers the creation of new words. The process *MyWordsLearn* and *MyWordsTest* are sub-processes, which call the generic learning and testing process using the parameters for showing only content from the individual vocabulary. Similarly, the process *MyWordsDelete* refers to a parameterized version of the generic process *Delete Word* – B.3.3.9.



(A) Process *FavoritesMyWords*



(B) Process *MyWordsDelete*

FIGURE B.42: Process *FavoritesMyWords* and sub-processes

Appendix C

Tutorial

The developed tutorial was primarily used during different workshops where participants got an introduction in the designed model-driven development infrastructure. The workshop and the material are designed as practical training with hands-on assignments. The tutorial may also be used without an instructor since it is online available (Online Tutorial [73]). **C.1**

The tutorial consists of twelve parts and a workshop will regularly take one day. The participants re-model the phone book application shown in Appendix B.1. The instructor presents introductorily the use cases of the phone book application followed by the installation of the model-driven development infrastructure and the required software development kits, emulators, etc. The first part of the tutorial will be completely presented by the instructor. The goal of this part is to show the overall process of model-driven development with the designed model-driven development infrastructure. The different sub-models and their purpose are explained and subsequently, the generation and execution of the corresponding mobile application are demonstrated. After that presentation, the participants process this part for their own. **C.2**

The second, third, and seventh part of the tutorial has similar content to practices the model-driven development process and the steps presented in the first part. **C.3**

Within the fourth part, the runtime features of the generated mobile application are presented. The participants create a new process *AllPersons* but will not register this process at design time. After the deployment of the generated mobile application, they will use the runtime process instance model to add the process *AllPersons*. **C.4**

Part five and nine of the tutorial are similar and present the runtime adaptation of the data and the graphical user interface. **C.5**

Part six of the tutorial presents the usage of the abstract CRUD modeling element. The detailed modeled processes from the prior parts could be substituted by the abstract version of these processes. **C.6**

Part eight and ten show how custom functionality could be added to the app model and thus to the generated mobile applications. Moreover, part ten shows how sub-processes could be called from a process. **C.7**

Finally, part twelve demonstrates how the codebase of a generated mobile application could be customized. However, this way of customization is only suitable when the model-driven development approach is used only to generate initially a software prototype. **C.8**

Besides the slides of the tutorial, the instructors provide prepared sets of app models after each part as a kind of sample solution. Thus large groups could be taught better, and participants with fewer skills could keep pace with the group. The sample solutions are also useful for self-study of the tutorial without an instructor. **C.9**

Table C.1 shows the conducted workshops, the taught material, and the number of participants. **C.10**

TABLE C.1: List of Workshops

Date	**Institution**	**Material**	**Participants**
14.02.2014	Advenco Consulting GmbH (Gießen, Germany)	Part 1-4	5
06/08.03.2014	Høgskolen i Bergen Bergen University College (Bergen, Norway)	Part 1–7	15
25.03.2014	Advenco Consulting GmbH (Gießen, Germany)	Part 1-12	5
19.05.2014	Berufsakademie Nordhessen gGmbh (Bad Wildungen, Germany)	Part 1-12	19
06.06.2014	Technische Hochschule Mittelhessen (Friedberg, Germany)	Part 1-12	7
13.06.2014	Advenco Consulting GmbH (Gießen, Germany)	Individual	5
29.10.2014	Philipps-Universität Marburg (Marburg, Germany)	Part 1-12	15
15.12.2014	University of Duisburg-Essen paluno - The Ruhr Institute for Software Technology (Essen, Germany)	Part 1-6	5
03.03.2015	Høgskolen i Bergen Bergen University College (Bergen, Norway)	Part 1–12	10
16.06.2015	Philipps-Universität Marburg (Marburg, Germany)	Part 1-12	4

PIMAR[1] Workshop

16.06.2015

Philipps-Universität Marburg (Marburg)

Steffen Vaupel (Philipps-Universität Marburg)

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

Agenda

- The Phone Book App (Introduction)
- Installation
- The Phone Book App
  - Modeling (Part 1)
    - Data modeling
    - GUI modeling
    - Process modeling („Create Person")
    - Code generation, build and execute the app

16.06.2015 PIMAR Workshop 2

Agenda

- Modeling (Part 2)
  - GUI modeling
  - Process modeling („Edit Person")
- Modeling (Part 3)
  - Process modeling („Delete Person")
- Runtime configuration (Part 4)
  - Process modeling („All Persons")
  - Process Instance Model
- Runtime configuration (Part 5)
  - Object modeling

16.06.2015 PIMAR Workshop 3

Agenda

- Modeling (Part 6)
  - CRUD modeling („Manage Persons")
- Modeling (Part 7)
  - Process modeling („Search Person")
- Add customized functionality (Part 8)
  - Data modeling
  - Process modeling („Call Person")
- Runtime configuration (Part 9)
  - Style model

16.06.2015 PIMAR Workshop 4

Agenda

- Add customized functionality (Part 10)[1]
  - Data modeling
  - Process modeling („Persons Location")
- Add customized functionality (Part 11)[1,2]
  - Data modeling
  - Process modeling („Near To Me")
- Customizing the generated Code (Part 12)

[1] Hardware device required
[2] GPS receiver required

16.06.2015 PIMAR Workshop 5

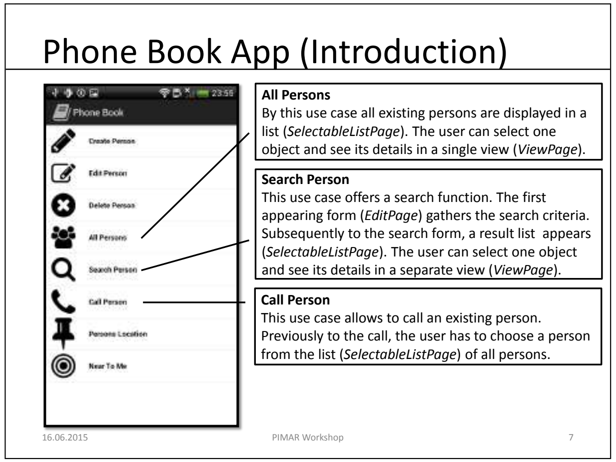
Phone Book App (Introduction)

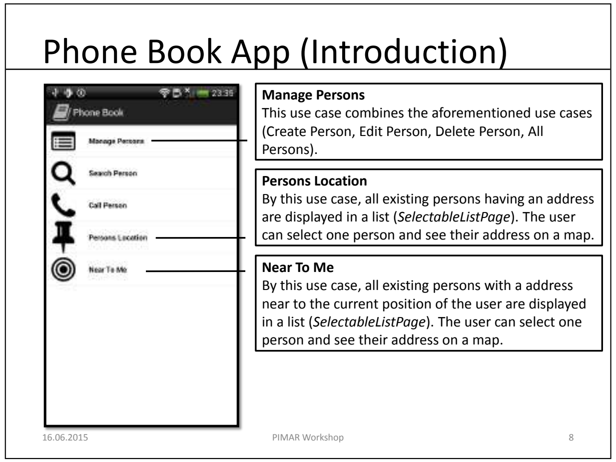
Create Person
This use case creates a new person. The first appearing form (*EditPage*) allows the user to enter the personal data of a person including the address. Subsequently, a non-editable form (*ViewPage*) displays the input data .

Edit Person
This use case allows to edit an existing Person object. Before editing, the user has to choose a person from a list (*SelectableListPage*). Subsequently, the edit form (*EditPage*) and a non-editable form (*ViewPage*) are shown.

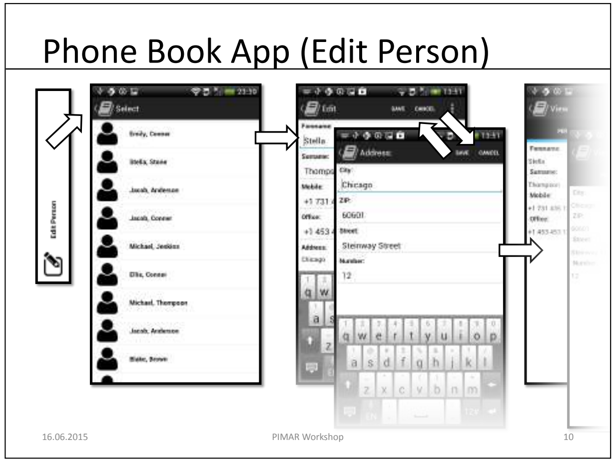
Delete Person
This use case allows to delete an existing Person object. Before deleting, the user has to choose a person from a list (*SelectableListPage*).

16.06.2015 PIMAR Workshop 6

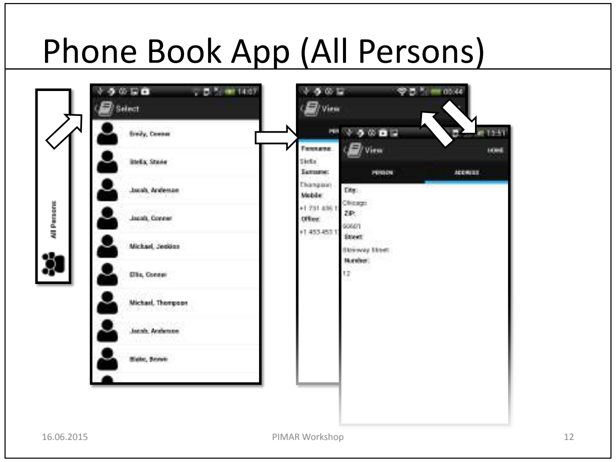

Phone Book App (Introduction)
Phone Book
Create Person
Edit Person
Delete Person
All Persons
Search Person
Call Person
Persons Location
Near To Me
All Persons
By this use case all existing persons are displayed in a list (SelectableListPage). The user can select one object and see its details in a single view (ViewPage).
Search Person
This use case offers a search function. The first appearing form (EditPage) gathers the search criteria. Subsequently to the search form, a result list appears (SelectableListPage). The user can select one object and see its details in a separate view (ViewPage).
Call Person
This use case allows to call an existing person. Previously to the call, the user has to choose a person from the list (SelectableListPage) of all persons.
16.06.2015
PIMAR Workshop
7


Phone Book App (Introduction)
Phone Book
Manage Persons
Search Person
Call Person
Persons Location
Near To Me
Manage Persons
This use case combines the aforementioned use cases (Create Person, Edit Person, Delete Person, All Persons).
Persons Location
By this use case, all existing persons having an address are displayed in a list (SelectableListPage). The user can select one person and see their address on a map.
Near To Me
By this use case, all existing persons with a address near to the current position of the user are displayed in a list (SelectableListPage). The user can select one person and see their address on a map.
16.06.2015
PIMAR Workshop
8


Phone Book App (Create Person)
Create Person
Insert
Address
View
Chicago
60601
Steinway Street
12
16.06.2015
PIMAR Workshop
9

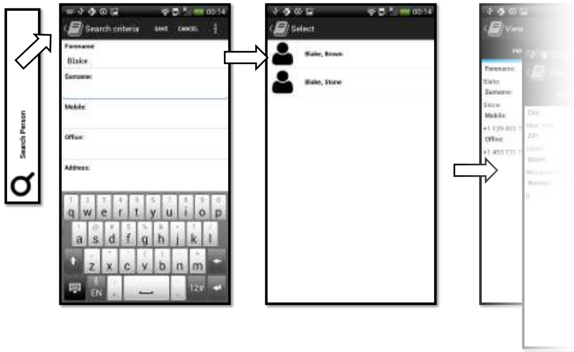

Phone Book App (Edit Person)
Edit Person
Select
Edit
Address
View
Stella
Chicago
60601
Steinway Street
12
16.06.2015
PIMAR Workshop
10


Phone Book App (Delete Person)
Delete Person
Select
Phone Book
16.06.2015
PIMAR Workshop
11


Phone Book App (All Persons)
All Persons
Select
View
16.06.2015
PIMAR Workshop
12

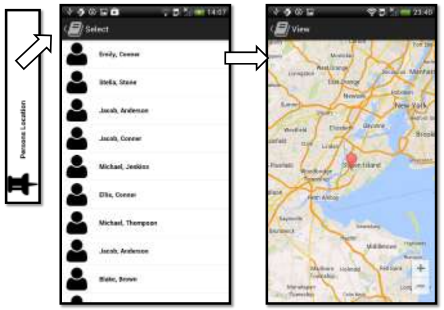
Phone Book App (Manage Persons)

Phone Book App (Search Person)

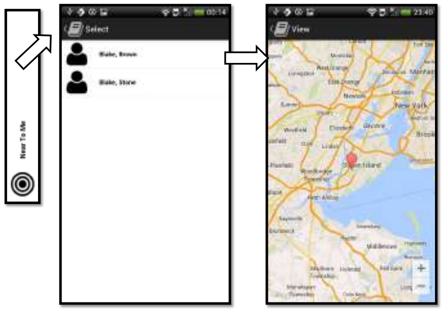


Phone Book App (Call Person)

Phone Book App (Persons Location)



Phone Book App (Near To Me)



Installation (Windows)

Please use the prepared USB sticks (maybe in groups of two).

You can work on the prepared USB sticks, if you don't want to copy the full installation to your hard disk.

Please execute *<mnt>:\eclipse-modeling-kepler-SR2-win32-x86_64\eclipse\eclipse.exe* and choose *<mnt>:\Workspace. Check (Eclipse >Window>Preferences: Android) if the SDK location conforms with your mount letter of the USB stick.*

<u>Pre-requisites:</u> JDK 1.7 (approx. 2.5 GB hard disk space (Full installation) / approx. 250 MB hard disk space (only AVD)).

Installation (others)

You need

- the GMF Runtime (1.7.0), GMF Notation (1.7.0) , GMF Tooling (GMF SDK) (3.1.0) from the Kepler Update-Site
- and the „Developer Tools“ from the ADT Update-Site
- and Xtext 2.5.1 and Xtend 2.5.1 from the Xtext Update-Site.

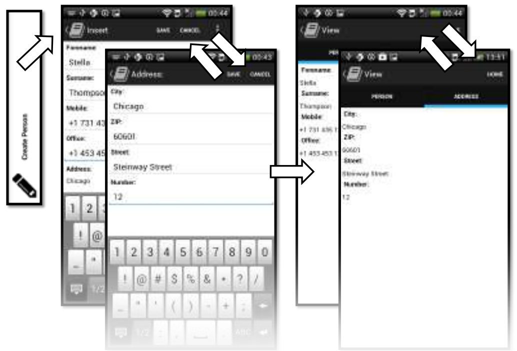
Please install also

- API Level 15 (Android 4.0.3)
- API Level 8 (Android 2.2)

with the Android SDK Manager.

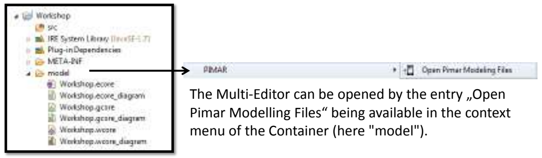
Modeling (Part 1)

Phone Book App (Create Person)



Modeling (Part 1)

Please create a new EMF Project (Empty EMF Project) with name "Workshop". By the wizard "Multi-page Editor Files" (Multi-page Editor Files) you can create a PIMAR Modeling Project in the container "Workshop/model/". Please name it "Workshop“ as well. The project structure should look like the one below:

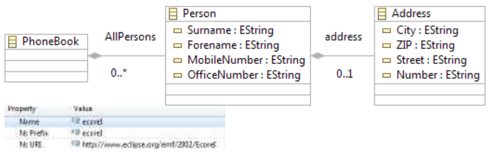


The Multi-Editor can be opened by the entry „Open Pimar Modelling Files“ being available in the context menu of the Container (here "model").

Data modeling (Part 1)

Select the first tab „*.ecore_diagram“ of the Pimar Modeling Editor to create the data model.

Model the following data structure of the Phone Book App:

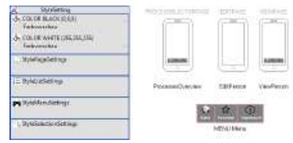


GUI modeling (Part 1)

Select the third tab „*.gcore_diagram“ of the Pimar Modeling Editor to create the GUI model.

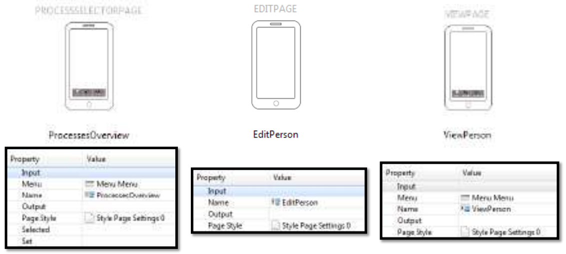
Create a *ProcessSelectorPage*, an *EditPage* and a *ViewPage*. Additionally to the existing *StyleSettings* add a menu.

The EditPage is used for gathering the data (Person) and the ViewPage is used to display the entered object afterwards.



GUI modeling (Part 1)

Set the page properties as shown below (empty fields are unused):

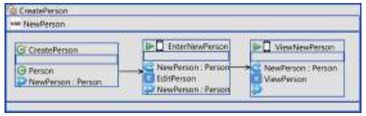


16.06.2015 PIMAR Workshop 25

Process modeling (Part 1)

Select the second tab „*.wcore_diagram"of the Pimar Modeling Editor to create the Process model.

The process „Create Person" (CreatePerson)[1] consists of the following tasks: *CreatePerson (Create)*, *EnterNewPerson (InvokeGUI)* and *ViewNewPerson (InvokeGUI)*

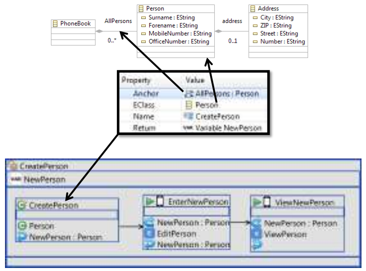


[1] Were refer usually to the process label e.g. „Create Person" instead of the technical name „CreatePerson"

16.06.2015 PIMAR Workshop 26

Process modeling (Part 1)

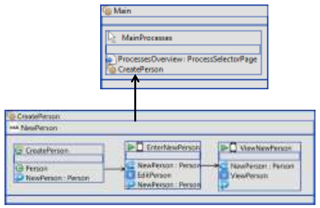
Where shall the new object be attached?



16.06.2015 PIMAR Workshop 27

Process modeling (Part 1)

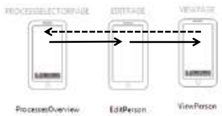
Register the process „Create Person" (CreatePerson) in a *ProcessSelector*. The *ProcessSelector* itself has to be in a Process named „Main" .



16.06.2015 PIMAR Workshop 28

Process modeling (Part 1)

Site map from the user's point of view:



16.06.2015 PIMAR Workshop 29

Code generation

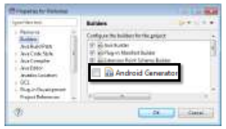
- Activate the Generator (Builder) by the context menu entry: *Configure > Add Android Generator*

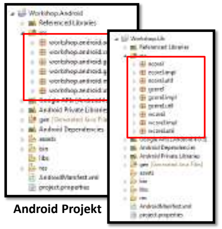
- The first „FULL BUILD" creates the empty Android projects only.
- Please change and save one of the *.ecore, *.wcore or *.gcore files to trigger an „AUTO BUILD" for code generation.

16.06.2015 PIMAR Workshop 30

Code generation

- The code generation is successful if the source folders of the projects *.Lib and *.Android are filled with packages and Java classes.
- You can also deactivate the generator.



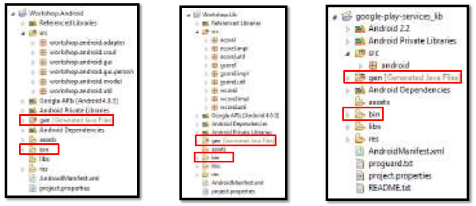


Android Projekt

Android Library Projekt (EMF)

Build

- The generated code is compiled automatically with all configured Android SDKs. The build is successfull if every project has the usual build artifacts in the folders *bin* and *gen*.



Android Projekt

Android Library Projekt (EMF)

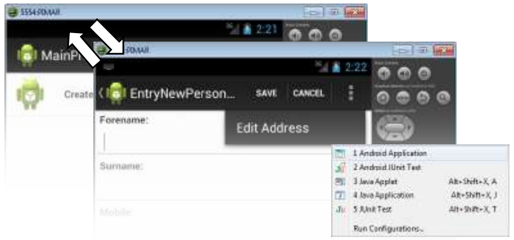
Android Library Projekt (Google Play Services)

Execute the app

- Please define a new virtual device by the *Android Virtual Device Manager*.
- Choose „Google APIs (Google Inc.) – API Level 15“ as target.
- Configure a SD card with approx. 200MB space.

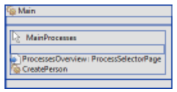
Execute the app

- Run the Workshop.Android project as *Android Application* on the virtual device.



Execute the app

- The app starts without data. The initial object model (My.ecorei) is empty.
- The app starts without special styles. The initial style model (My.gcorei) is empty.
- The app starts with an initial process instance model (My.wcorei). Only the processes registered in the *ProcessSelector* are available per default.



<processes ref="/mnt/sdcard/workshop.android/Workshop.wcore#//processContainerI/CreatePerson"/>

Execute the app

- Important: All models and instance models (shown below) must be deleted manually after a new generation and installation of the app.

Modeling (Part 2)

16.06.2015 PIMAR Workshop 37

Phone Book App (Edit Person)

16.06.2015 PIMAR Workshop 38

GUI modeling (Part 2)

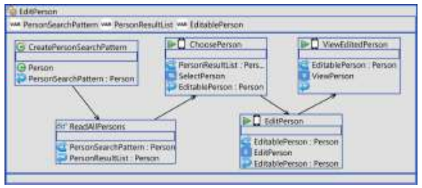
Within the second part, we can adopt the unchanged data model. Before editing a person (*EditPage*), the user has to choose a person from a list (*SelectableListPage*). So we extend the GUI model (Workshop.gcore) with a *SelectableListPage* being named "SelectPerson".

Additionally, set the corresponding attributes as shown below:

16.06.2015 PIMAR Workshop 39

Process modeling (Part 2)

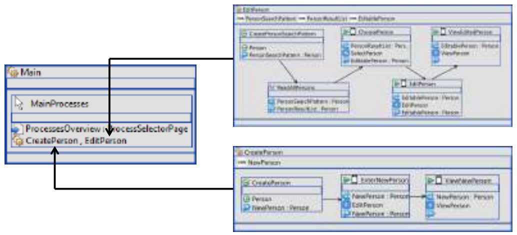
The process „Edit Person“ (EditPerson) consists of the following tasks: *CreatePersonSearchPattern (Create), ReadAllPersons (Read), ChoosePerson (InvokeGUI), EditPerson (InvokeGUI)* and *ViewEditedPerson (InvokeGUI)*



16.06.2015 PIMAR Workshop 40

Process modeling (Part 2)

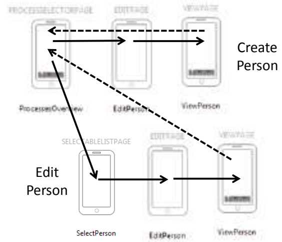
Again, we have to register the process „Edit Person“ (EditPerson) at the *ProcessSelector*.



16.06.2015 PIMAR Workshop 41

Process modeling (Part 2)

Site map from the user‘s point of view (multiply used pages are shown several times with different contexts):



16.06.2015 PIMAR Workshop 42

Modeling (Part 3)

16.06.2015 PIMAR Workshop 43

Phone Book App (Delete Person)

16.06.2015 PIMAR Workshop 44

Process modeling (Part 3)

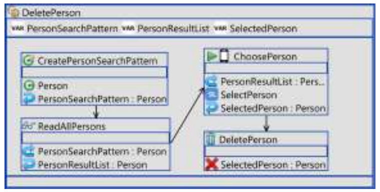
Within the third part we can adopt the unchanged data model and GUI model.

This use case allows to delete an existing Person object. Before deleting, the user has to choose a person from a list (*SelectableListPage*). The user can select one object and will be forwarded directly, currently without a confirm dialog, to the main menu (*ProcessSelectorPage*). The selected list item was removed between these dialogs.

16.06.2015 PIMAR Workshop 45

Process modeling (Part 3)

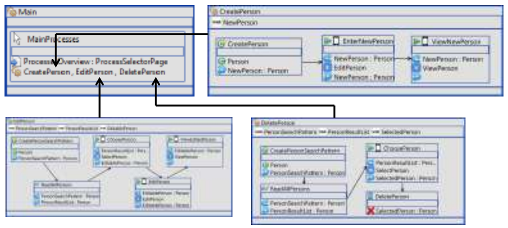
The process „Delete Person“ (DeletePerson) consists of the following tasks: *CreatePersonSearchPattern (Create), ReadAllPersons (Read), ChoosePerson (InvokeGUI)* and *DeletePerson (Delete)*



16.06.2015 PIMAR Workshop 46

Process modeling (Part 3)

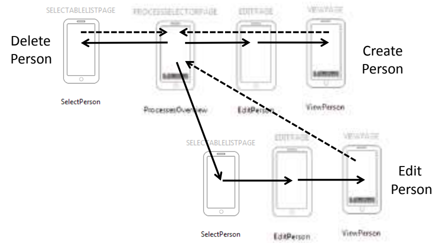
Again we have to register the process „Delete Person“ (DeletePerson) at the *ProcessSelector*.



16.06.2015 PIMAR Workshop 47

Process modeling (Part 3)

Site map from the user's point of view (multiply used pages are shown several times with different contexts):



16.06.2015 PIMAR Workshop 48

Runtime configuration (Part 4)

16.06.2015 PIMAR Workshop 49

Phone Book App (All Persons)

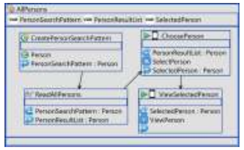
16.06.2015 PIMAR Workshop 50

Process modeling (Part 4)

Within this part we can adopt the unchanged data model and GUI model.

Process „All Persons“ (AllPersons) consists of the following tasks: *CreatePersonSearchPattern (Create), ReadAllPersons (Read), ChoosePerson (InvokeGUI) and ViewSelectedPerson (InvokeGUI).*

Exceptionally we do not register the process in the *ProcessSelector*. This process is not visible per default.



16.06.2015 PIMAR Workshop 51

Runtime configuration (Part 4)

- Please install and start the app.
- Export the default process instance model (My.wcorei) by using the export function in the File Explorer. Inspect the downloaded file. It should look like this:

```
<?xml …
<startTask xsi:type="wcore:ProcessSelector" name="MainProcesses"> …
<processes
ref="/mnt/sdcard/workshop.android/Workshop.wcore#//processContainerI/CreatePerson">
<processes
ref="/mnt/sdcard/workshop.android/Workshop.wcore#//processContainerI/EditPerson">
</startTask>
</instanceProcesses>
</wcoreI:processContainerI>
```

16.06.2015 PIMAR Workshop 52

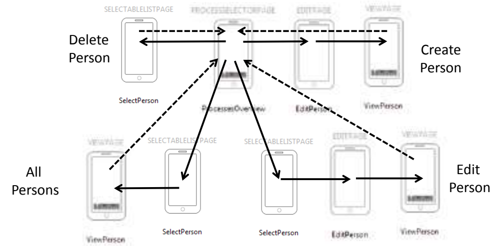
Runtime configuration (Part 4)

- Extend the process instance model carefully by hand (`<processes hef="/mnt/sdcard/workshop.android/Workshop.wcore#//processContainerI/AllPersons">`) or use the prepared object model My.wcorei of Part 4.
- Import the changed process instance model (My.wcorei) by using the import function in the File Explorer .
- Restart the app and check the available processes.
- Alternatively you can drop the process instance model in the folder “assets” of the project Workshop.Android and reinstall the app.

16.06.2015 PIMAR Workshop 53

Process modeling (Part 4)

Site map from the user‘s point of view (multiply used pages are shown several times with different contexts):



16.06.2015 PIMAR Workshop 54

Runtime configuration (Part 5)

Runtime configuration (Part 5)

- Please create at least one person in your app.
- Export the changed object model (My.ecorei) by using the export function in the File Explorer. Inspect the downloaded file. It should looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<ecoreI:PhoneBook xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:ecoreI="http://www.eclipse.org/emf/2002/EcoreI">
…
  <allPersons Surname="Conner" Forename="Michael" MobileNumber="+1 129 453 731"
OfficeNumber="+1 731 731 129">
    <address City="Chicago" ZIP="60601" Street="Woodrow Road"
Number="11"/></allPersons>
…
</ecoreI:PhoneBook>
```


Runtime configuration (Part 5)

- Extend the object model carefully by hand or use the prepared object model My.ecorei of Part 5.
- Import the changed object model (My.ecorei) by using the import function in the File Explorer .
- Restart the app and check the list of all persons.
- Alternatively you can drop the object model file into the folder “assets” of the project Workshop.Android and reinstall the app.

Modeling (Part 6)

Phone Book App (Manage Persons)

Process modeling (Part 6)

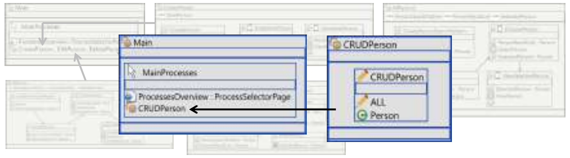
Within this part we can adopt the unchanged data model and GUI model (although we don’t need the pages at first).

The use case combines the aforementioned use cases (Create Person, Edit Person, Delete Person, All Persons).

The Process „Manage Persons“ (CRUDPerson) consists of the *CRUDPerson (CrudGui)* task only.

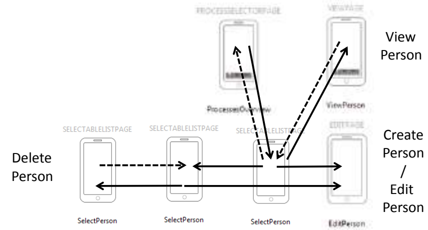
Process modeling (Part 6)

Now we have to unregister all customized CRUD processes namely „Create Person“, „Edit Person“, „Delete Persons“ but not „All Persons“ (because it was runtime configured). Instead we have to register the „Manage Persons“ process.



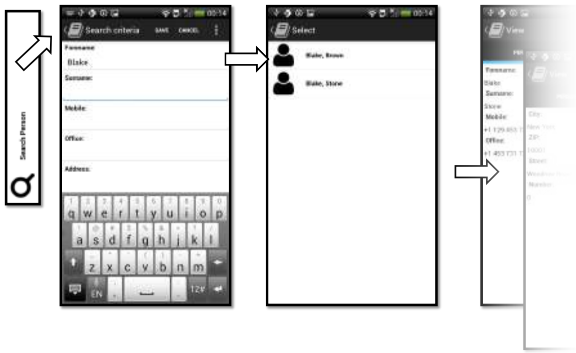
Process modeling (Part 6)

Site map from the user‘s point of view (multiply used pages are shown several times with different contexts):



Modeling (Part 7)

Phone Book App (Search Person)



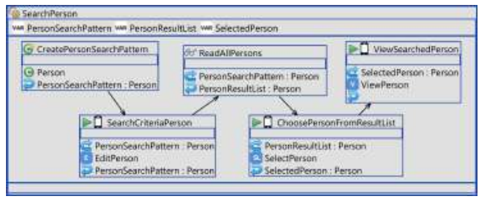
Process modeling (Part 7)

Within this part we can adopt the unchanged data model and GUI model.

The first appearing form (*EditPage*) gathers the search criteria. Subsequently to the search form, a result list appears (*SelectableListPage*). The user can select one object and see the its details in a separate view (*ViewPage*).

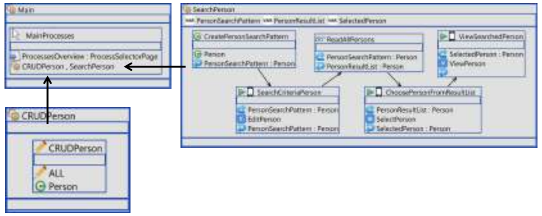
Process modeling (Part 7)

The process „Search Person“ consists of the following tasks: *CreatePersonSearchPattern (Create), SearchCriteriaPerson (InvokeGUI), ReadAllPersons (Read), ChoosePersonFromResultlist (InvokeGUI)* and *ViewSearchedPerson (InvokeGUI)*



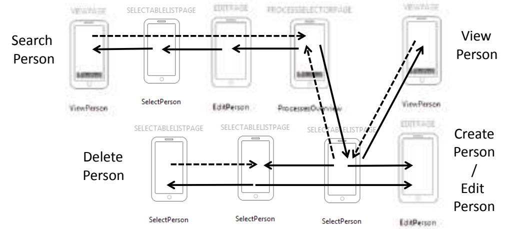
Process modeling (Part 7)

Again we have to register the process „Search Person“ (SearchPerson) at the *ProcessSelector*.



Process modeling (Part 7)

Site map from the user's point of view (multiply used pages are shown several times with different contexts):

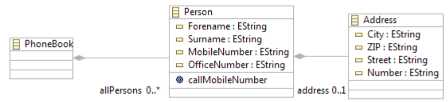


Add customized functionality (Part 8)

Phone Book App (Call Person)

Data modeling (Part 8)

We implement customized functionality as operations of classes of the data model (which is basically a common ecore model).

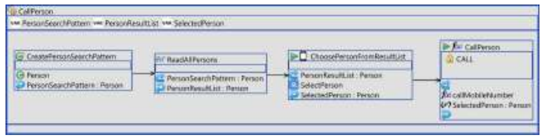


```
String uri = "tel:" + this.getMobileNumber();
android.content.Intent intent = new
android.content.Intent(android.content.Intent
.ACTION_CALL);
intent.setData(android.net.Uri.parse(uri));
intent.addFlags(android.content.Intent.
FLAG_ACTIVITY_NEW_TASK);
context.startActivity(intent);
```

* Source: http://www.eclipse.org/emf/2002/GenModel

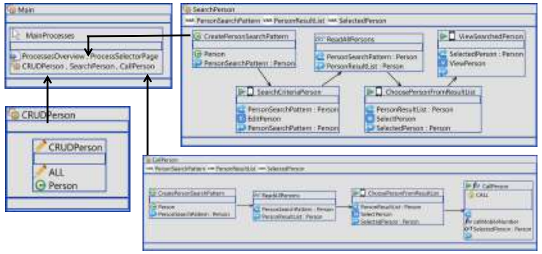
Process modeling (Part 8)

Process „Call Person“ (CallPerson) consists of the following tasks: *CreatePersonSearchPattern (Create), ReadAllPersons (Read), ChoosePerson (InvokeGUI)* and *CallPerson (InvokeOperation)*



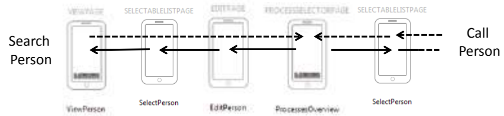
Process modeling (Part 8)

Again, we have to register the process „Call Person“ at the *ProcessSelector*.



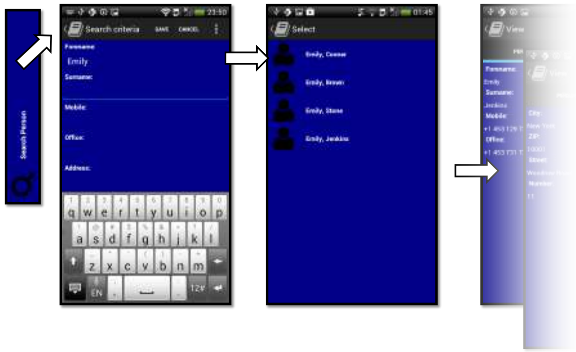
Process modeling (Part 8)

Site map from the user‘s point of view (multiply used pages are shown several times with different contexts) whithout CRUD processes:



Runtime configuration (Part 9)

Phone Book App (Search Person)



Runtime configuration (Part 9)

- Please install and start the app.
- Export the default style model (My.gcorei) by using the export function in the File Explorer. Inspect the downloaded file. It should look like this:

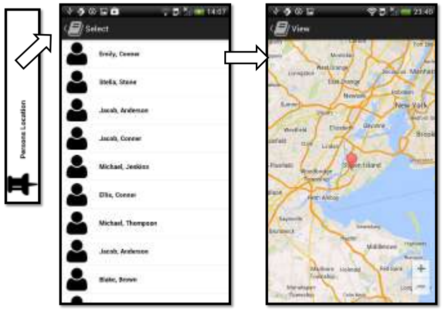
```
<?xml …
<rgbColors name="FRONTCOLOR"/>
<rgbColors name="BACKCOLOR" blue="255" red="255" green="255"/>
<menuStyle/>    <selectionStyle
…
</gcoreI:pageContainerI>
```


Runtime configuration (Part 9)

- Extend the style model carefully by hand (`<rgbColors name=„BLACK"` **`blue="255" red="0" green="0"`**`/>`) or use the prepared style model My.gcorei of Part 9.
- Import the changed style model (My.gcorei) by using the import function in the File Explorer.
- Restart the app and check the colors of an *EditPage*.
- Alternatively you can drop the process instance model in the folder “assets” of the project Workshop.Android and reinstall the app.

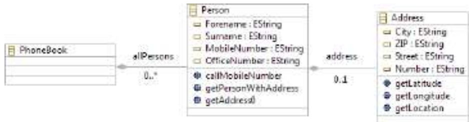
Add customized functionality (Part 10)

Phone Book App (Persons Location)



Data modeling (Part 10)

We implement customized functionality as operations of classes of the data model (which is basically a common ecore model).



- `Person getPersonWithAddress(Address address)`
- `Address getAddress0()`

- `EDouble getLatitude(); EDouble get Longitude();`
- `EJavaObject getLocation()`

GUI modeling (Part 10)

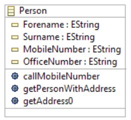
We extend the GUI model (Workshop.gcore) with a *MapPage* being named "ShowAddress".

Additionally, set the corresponding attributes as shown below:

Property	Value
Name	ShowAddress
Page Style	Style Page Settings 0

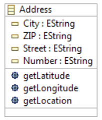
Process modeling (Part 10)

```
public Person getPersonWithAddress(final Context context,
    final Address address) {
    this.setAddress(address);
    return this;
}

public Address getAddress0(final Context context){
    return this.getAddress();
}
```



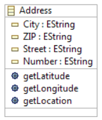
Process modeling (Part 10)

```
public Object getLocation(final Context context) {
    android.location.Geocoder coder = new
        android.location.Geocoder(context);
    java.util.List<android.location.Address> address = null;
    try {
        address = coder.getFromLocationName(this.getStreet() +  "  "
                                          + this.getNumber() +  "  "
                                          + this.getCity() +  "  "
                                          + this.getZIP(), 1);
    } catch (java.io.IOException e) {
        e.printStackTrace();
    }
    if (address.size()==0) {
         return null;
    } else {
        return (Object) address.get(0);
    }
}
```



Process modeling (Part 10)

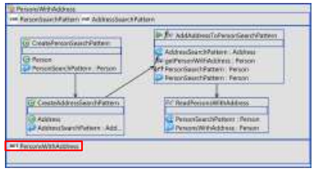
```
public double getLatitude(final Context context){
    android.location.Address location = null;
    location = ((android.location.Address)
        this.getLocation(context));
    if (location == null) {
       return 0;
    } else {
        return location.getLatitude();
    }
}

public double getLongitude(final Context context){
    ...
         return location.getLongitude();
    }
}
```



16.06.2015 PIMAR Workshop 85

Process modeling (Part 10)

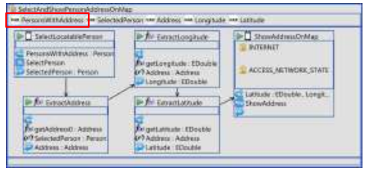
The auxiliary process „Person With Address" consists of the following tasks: *CreatePersonSearchPattern (Create), CreateAddressSearchPattern (Create), AddAddressToPersonSearchPattern (InvokeOperation)* and *ReadPersonWithAdddress (Read)*



16.06.2015 PIMAR Workshop 86

Process modeling (Part 10)

The auxiliary process „SelectAndShowPersonAddressOnMap" consists of the following tasks: *SelectLocatablePerson (InvokeGUI), ExtractAddress (InvokeOperation), ExtractLongitude (InvokeOperation), ExtractLatitude (InvokeOperation) and ShowAddressOnMap (InvokeGUI)*



16.06.2015 PIMAR Workshop 87

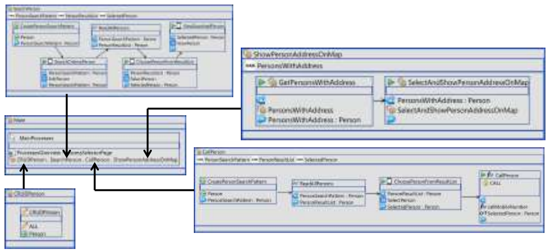
Process modeling (Part 10)

Process „Persons Location " (ShowPersonAddressOnMap) consists of the following tasks: *GetPersonsWithAddress (InvokeProcess)* and SelectAndShowPersonAddressOnMap *(InvokeProcess)*

16.06.2015 PIMAR Workshop 88

Process modeling (Part 10)

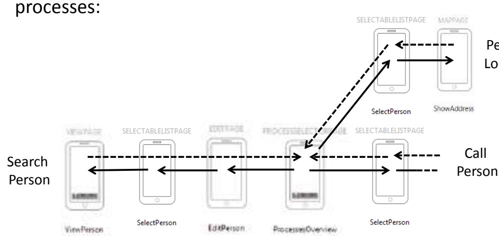
Again, we have to register the process „Persons Location" (ShowPersonAddressOnMap) at the *ProcessSelector*.



16.06.2015 PIMAR Workshop 89

Process modeling (Part 10)

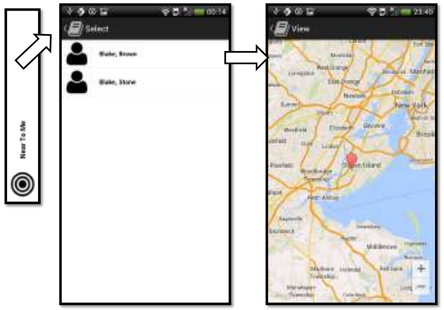
Site map from the user's point of view (multiply used pages are shown several times with different contexts) whithout CRUD processes:



16.06.2015 PIMAR Workshop 90

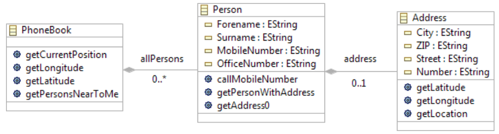
Add customized functionality (Part 11)

Phone Book App (Near To Me)



Data modeling (Part 11)

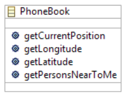
We implement customized functionality as operations of classes of the data model (which is basically a common ecore model).



- `EList<Person> getPersonsNearToMe(EJavaObject allPersons)`
- `EDouble get Longitude(); EDouble getLatitude()`
- `EJavaObject getCurrentPosition()`

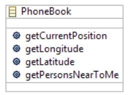
Process modeling (Part 11)

```
public Object getCurrentPosition(final Context context){

     class MyLocationListener implements
         android.location.LocationListener {
         public double latitude;
         public double longitude;

        public void onLocationChanged(
            android.location.Location loc){
            loc.getLatitude();
            loc.getLongitude();
            latitude=loc.getLatitude();
            longitude=loc.getLongitude();
        } …
     }
     android.location.LocationManager locationManager =
         (android.location.LocationManager)
         context.getSystemService(Context.LOCATION_SERVICE);
```



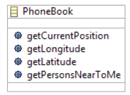
Process modeling (Part 11)

```
    android.location.LocationListener lmh = new
        MyLocationListener();

    String mlocProvider;
    android.location.Criteria hdCrit = new
        android.location.Criteria();

    hdCrit.setAccuracy(
        android.location.Criteria.ACCURACY_COARSE);
    mlocProvider = locationManager.getBestProvider(hdCrit,true);

   locationManager.requestLocationUpdates(
       android.location.LocationManager.GPS_PROVIDER,
       3000, 1000, lmh);
   android.location.Location currentLocation =
       locationManager.getLastKnownLocation(mlocProvider);
   locationManager.removeUpdates(lmh);
   return currentLocation;
}
```



Process modeling (Part 11)

```
public double getLongitude(final Context context){
    android.location.Location currentLocation =
        (android.location.Location)
    this.getCurrentPosition(context);
    return currentLocation.getLongitude();
}

public double getLatitude (final Context context){
    …
    return currentLocation.getLatitude();
}
```



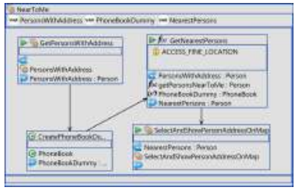
Process modeling (Part 11)

```
public EList<Person> getPersonsNearToMe(final Context context,
    final Object allPersons){
    EList<Person> personList = new
    org.eclipse.emf.common.util.BasicEList<Person>();
    java.util.Iterator it = ((java.util.List) allPersons).iterator();
    double diff = 90;
    double longitude = this.getLongitude(context);
    double latitude = this.getLatitude(context);

    while (it.hasNext()) {
        Person person = (Person) it.next();
        if (person.getAddress()!=null &&
             person.getAddress().getLongitude(context) >= longitude-diff &&
             person.getAddress().getLongitude(context) <= longitude+diff &&
             person.getAddress().getLatitude(context) >= latitude-diff &&
             person.getAddress().getLatitude(context) <= latitude+diff){
                personList.add(person);
        }
    }
    return personList;
}
```

16.06.2015 PIMAR Workshop 97

Process modeling (Part 11)

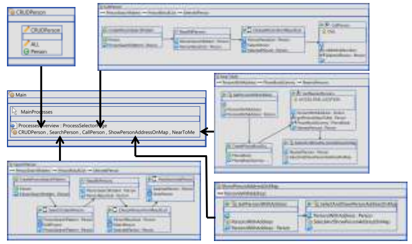
Process „Near To Me" (NearToMe) consists of the following tasks: *GetPersonWithAddress (InvokeProcess), CreatePhoneBookDummy (Create), GetNearestPersons (InvokeOperation)* and *SelectAndShowPersonsAddressOnMap (InvokeProcess)*



16.06.2015 PIMAR Workshop 98

Process modeling (Part 11)

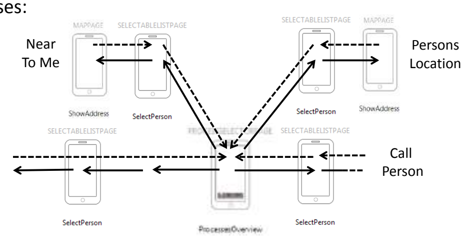
Again, we have to register the process „Near To Me" (NearToMe) at the *ProcessSelector*.



16.06.2015 PIMAR Workshop 99

Process modeling (Part 11)

Site map from the user's point of view (multiply used pages are shown several times with different contexts) whithout CRUD processes:



16.06.2015 PIMAR Workshop 100

Customizing the generated Code (Part 12)

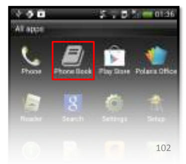
16.06.2015 PIMAR Workshop 101

Customizing the generated Code

Add custom App-Icon:

Put icon.png into Workshop.Android\res\drawable\ and change the following line in the AndroidManifest.xml:

```
<application android:allowBackup="true"
android:icon="@drawable/<icon>"
android:label="@string/app_name"
android:theme="@style/AppTheme">
```



16.06.2015 PIMAR Workshop 102

Customizing the generated Code

Add custom Process-Icons:

Put all icons into Workshop.Android\res\drawable\
and change the class ProcessAdapter.java as follows:

```
if (proc.getName().equals("EditPerson")){
    iv_icon.setImageResource(R.drawable.icon_pencil_square_o);
}
…
if (proc.getName().equals("DeletePerson")){
    iv_icon.setImageResource(R.drawable.icon_times_circle);
}
```

Appendix D

Miscellaneous

D.1 The Research Project *PIMAR*

This research project PIMAR (Platform Independent Mobile Augmented Reality)[1] **D.1**
is part of the "Hessen ModellProjekte" program, funded by the LOEWE - State Offensive for the Development of Scientific and Economic Excellence, funding line 3: promoting SME (Small and Medium-sized Enterprises) collaborative projects.

Mobile devices influence our life daily, affecting both the work and the leisure **D.2**
sector. The use of these systems goes far beyond applications with classical human-computer interaction (keyboard or mouse input). Applications with automatic localization of the user or image and sound recognition open up new possibilities – the so-called augmented reality.

The goals of the project are as follows: **D.3**

- Create a model-driven development infrastructure for cross-platform development of applications for mobile devices while avoiding multiple developments as far as possible.
- Realization of image recognition processes that operate even without a permanent internet connection. Implementation of these processes based on the model-driven development infrastructure created.
- Realization of a mobile system whereby maintenance and assembly activities can be made safer and users can be better protected. Therefore, it is essential to identify machinery and equipment accurately and provide handling information at the right time and at the right place.
- Development of algorithms for automatic detection and selection of AR technologies to be used (e.g., computer vision, compass). These should be suitable for both indoor and outdoor use.

The consortium comprises the Department MND (Mathematics, Natural science, **D.4**
Data science) of the *Technischen Hochschule Mittelhessen* (THM), *advenco Consulting GmbH* in Giessen, and the Department of Computer Science of the *Philipps-University* in Marburg. The THM has many years of experience in software development, computer graphics, and image processing, while advenco provides software solutions for manufacturing companies that use mobile technologies productively. Philipps-University has proven expertise in the field of model-driven software development.

[1] HA - Project no.: 355/12-45

D.2 The Mobile Application *key2guide*

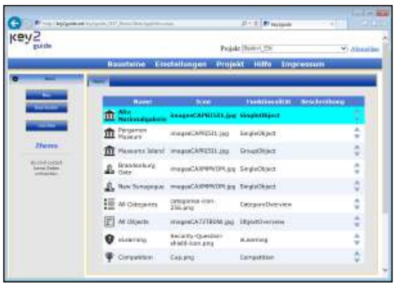
D.5 The mobile application *key2guide*, developed by the *advenco Consulting GmbH*, is a mobile multimedia guide for a wide range of applications. Cities and municipalities, exhibition and conference organizers, museums and tourist facilities, besides as many other user groups, can use this application to create their own multimedia guides as a mobile applications for smartphones and tablets.

D.6 The novelty of this mobile application lies in the self-service administration by its customers. The mentioned institutions and organizations can use a user-friendly web-based content management system (i.e., back end) to maintain the content of the mobile application (i.e., front-end). The content management system provides the configuration of general application settings. The look (e.g., colors, menu styles) of mobile applications can be adapted to the customer's corporate design. Mobile application administrators may create new data objects (e.g., categories, collections, and objects), configure predefined functions (e.g., filtering, searching, and sorting), upload different kinds of media files (e.g., audio, video, and animations), compose tours that refer to different kinds of objects and create calendars, surveys, and lotteries.

D.7 The mobile application was realized for Android and iOS, while the content management system may be used for both application variants. Thus, two native implementations of the mobile application are available. The nativeness of the architecture enables standalone operation, while the mobile application is disconnected to the server-based content management system. Initially, the mobile application provides no data. Usually at the first startup, a mobile end user downloads a prepared project file from the content management system which contains all data records (stored in XML documents) and related files composed by the providers. Henceforth, the mobile application requires only an occasional network connection to update the replicated project or to upload data entered by the mobile end user (e.g., from a survey or lottery).

(A) *key2guide* (iOS)

(B) *key2guide* (Android)



(C) *key2guide* Content Management System (CMS)

FIGURE D.1: *key2guide* front-end variants and shared back end (CMS)

D.8 Figure D.1 shows the two front-end variants of the mobile application[2] and the shared content management system. The content management system can not only be used by different front ends of the same customer, but it also provides multiple projects for different customers.

2 The releases of *key2guide* are shipped customized for the respective customers, i.e., built-in credentials for the content management system, market place description.

D.3 The Mobile Application *key2operate*

D.9

The mobile application *key2operate*, developed by *advenco Consulting GmbH*, focuses on data acquisition during production and maintenance processes. Machines and production plants are often not fully connected to an automatic monitoring or production system due to technological or financial reasons. Hence, workers must collect the relevant information manually and fill out paper-based forms. *key2operate* constitutes an electronic version of these paper-based forms. The entered data are validated immediately and sent to a production management system. It can be connected to various production management systems. The mobile application works bi-directionally. Data can be acquired and sent to the production management system, or it can be retrieved by the mobile app from the production management system.

D.10

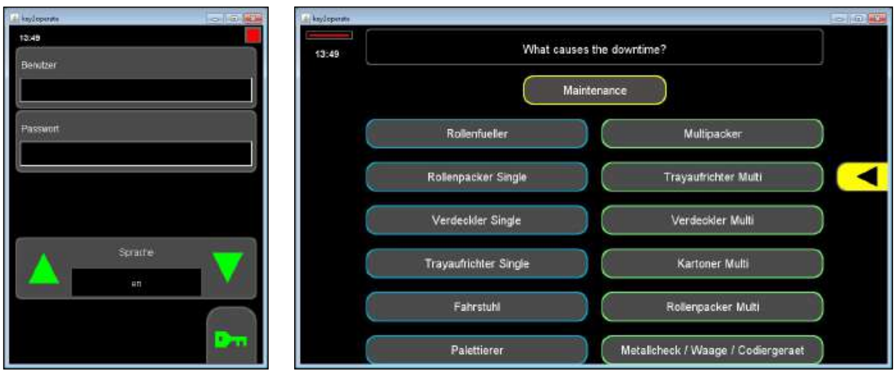
Due to the individuality of production processes, *key2operate* provides flexible data modeling as well as process modeling. Customers may create individual processes and corresponding graphical user interfaces (cf. Figure D.2). The mobile application works as an interpreter for such process descriptions. Listing D.1 gives a process description written in a domain-specific XML format. The general structure is as follows: *Workflows* contain *Processes*. In turn, *Processes* contains one or more *Forms*. These *Forms* contain *Widgets*. The behavior can be modeled by different scripts, allowing calculation and dynamic linking to other *Forms*. The layout of the graphical user interface is set by obligatory widget attributes (not shown).

LISTING D.1: Process description used in *key2operate*

```
<Workflows ... >
  <Strings><Str en="Request" key="3032" .../> ...</Strings>
 ...
<Process name="Begin" startProcess="true" ...>
  <Form name="OrderList" startForm="true" ...>
    <Widget name="OrderListO" actionScript="...␣if␣statusQuery
␣␣␣␣␣␣isEmpty␣isTrueThen␣@successor␣is␣"OrderStepList"
␣␣␣␣␣␣else␣..." "␣editable="true"␣...␣/>␣...
␣␣</Form>...
</Process>
</Workflows>
```

D.11

The mobile application was initially written for Java Micro Edition (J2ME) [51] [52], but soon there will be support for other platforms, e.g., Android (JSE) and iOS. The mobile app can work offline due to its nativeness and corresponding architectural features. It should be noted that offline capability is an essential requirement because, in many industrial settings, Wi-Fi coverage cannot be provided.



(A) Login screen (B) Downtime report screen

FIGURE D.2: Graphical user interface of *key2operate*

Bibliography

[01] Altova Mobile Together. https://www.altova.com/mobiletogether. (last downloaded 2018-07-13).

[02] Amazon Europe Core S.à.r.l. http://www.amazon.de. (last downloaded 2018-07-13).

[03] Apache Cordova. https://cordova.apache.org/. (last downloaded 2018-07-13).

[04] Appian. *Appian Mobile*. http://www.appian.com/cp/mobile-app-development. (last downloaded 2018-07-13).

[05] Applause. https://github.com/applause/applause. (last downloaded 2018-07-13).

[06] Apple Inc. *Apple Developer Connection*. https://developer.apple.com/devcenter/ios/. (last downloaded 2018-07-13).

[07] Apple Inc. *Apple Pay*. http://www.apple.com/apple-pay. (last downloaded 2018-07-13).

[08] arconsis IT-Solutions GmbH. *ModAgile*. http://www.modagile-mobile.de. (last downloaded 2016-04-26).

[09] Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. *Manifesto for agile software development*. http://users.jyu.fi/~mieijala/kandimateriaali/Agile-Manifesto.pdf. (last downloaded 2018-07-13).

[10] BookFit. http://www.bookfitapp.co.uk/. (last downloaded 2018-07-13).

[11] Google Inc. *Android Dashboards*. https://developer.android.com/about/dashboards/index.html. (last downloaded 2018-07-13).

[12] Google Inc. *Google Play Services*. http://developer.android.com/google/play-services. (last downloaded 2018-07-13).

[13] Google Inc. *In-App Billing*. http://developer.android.com/google/play/billing/index.html. (last downloaded 2018-07-13).

[14] Google Payment Corp. (GPC). *Google Wallet*. https://www.google.com/wallet/. (last downloaded 2018-07-13).

[15] GymJam. http://www.thegymjam.com/. (last downloaded 2018-07-13).

[16] GymSync. http://www.gymsync.co.uk/. (last downloaded 2018-07-13).

[17] Hwaci - Applied Software Research. *Sqlite*. https://www.sqlite.org/. (last downloaded 2018-07-13).

[18] International Business Machines Corp. *IBM DB2 Everyplace Installation and User's Guide*. ftp://ftp.software.ibm.com/software/data/db2/everyplace/doc/enu/iug.pdf. (last downloaded 2018-07-13).

[19] International Business Machines Corp. *IBM Rational Rhapsody*. www.ibm.com/software/awdtools/rhapsody/. (last downloaded 2018-07-13).

[20] Jean-Jacques Dubray. *Canappi MDSL*. https://github.com/canappi. (last downloaded 2018-07-13).

[21] Johan den Haan. *15 Reasons Why You Should Start Using Model Driven Development*. http://www.theenterprisearchitect.eu/blog/2009/11/25/. (last downloaded 2018-07-13).

[22] Koukaam. *Koukaam NETIO*. http://www.koukaam.se/. (last downloaded 2018-07-13).

[23] Massachusetts Institute of Technology. *App Inventor*. http://appinventor.mit.edu. (last downloaded 2018-07-13).

[24] Matthias Bohlen. *AndroMDA Version 3.3*. www.andromda.org. (last downloaded 2018-07-13).

[25] Mendix. *Mendix App Platform*. https://www.mendix.com/. (last downloaded 2018-07-13).

[26] Metaio Inc. *Metaio SDK*. http://www.metaio.com/sdk. (last downloaded 2015-05-28).

[27] Michael DeGusta. *Are Smart Phones Spreading Faster than Any Technology in Human History?* https://www.technologyreview.com/s/427787/are-smart-phones-spreading-faster-than-any-technology-in-human-history/. (last downloaded 2018-07-13).

[28] Microsoft Corp. *Microsoft – Beginner's Guide to SQL Server Compact*. https://msdn.microsoft.com/en-us/data/ff687144. (last downloaded 2018-07-13).

[29] Mobile World Live. *Visa deal eases PayPal's way into offline payment*. http://www.mobileworldlive.com/money/news-money/visa-deal-eases-paypals-way-into-offline-payments/. (last downloaded 2018-07-13).

[30] Morfeo Competence Center. *Morfeo MyMobileWeb*. https://web.archive.org/web/20130515132358/http://mymobileweb.morfeo-project.org:80/. (last downloaded 2018-07-13).

[31] Object Management Group (OMG) / Business Process Management Initiative (BPMI). *Business Process Model And Notation (BPMN) Version 2.0*. http://www.omg.org/spec/BPMN/2.0. (last downloaded 2018-07-13).

[32] Object Management Group (OMG). *Human-Usable Textual Notation Version 1.0*. http://www.omg.org/cgi-bin/doc?formal/2004-08-01. (last downloaded 2018-07-13).

[33] Object Management Group (OMG). *The Model-Driven Architecture, Guide Version 1.0.1*. www.omg.org/cgi-bin/doc?omg/03-06-01.pdf. (last downloaded 2018-07-13).

[34] Object Management Group (OMG). *MOF Core specification Ver. 2.4.1*. http://www.omg.org/spec/MOF/2.4.1/PDF/. (last downloaded 2018-07-13).

[35] Object Management Group (OMG). *Object Constraint Language (OCL) Version 2.4*. http://www.omg.org/spec/OCL/2.4. (last downloaded 2018-07-13).

[36] Object Management Group (OMG). *OMG Unified Modeling Language Infrastructure Specification - Version 2.4.1*. http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF. (last downloaded 2018-07-13).

[37] Object Management Group (OMG). *OMG Unified Modeling Language Superstructure Specification - Version 2.4.1*. http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF. (last downloaded 2018-07-13).

[38] Oracle Corp. *Oracle ® Database Lite Getting Started Guide*. https://docs.oracle.com/cd/E12095_01/doc.10303/e12080.pdf. (last downloaded 2018-07-13).

[39] Organization for the Advancement of Structured Information Standards (OASIS). *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. (last downloaded 2018-07-13).

[40] Otto (GmbH & Co KG). https://www.otto.de. (last downloaded 2018-07-13).

[41] Paul Boocock. *Jamda - Java Model Driven Architecture*. jamda.sourceforge.net/. (last downloaded 2018-07-13).

[42] PayPal. https://www.paypal.com. (last downloaded 2018-07-13).

[43] Philipps-Universität Marburg – Fachbereich 09 DaF-Abteilung. *Wörter für den Beruf*. https://play.google.com/store/apps/details?id=pkg.alphamar2app. (last downloaded 2018-07-13).

[44] Philipps-Universität Marburg – Fachbereich 12 Softwaretechnik /Technische Hochschule Mittelhessen – Fachbereich 08 (MND) KITE. *MoDELS2014: Conference Guide*. https://play.google.com/store/apps/details?id=modelsconference.android. (last downloaded 2018-07-13).

[45] Philipps-Universität Marburg – Fachbereich 12 Softwaretechnik /Technische Hochschule Mittelhessen – Fachbereich 08 (MND) KITE. *PhoneBook*. https://www.uni-marburg.de/fb12/arbeitsgruppen/swt/research/software/pimar1?language_sync=1#phone-book. (last downloaded 2018-07-13).

[46] Philipps-Universität Marburg – Fachbereich 12 Softwaretechnik. *Case examples*. http://www.uni-marburg.de/fb12/arbeitsgruppen/swt/research/esec17/caseexamples. (last downloaded 2018-07-13).

[47] SAP Mobile Platform 3.0. https://www.sap.com/trends/mobile-technology.html. (last downloaded 2018-07-13).

[48] StatCounter - Free Invisible Web Tracker, Hit Counter and Web Stats. *Desktop, mobile, and tablet comparison*. http://gs.statcounter.com/#desktop+mobile+tablet-comparison-ww-monthly-201610-201610-bar. (last downloaded 2018-07-13).

[49] Statista GmbH. *The Average Smartphone User Has Installed 26 Apps*. https://www.statista.com/chart/1435/top-10-countries-by-app-usage/. (last downloaded 2018-07-13).

[50] Statista GmbH and EHI Retail Institute GmbH. *Study 'Der deutsche E-Commerce-Markt 2014'*. http://www.ibusiness.de/shop/db/ib_shop.055782bma.html. (last downloaded 2018-07-13).

[51] Sun Microsystems, Inc. *JSR 30: J2ME®Connected, Limited Device Configuration*. https://jcp.org/en/jsr/detail?id=30. (last downloaded 2018-07-13).

[52] Sun Microsystems, Inc. *JSR 37: Mobile Information Device Profile for the J2ME®Platform*. https://jcp.org/en/jsr/detail?id=37. (last downloaded 2018-07-13).

[53] Sun Microsystems, Inc. *Java Persistence 2.1 Specification*. https://jcp.org/en/jsr/detail?id=338. (last downloaded 2018-07-13).

[54] Sybase Inc. / iAnywhere Solutions, Inc. *Adaptive Server ® Anywhere 9.0.2 – Adaptive Server Anywhere SQL User's Guide*. http://infocenter.sybase.com/archive/topic/com.sybase.help.adaptive_server_anywhere_9.0.2/pdf/asa902/dbugen9.pdf. (last downloaded 2018-07-13).

[55] Sybase Inc. / iAnywhere Solutions, Inc. *Adaptive Server ® Anywhere 9.0.2 – SQL Remote ™ User's Guide*. http://infocenter.sybase.com/archive/topic/com.sybase.help.adaptive_server_anywhere_9.0.2/pdf/asa902/dbsren9.pdf. (last downloaded 2018-07-13).

[56] Sybase Inc. / iAnywhere Solutions, Inc. *Adaptive Server ® Anywhere 9.0.2 – Introducing SQL Anywhere ® Studio*. http://infocenter.sybase.com/archive/topic/com.sybase.help.adaptive_server_anywhere_9.0.2/pdf/asa902/dbfgen9.pdf. (last downloaded 2018-07-13).

[57] Technische Hochschule Mittelhessen – Fachbereich 08 (MND) KITE / Philipps-Universität Marburg – Fachbereich 12 Softwaretechnik. *PIMAR Mathematikum*. https://play.google.com/store/apps/details?id=mathematikum.android. (last downloaded 2018-07-13).

[58] The Eclipse Foundation. *Eclipse ATL Transformation Language*. http://www.eclipse.org/atl. (last downloaded 2018-07-13).

[59] The Eclipse Foundation. *The connected data objects model repsitory project (CDO)*. http://www.eclipse.org/cdo. (last downloaded 2018-07-13).

[60] The Eclipse Foundation. *Eclipse Edapt*. http://www.eclipse.org/edapt. (last downloaded 2018-07-13).

[61] The Eclipse Foundation. *Eclipse EGL Development Tools (EDT)*. http://www.eclipse.org/edt. (last downloaded 2018-07-13).

[62] The Eclipse Foundation. *Eclipse EMF Compare*. http://www.eclipse.org/emf/compare. (last downloaded 2018-07-13).

[63] The Eclipse Foundation. *Eclipse EMF Refactor*. http://www.eclipse.org/emf-refactor. (last downloaded 2018-07-13).

[64] The Eclipse Foundation. *The EMFStore project*. http://www.eclipse.org/emfstore. (last downloaded 2018-07-13).

[65] The Eclipse Foundation. *Graphical Modeling Framework*. http://www.eclipse.org/gmf. (last downloaded 2018-07-13).

[66] The Eclipse Foundation. *Eclipse Henshin*. http://www.eclipse.org/henshin. (last downloaded 2018-07-13).

[67] The Eclipse Foundation. *Java Emitter Templates (JET)*. http://www.eclipse.org/articles/Article-JET/jet_tutorial1.html. (last downloaded 2018-07-13).

[68] The Eclipse Foundation. *Jubula Functional Testing Tool*. http://www.eclipse.org/jubula/. (last downloaded 2018-07-13).

[69] The Eclipse Foundation. *RCP Testing Tool (RCPTT)*. http://www.eclipse.org/rcptt/. (last downloaded 2018-07-13).

[70] The Eclipse Foundation. *Eclipse Sirius*. http://www.eclipse.org/sirius. (last downloaded 2018-07-13).

[71] The Eclipse Foundation. *Eclipse Teneo*. https://wiki.eclipse.org/Teneo. (last downloaded 2018-07-13).

[72] The Eclipse Foundation. *Eclipse Xtext*. http://www.eclipse.org/Xtext. (last downloaded 2018-07-13).

[73] Tutorial "PIMAR Workshop". https://srv55.svn-repos.de/dev1018/pimar/documentation/html/index.html?Tutorial.html. (last downloaded 2018-07-13).

[74] University of Illinois. *The Gaia Project*. http://gaia.cs.uiuc.edu/. (last downloaded 2018-07-13).

[75] W3C World Wide Web Consortium. *State Chart XML (SCXML): State Machine Notation for Control Abstraction*. https://www.w3.org/TR/2015/REC-scxml-20150901/. (last downloaded 2018-07-13).

[76] W3C World Wide Web Consortium. *Web Content Accessibility Guidelines 2.0*. http://www.w3.org/TR/2008/REC-WCAG20-20081211/. (last downloaded 2018-07-13).

[77] WebRatio. *WebRatio Mobile Platform*. http://www.webratio.com/site/content/en/mobile-app-development. (last downloaded 2018-07-13).

[78] Zalando SE. https://www.zalando.de. (last downloaded 2018-07-13).

[Abo+14] Saeid Abolfazli, Zohreh Sanaei, Abdullah Gani, Feng Xia, and Laurence T. Yang. "Rich Mobile Applications: Genesis, taxonomy, and open issues". In: *Journal Network and Computer Applications* 40 (2014), pp. 345–362.

[Abo+99] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. "Towards a Better Understanding of Context and Context-Awareness". In: *Handheld and Ubiquitous Computing, First International Symposium, HUC'99, Karlsruhe, Germany, September 27-29, 1999, Proceedings*. Ed. by Hans-Werner Gellersen. Vol. 1707. Lecture Notes in Computer Science. Springer, 1999, pp. 304–307.

[Abr+04] Pekka Abrahamsson, Antti Hanhineva, Hanna Hulkko, Tuomas Ihme, Juho Jäälinoja, Mikko Korkala, Juha Koskela, Pekka Kyllönen, and Outi Salo. "Mobile-D: an agile approach for mobile application development". In: *Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2004, October 24-28, 2004, Vancouver, BC, Canada*. Ed. by John M. Vlissides and Douglas C. Schmidt. ACM, 2004, pp. 174–175.

[Abr05] Pekka Abrahamsson. "Keynote: Mobile software development–the business opportunity of today". In: *Proceedings of the International Conference on Software Development, Reykjavik, Iceland*. Ed. by Benediktssson O., Abrahamsson P., Dalcer D., Hvannberg E. T., O'Connor R., and Thorbergsson H. University of Iceland Press, 2005, pp. 20–23.

[AC06] Michal Antkiewicz and Krzysztof Czarnecki. "Framework-Specific Modeling Languages with Round-Trip Engineering". In: *Model Driven Engineering Languages and Systems, 9th International Conference, MoDELS 2006, Genova, Italy, October 1-6, 2006, Proceedings*. Ed. by Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio. Vol. 4199. Lecture Notes in Computer Science. Springer, 2006, pp. 692–706.

[Ady+00] Atul Adya, Barbara Liskov, and Patrick E. O'Neil. "Generalized Isolation Level Definitions". In: *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA, February 28 - March 3, 2000*. Ed. by David B. Lomet and Gerhard Weikum. IEEE Computer Society, 2000, pp. 67–78.

[AI10] S.A. Ahson and M. Ilyas. *Mobile Web 2.0: Developing and Delivering Services to Mobile Devices*. Boca Raton (FL), New York City (NY) USA, London UK: CRC Press, 2010.

[Ail+02] Heikki Ailisto, Petteri Alahuhta, Ville Haataja, Vesa Kyllönen, and Mikko Lindholm. "Structuring context aware applications: Five-layer model and example case". In: *UbiComp 2002: Ubiquitous Computing: 4th International Conference, Göteborg, Sweden, September 29-October 1, 2002. Proceedings*. Ed. by Gaetano Borriello and Lars E Holmquist. Springer Science & Business Media, 2002, pp. 1–5.

[AK03] Colin Atkinson and Thomas Kuhne. "Model-driven development: a metamodeling foundation". In: *IEEE software* 20.5 (2003), pp. 36–41.

[Ale12] P. Alencar. *Handbook of Research on Mobile Software Engineering: Design, Implementation, and Emergent Applications: Design, Implementation, and Emergent Applications*. Hershey (PA) USA: IGI Global, 2012.

[Ali97] Daniel G. Aliaga. "Virtual Objects in the Real World". In: *Communications of the ACM* 40.3 (1997), pp. 49–54.

[All+10] S. Allen, V. Graupera, and L. Lundrigan. *Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution*. New York City (NY) USA: Apress, 2010.

[Alm+06] João Paulo A. Almeida, Maria-Eugenia Iacob, Henk Jonkers, and Dick A. C. Quartel. "Model-Driven Development of Context-Aware Services". In: *Distributed Applications and Interoperable Systems, 6th IFIP WG 6.1 International Conference, DAIS 2006, Bologna, Italy, June 14-16, 2006, Proceedings*. Ed. by Frank Eliassen and Alberto Montresor. Vol. 4025. Lecture Notes in Computer Science. Springer, 2006, pp. 213–227.

[Amb12] Scott Ambler. *Agile Database Techniques: Effective Strategies for the Agile Software Developer*. New York City (NY) USA: Wiley, 2012.

[Are+10] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. "Henshin: Advanced Concepts and Tools for

In-Place EMF Model Transformations". In: *Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*. Ed. by Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen. Vol. 6394. Lecture Notes in Computer Science. Springer, 2010, pp. 121–135.

[AT13] Thorsten Arendt and Gabriele Taentzer. "A tool environment for quality assurance based on the Eclipse Modeling Framework". In: *Automated Software Engineering* 20.2 (2013), pp. 141–184.

[Aye+07a] Dhouha Ayed, Didier Delanote, and Yolande Berbers. *MDD Approach and Evaluation of Development of Context-Aware Applications*. Tech. rep. Katholieke Universiteit Leuven Belgium – CW495 Dept. of Computer Science, 2007.

[Aye+07b] Dhouha Ayed, Didier Delanote, and Yolande Berbers. "MDD Approach for the Development of Context-Aware Applications". In: *Modeling and Using Context, 6th International and Interdisciplinary Conference, CONTEXT 2007, Roskilde, Denmark, August 20-24, 2007, Proceedings*. Ed. by Boicho N. Kokinov, Daniel C. Richardson, Thomas Roth-Berghofer, and Laure Vieu. Vol. 4635. Lecture Notes in Computer Science. Springer, 2007, pp. 15–28.

[B'f04] Reza B'far. *Mobile computing principles: designing and developing mobile applications with UML and XML*. Cambridge UK: Cambridge University Press, 2004.

[Bac+12] Philip Bacon, Reinhard Budde, Karlheinz Kautz, Karin Kuhlenkamp, and Heinz Züllighoven. *Prototyping: An Approach to Evolutionary System Development*. Berlin, Heidelberg GER: Springer, 2012.

[Bae11] Nick Baetens. *Comparing graphical DSL editors: AToM3, GMF, MetaEdit*. Tech. rep. University of Antwerp, 2011.

[Bal+04] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. "Model-Based Performance Prediction in Software Development: A Survey". In: *IEEE Transactions on Software Engineering* 30.5 (2004), pp. 295–310.

[Bal+07] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. "A survey on context-aware systems". In: *International Journal of Ad Hoc and Ubiquitous Computing* 2.4 (2007), pp. 263–277.

[Bar+99] R. Bardohl, M. Minas, G. Taentzer, and A. Schürr. "Application of Graph Transformation to Visual Languages". In: *Handbook of Graph Grammars and Computing by Graph Transformation*. Ed. by H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg. World Scientific Publishing Co., Inc., 1999, pp. 105–180.

[Bat05] Don S. Batory. "Feature Models, Grammars, and Propositional Formulas". In: *Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings*. Ed. by J. Henk Obbink and Klaus Pohl. Vol. 3714. Lecture Notes in Computer Science. Springer, 2005, pp. 7–20.

[Bau+06] Benoit Baudry, Trung Dinh-Trong, Jean-Marie Mottu, Devon Simmonds, Robert France, Sudipto Ghosh, Franck Fleurey, and Yves Le Traon. "Model transformation testing challenges". In: *Proceedings of the ECMDA Workshop on Integration of Model Driven Development and Model Driven Testing, Bilbao, Spain*. Ed. by Tom Ritter Hajo Eichler. Fraunhofer IRB Verlag, 2006.

[BC04] Gregory Biegel and Vinny Cahill. "A Framework for Developing Mobile, Context-aware Applications". In: *Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom 2004), 14-17 March 2004, Orlando, FL, USA*. IEEE Computer Society, 2004, pp. 361–365.

[Beh10] Heiko Behrens. "MDSD for the iPhone: developing a domain-specific language and IDE tooling to produce real world applications for mobile devices". In: *Companion to the 25th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, SPLASH/OOPSLA 2010, October 17-21, 2010, Reno/Tahoe, Nevada, USA*. Ed. by William R. Cook, Siobhán Clarke, and Martin C. Rinard. ACM, 2010, pp. 123–128.

[Ben+11] Amel Bennaceur, Robert B. France, Giordano Tamburrelli, Thomas Vogel, Pieter J. Mosterman, Walter Cazzola, Fábio M. Costa, Alfonso Pierantonio, Matthias Tichy, Mehmet Aksit, Pär Emmanuelson, Gang Huang, Nikolaos Georgantas, and David Redlich. "Mechanisms for Leveraging Models at Runtime in Self-adaptive Software". In: *Models@run.time - Foundations, Applications, and Roadmaps [Dagstuhl Seminar 11481, November 27 - December 2, 2011]*. Ed. by Nelly Bencomo, Robert B. France, Betty H. C. Cheng, and Uwe Aßmann. Vol. 8378. Lecture Notes in Computer Science. Springer, 2011, pp. 19–46.

[Ber+04] Guy Bernard, Jalel Ben-Othman, Luc Bouganim, Gérôme Canals, Sophie Chabridon, Bruno Defude, Jean Ferrié, Stéphane Gançarski, Rachid Guerraoui, Pascal Molli, Philippe Pucheral, Claudia Roncancio, Patricia Serrano-Alvarado, and Patrick Valduriez. "Mobile Databases: a Selection of Open Issues and Research Directions". In: *ACM SIGMOD Record* 33.2 (2004), pp. 78–83.

[Ber+87] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Boston (MA) USA: Addison-Wesley, 1987.

[Ber+95] Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. "A Critique of ANSI SQL Isolation Levels". In: *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995.* Ed. by Michael J. Carey and Donovan A. Schneider. ACM Press, 1995, pp. 1–10.

[Bet+10] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. "A survey of context modelling and reasoning techniques". In: *Pervasive and Mobile Computing* 6.2 (2010), pp. 161–180.

[Bet13] Lorenzo Bettini. *Implementing Domain-Specific Languages with Xtext and Xtend*. Birmingham UK: Packt Publishing Ltd., 2013.

[BH16] Achim D. Brucker and Michael Herzberg. "On the Static Analysis of Hybrid Mobile Apps - A Report on the State of Apache Cordova Nation". In: *Engineering Secure Software and Systems - 8th International Symposium, ESSoS 2016, London, UK, April 6-8, 2016. Proceedings*. Ed. by Juan Caballero, Eric Bodden, and Elias Athanasopoulos. Vol. 9639. Lecture Notes in Computer Science. Springer, 2016, pp. 72–88.

[Bis11] R. H. Bishop. *Learning with LabVIEW*. London UK: Pearson Education, 2011.

[Boe79] Barry W. Boehm. "Guidelines for Verifying and Validating Software Requirements and Design Specifications". In: *Proceedings of the European Conference on Applied Information Technology of the International Federation for Information Processing, London, 25-28 September, 1979*. Ed. by P. A. Samet. North Holland, 1979, pp. 711–719.

[Boe84] Barry W. Boehm. "Verifying and Validating Software Requirements and Design Specifications". In: *IEEE Software* 1.1 (1984), pp. 75–88.

[Boo+05] Matthias Book, Volker Gruhn, Malte Hülder, and Clemens Schäfer. "A Methodology for Deriving the Architectural Implications of Different Degrees of Mobility in Information Systems". In: *New Trends in Software Methodologies, Tools and Techniques - Proceedings of the Fifth SoMeT 2005, September 28-30, 2005, Tokyo, Japan*. Ed. by Hamido Fujita and Mohamed

Mejri. Vol. 129. Frontiers in Artificial Intelligence and Applications. IOS Press, 2005, pp. 281–292.

[Boo03] Paul Boocock. "A Framework for Rapid Development of Model Compilers". In: *MDA Implementers' Workshop Succeeding with Model Driven Systems, May 12-15, 2003, Orlando, FL, USA*. Ed. by Object Management Group. Object Management Group, 2003.

[Bor+15] Kristopher Born, Thorsten Arendt, Florian Heß, and Gabriele Taentzer. "Analyzing Conflicts and Dependencies of Rule-Based Transformations in Henshin". In: *Fundamental Approaches to Software Engineering - 18th International Conference, FASE 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. Ed. by Alexander Egyed and Ina Schaefer. Vol. 9033. Lecture Notes in Computer Science. Springer, 2015, pp. 165–168.

[BP88] Barry W. Boehm and Philip N. Papaccio. "Understanding and Controlling Software Costs". In: *IEEE Transactions on Software Engineering* 14.10 (1988), pp. 1462–1477.

[Bre+14] Timothy Bresnahan, Joe Orsini, and Pai-Ling Yin. "Platform Choice by Mobile App Developers". In: *NBER Working Paper* (2014).

[Bro+06] Erwan Brottier, Franck Fleurey, Jim Steel, Benoit Baudry, and Yves Le Traon. "Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool". In: *17th International Symposium on Software Reliability Engineering (ISSRE 2006), 7-10 November 2006, Raleigh, North Carolina, USA*. IEEE Computer Society, 2006, pp. 85–94.

[BS13] Hamid Bagheri and Kevin J. Sullivan. "Bottom-up model-driven development". In: *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*. Ed. by David Notkin, Betty H. C. Cheng, and Klaus Pohl. IEEE Computer Society, 2013, pp. 1221–1224.

[Cal+02] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Nathalie Souchon, Laurent Bouillon, Murielle Florins, and Jean Vanderdonckt. "Plasticity of User Interfaces: A Revised Reference Framework". In: *Task Models and Diagrams for User Interface Design: Proceedings of the First International Workshop on Task Models and Diagrams for User Interface Design - TAMODIA 2002, 18-19 July 2002, Bucharest, Romania*. Ed. by Costin Pribeanu and Jean Vanderdonckt. INFOREC Publishing House Bucharest, 2002, pp. 127–134.

[Cal+03] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. "A Unifying Reference Framework for multi-target user interfaces". In: *Interacting with Computers* 15.3 (2003), pp. 289–308.

[Cer+00] Stefano Ceri, Piero Fraternali, and Aldo Bongio. "Web Modeling Language (WebML): a modeling language for designing Web sites". In: *Computer Networks* 33.1-6 (2000), pp. 137–157.

[Cer+02] Stefano Ceri, Piero Fraternali, and Maristella Matera. "Conceptual Modeling of Data-Intensive Web Applications". In: *IEEE Internet Computing* 6.4 (2002), pp. 20–30.

[Cer+03] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, and Maristella Matera. *Designing data-intensive Web applications*. Burlington (MA) USA: Morgan Kaufmann, 2003.

[Cer+07] Stefano Ceri, Florian Daniel, Maristella Matera, and Federico Michele Facca. "Model-driven development of context-aware Web applications". In: *ACM Transactions on Internet Technology (TOIT)* 7.1 (2007), p. 2.

[Cha+14] Salma Charkaoui, Zakaria Adraoui, and El Habib Benlahmar. "Cross-platform mobile development approaches". In: *Third IEEE International Colloquium in Information Science and Technology, CIST 2014, Tetouan, Morocco, October 20-22, 2014*. Ed. by Mohammed El Mohajir, Mohammed

Al Achhab, and Mohamed Chahhou. IEEE Computer Society, 2014, pp. 188–191.

[Che+03] Harry Chen, Tim Finin, and Anupam Joshi. "An ontology for context-aware pervasive computing environments". In: *Knowledge Engineering Review* 18.3 (2003), pp. 197–207.

[Che04] Harry Chen. "An intelligent broker architecture for pervasive context-aware systems". PhD thesis. University of Maryland, Baltimore County, 2004.

[Chr93] Panos K Chrysanthis. "Transaction Processing in Mobile Computing Environment". In: *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*. Ed. by Bharat Bhargava. IEEE Computer Society, 1993, pp. 77 –82.

[Cic+08] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. "Automating Co-evolution in Model-Driven Engineering". In: *12th International IEEE Enterprise Distributed Object Computing Conference, ECOC 2008, 15-19 September 2008, Munich, Germany*. IEEE Computer Society, 2008, pp. 222–231.

[CL11] Andre Charland and Brian LeRoux. "Mobile application development: web vs. native". In: *Communications of the ACM* 54.5 (2011), pp. 49–53.

[Cos+05a] Gennaro Costagliola, Vincenzo Deufemia, Giuseppe Polese, and Michele Risi. "Building syntax-aware editors for visual languages". In: *Journal of Visual Languages and Computing* 16.6 (2005), pp. 508 –540.

[Cos+05b] Gennaro Costagliola, Vincenzo Deufemia, and Giuseppe Polese. "Towards Syntax-Aware Editors for Visual Languages". In: *Electronic Notes in Theoretical Computer Science* 127.4 (2005), pp. 107 –125.

[Cri92] John Crinnion. *Evolutionary Systems Development: A Practical Guide to the Use of Prototyping Within a Structured Systems Methodology*. New York City (NY) USA: Perseus Publishing, 1992.

[Cun+11] Thiago Ferraz V. da Cunha, Valéria Lelli Leitão Dantas, and Rossana M. C. Andrade. "SLeSS: A Scrum and Lean Six Sigma Integration Approach for the Development of Sofware Customization for Mobile Phones". In: *25th Brazilian Symposium on Software Engineering, SBES 2011, Sao Paulo, Brazil, September 28-30, 2011*. IEEE Computer Society, 2011, pp. 283–292.

[DA00] Anind K Dey and Gregory D Abowd. "The context toolkit: Aiding the development of context-aware applications". In: *Workshop on Software Engineering for wearable and pervasive computing, June 6, 2000, Limerick, Ireland. Proceedings*. University of Washington, 2000, pp. 431–441.

[Dal+13] Isabelle Dalmasso, Soumya Kanti Datta, Christian Bonnet, and Navid Nikaein. "Survey, comparison and evaluation of cross platform mobile application development tools". In: *2013 9th International Wireless Communications and Mobile Computing Conference, IWCMC 2013, Sardinia, Italy, July 1-5, 2013*. Ed. by Roberto Saracco, Khaled Ben Letaief, Mario Gerla, Sergio Palazzo, and Luigi Atzori. IEEE, 2013, pp. 323–328.

[DB92] Paul Dourish and Victoria Bellotti. "Awareness and Coordination in Shared Workspaces". In: *CSCW '92, Proceedings of the Conference on Computer Supported Cooperative Work, Toronto, Canada, October 31 - November 4, 1992*. Ed. by Marilyn Mantel and Ronald Baecker. ACM, 1992, pp. 107–114.

[DD11] Josh Dehlinger and Jeremy Dixon. "Mobile application software engineering: Challenges and research directions". In: *Workshop on Mobile Software Engineering, 9th October 2011, Los Angeles, United States*. Ed. by Anthony I. Wassermann. mobileseworkshop.org, 2011.

[Deg+14] Sylvain Degrandsart, Serge Demeyer, Jan Van den Bergh, and Tom Mens. "A transformation-based approach to context-aware modelling". In: *Software & Systems Modeling* 13.1 (2014), pp. 191–208.

[Dey+01] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of

Context-Aware Applications”. In: *Human-Computer Interaction* 16.2-4 (2001), pp. 97–166.

[Dey01] Anind K. Dey. “Understanding and Using Context”. In: *Personal and Ubiquitous Computing* 5.1 (2001), pp. 4–7.

[DG00] Ravi A. Dirckze and Le Gruenwald. “A pre-serialization transaction management technique for mobile multidatabases”. In: *Mobile Networks and Applications* 5.4 (2000), pp. 311–321.

[DL+04] Juan De Lara, Hans Vangheluwe, and Manuel Alfonseca. “Meta-modelling and graph grammars for multi-paradigm modelling in AToM3”. In: *Software & Systems Modeling* 3.3 (2004), pp. 194–209.

[Dun+97] Margaret H. Dunham, Abdelsalam Helal, and Santosh Balakrishnan. “A Mobile Transaction Model That Captures Both the Data and Movement Behavior”. In: *Mobile Networks and Applications* 2.2 (1997), pp. 149–162.

[Ehr+05] Karsten Ehrig, Claudia Ermel, Stefan Hänsgen, and Gabriele Taentzer. “Generation of visual editors as eclipse plug-ins”. In: *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), November 7-11, 2005, Long Beach, CA, USA*. Ed. by David F. Redmiles, Thomas Ellman, and Andrea Zisman. ACM, 2005, pp. 134–143.

[Ehr+06] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Berlin, Heidelberg GER: Springer, 2006.

[Eis+01] Jacob Eisenstein, Jean Vanderdonckt, and Angel R. Puerta. “Applying model-based techniques to the development of UIs for mobile computers”. In: *Proceedings of the 6th International Conference on Intelligent User Interfaces, IUI 2001, Santa Fe, NM, USA, January 14-17, 2001*. Ed. by Candy L. Sidner and Johanna D. Moore. ACM, 2001, pp. 69–76.

[EK+15] Wafaa S El-Kassas, Bassem A Abdullah, Ahmed H Yousef, and Ayman M Wahba. “Taxonomy of Cross-Platform Mobile Applications Development Approaches”. In: *Ain Shams Engineering Journal* 8.2 (2015), pp. 163–190.

[Elo+14] Nadia Elouali, Xavier Le Pallec, José Rouillard, and Jean-Claude Tarby. “A Model-based Approach for Engineering Multimodal Mobile Interactions”. In: *Proceedings of the 12th International Conference on Advances in Mobile Computing and Multimedia, Kaohsiung, Taiwan, December 8-10, 2014*. Ed. by Shyue-Liang Leon Wang, Yu-Hui Tao, Liming Chen, and Chung-Nan Lee. ACM, 2014, pp. 52–61.

[Emm+13] Christos Emmanouilidis, Remous-Aris Koutsiamanis, and Aimilia Tasidou. “Mobile guides: Taxonomy of architectures, context awareness, technologies and applications”. In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 103–125.

[Eri13] Ericsson. *Ericsson mobility report: On the pulse of the Networked Society*. Tech. rep. Ericsson, Sweden, Tech. Rep. EAB-13:028853, 2013.

[Eri16] Ericsson. *Ericsson mobility report: On the pulse of the Networked Society*. Tech. rep. Ericsson, Sweden, Tech. Rep. EAB-16:018498, 2016.

[Esc+14] Javier R Escolar, Cristina G Cachón, Ignacio Marín, Jean Vanderdonckt, and Vivian Motti. “A Model-Based Approach to Generate Connection-aware Applications for the Mobile Web”. In: *Romanian Journal of Human-Computer Interaction* 7.2 (2014), p. 117.

[Fav04] Jean-Marie Favre. “Towards a basic theory to model model driven engineering”. In: *UML Modeling Languages and Applications: UML 2004 Satellite Activities Lisbon, Portugal, October 11-15, 2004, Revised Selected Papers*. Ed. by N.J. Nunes, B. Selic, A.R. da Silva, and A.T. Alvarez. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 44–45.

[FC04] Patrick Fahy and Siobhan Clarke. "CASS–a middleware for mobile context-aware applications". In: *Workshop on context awareness, Co-located at MobiSys'04 The Second International Conference on Mobile Systems, Applications and Services 2004 Boston, MA, USA – June 06 - 09, 2004*. ACM SIGMOBILE, 2004.

[FK05] William B. Frakes and Kyo Kang. "Software Reuse Research: Status and Future". In: *IEEE Transactions on Software Engineering* 31.7 (2005), pp. 529–536.

[Flo+06] Jacqueline Floch, Svein O. Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund, and Eli Gjørven. "Using Architecture Models for Runtime Adaptability". In: *IEEE Software* 23.2 (2006), pp. 62–70.

[Flo+14a] Harleen K Flora, Swati V Chande, and Xiaofeng Wang. "Adopting an Agile Approach for the Development of Mobile Applications". In: *International Journal of Computer Applications* 94.17 (2014), pp. 43–50.

[Flo+14b] Harleen K Flora, Xiaofeng Wang, and Swati V Chande. "An Investigation into Mobile Application Development Processes: Challenges and Best Practices". In: *International Journal of Modern Education and Computer Science* 6.6 (2014), pp. 1–9.

[Fon07] Frédéric Fondement. "Concrete syntax definition for modeling languages". PhD thesis. École Polytechnique Fédérale de Lausanne, 2007.

[Fra+14] Mirco Franzago, Henry Muccini, and Ivano Malavolta. "Towards a Collaborative Framework for the Design and Development of Data-intensive Mobile Applications". In: *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*. MOBILESoft 2014. ACM, 2014, pp. 58–61.

[Fra+15] Rita Francese, Michele Risi, Giuseppe Scanniello, and Genoveffa Tortora. "A Qualitative Empirical Study in the Development of Multi-platform Mobile Applications". In: *Product-Focused Software Process Improvement - 16th International Conference, PROFES 2015, Bolzano, Italy, December 2-4, 2015, Proceedings*. Ed. by Pekka Abrahamsson, Luis Corral, Markku Oivo, and Barbara Russo. Vol. 9459. Lecture Notes in Computer Science. Springer, 2015, pp. 471–478.

[Fuc09] Thomas Fuchß. *Mobile Computing: Grundlagen und Konzepte für mobile Anwendungen*. (in german). Munich GER: Hanser, 2009.

[FV04] Murielle Florins and Jean Vanderdonckt. "Graceful degradation of user interfaces as a design method for multiplatform systems". In: *Proceedings of the 9th International Conference on Intelligent User Interfaces, IUI 2004, Funchal, Madeira, Portugal, January 13-16, 2004*. Ed. by Jean Vanderdonckt, Nuno Jardim Nunes, and Charles Rich. ACM, 2004, pp. 140–147.

[Gao+15] Lamia Gaouar, Abdelkrim Benamar, and Fethi Tarik Bendimerad. "Model Driven Approaches to Cross Platform Mobile Development". In: *Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication, Batna, Algeria – November 23 - 25, 2015*. Ed. by Djallel Eddine Boubiche, Faouzi Hidoussi, and Homero Toral Cruz. New York, NY, USA: ACM, 2015, 19:1–19:5.

[Gar+06] Angelo Gargantini, Elvinia Riccobene, and Patrizia Scandurra. "Deriving a textual notation from a metamodel: an experience on bridging Modelware and Grammarware". In: *European Workshop on Milestones, Models and Mappings for Model-Driven Architecture (3M4MDA), Bilbao, Spain, July 11, 2006. Proceedings*. Ed. by João Paulo Andrade Almeida, Luís Ferreira Pires, and Marten van Sinderen. CTIT Workshop Proceedings Series WP06-02. Centre for Telematic and Information Technology, 2006.

[Geo+07] Nektarios Georgalas, Shumao Ou, Manooch Azmoodeh, and Kun Yang. "Towards a Model-Driven Approach for Ontology-Based Context-Aware Application Development: A Case Study". In: *Model-based Methodologies for Pervasive and Embedded Software, 4th International Workshop on*

Model-based Methodologies for Pervasive and Embedded Software, MOMPES 2007, Braga, Portugal, March 31, 2007, Proceedings. Ed. by João M. Fernandes, Ricardo Jorge Machado, Ridha Khédri, and Siobhán Clarke. IEEE Computer Society, 2007, pp. 21–32.

[Ger+16] René Gerlach, Michael Guckert, Cornelius Malerczyk, Hans Christian Arlt, Steffen Vaupel, Gabriele Taentzer, and Michael Fatum. "Modellgetriebene Entwicklung mobiler Anwendungen mit Augmented Reality Funktionalität". In: *Mobile Anwendungen in Unternehmen*. Ed. by Thomas Barton, Christian Müller, and Christian Seel. Springer, 2016, pp. 193–211.

[GH04] Ping Guo and Reiko Heckel. "Modeling and Simulation of Context-Aware Mobile Systems". In: *19th IEEE International Conference on Automated Software Engineering (ASE 2004), 20-25 September 2004, Linz, Austria*. IEEE Computer Society, 2004, pp. 430–433.

[GM83] Hector Garcia-Molina. "Using Semantic Knowledge for Transaction Processing in a Distributed Database". In: *ACM Transactions on Database Systems (TODS)* 8.2 (1983), pp. 186–213.

[GM94] Saul Greenberg and David Marwood. "Real Time Groupware as a Distributed System: Concurrency Control and Its Effect on the Interface". In: *CSCW '94, Proceedings of the Conference on Computer Supported Cooperative Work, Chapel Hill, NC, USA, October 22-26, 1994*. Ed. by John B. Smith, F. Donelson Smith, and Thomas W. Malone. ACM, 1994, pp. 207–217.

[Gol03a] Christoph Gollmick. "Client-oriented replication in Mobile Database Environments". In: *International Conference on Management of Data and Symposium on Principles Database and Systems (co-located with FCRC 2003 Conference) San Diego, CA, USA – June 09 - 12, 2003*. Ed. by Zachary Ives, Yannis Papakonstantinou, and Alon Halevy. ACM, 2003.

[Gol03b] Christoph Gollmick. "Replication in Mobile Database Environments: A Client-Oriented Approach". In: *14th International Workshop on Database and Expert Systems Applications (DEXA'03), September 1-5, 2003, Prague, Czech Republic*. IEEE Computer Society, 2003, pp. 980–981.

[Gol06] Christoph Gollmick. "Konzept, Realisierung und Anwendung nutzerdefinierter Replikation in mobilen Datenbanksystemen". PhD thesis. University of Jena, Germany, 2006.

[GR11] Mark H. Goadrich and Michael P. Rogers. "Smart smartphone development: iOS versus android". In: *Proceedings of the 42nd ACM technical symposium on Computer science education, SIGCSE 2011, Dallas, TX, USA, March 9-12, 2011*. Ed. by Thomas J. Cortina, Ellen Lowenfeld Walker, Laurie A. Smith King, and David R. Musicant. ACM, 2011, pp. 607–612.

[Gra+96] Jim Gray, Pat Helland, Patrick E. O'Neil, and Dennis E. Shasha. "The Dangers of Replication and a Solution". In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996.* Ed. by H. V. Jagadish and Inderpal Singh Mumick. ACM Press, 1996, pp. 173–182.

[Gro+14] Tor-Morten Gronli, Jarle Hansen, Gheorghita Ghinea, and Muhammad Younas. "Mobile application platform heterogeneity: Android vs Windows Phone vs iOS vs Firefox OS". In: *Advanced Information Networking and Applications (AINA), 2014 IEEE 28th International Conference on*. IEEE, 2014, pp. 635–641.

[Gro09] R.C. Gronback. *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*. London UK: Pearson Education, 2009.

[Gru+07] Boris Gruschko, Dimitrios Kolovos, and Richard Paige. "Towards Synchronizing Models with Evolving Metamodels". In: *Proceeding Internatiol Workshop on Model-Driven Software Evolution at IEEE European Conference on Software Maintenance and Reengineering (ECSMR) March 21-23, 2007*. Ed. by Dalila Tamzalit. 2007.

[Grø+10] Tor-Morten Grønli, Jarle Hansen, and Gheorghita Ghinea. "Android vs Windows Mobile vs Java ME: a comparative study of mobile development environments". In: *Proceedings of the 3rd International Conference on Pervasive Technologies Related to Assistive Environments, PETRA 2010, Samos, Greece, June 23-25, 2010*. Ed. by Fillia Makedon. ACM International Conference Proceeding Series. ACM, 2010.

[Gu+04] Tao Gu, Hung Keng Pung, and Da Qing Zhang. "A middleware for building context-aware mobile services". In: *Vehicular Technology Conference, 2004. VTC 2004-Spring. 2004 IEEE 59th*. Vol. 5. IEEE, 2004, pp. 2656–2660.

[Guc+15] Michael Guckert, Cornelius Malerczyk, René Gerlach, Gabriele Taentzer, Steffen Vaupel, and Michael Fatum. "Plattformunabhängige Entwicklung mobiler Anwendungen mit Augmented Reality-Funktionalität". In: *Anwendungen und Konzepte in der Wirtschaftsinformatik – AKWI* 3 (2015), pp. 14–18.

[GW13] Olivier Le Goaer and Sacha Waltham. "Yet another DSL for cross-platforms mobile development". In: *Proceedings of the First Workshop on the Globalization of Domain Specific Languages, GlobalDSL@ECOOP 2013, Montpellier, France, July 1, 2013*. Ed. by Benoît Combemale, Walter Cazzola, and Robert B. France. ACM, 2013, pp. 28–33.

[Ham+16] Nicholas A.S. Hamm, van Lochem M., Hoek G., Otjes R., van der Sterren S., and Verhoeven H. "The Invisible Made Visible: Science and Technology." In: *AiREAS: Sustainocracy for a Healthy City*. Ed. by Jean-Paul Close. Springer International Publishing, 2016.

[Hei+12] Henning Heitkötter, Sebastian Hanschke, and Tim A. Majchrzak. "Evaluating Cross-Platform Development Approaches for Mobile Applications". In: *Web Information Systems and Technologies - 8th International Conference, WEBIST 2012, Porto, Portugal, April 18-21, 2012, Revised Selected Papers*. Ed. by José Cordeiro and Karl-Heinz Krempels. Vol. 140. Lecture Notes in Business Information Processing. Springer, 2012, pp. 120–138.

[Hei+13] Henning Heitkötter, Tim A. Majchrzak, and Herbert Kuchen. "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*. Ed. by Sung Y. Shin and José Carlos Maldonado. ACM, 2013, pp. 526–533.

[Hei13] Henning Heitkötter. "Cross-platform development of mobile business apps: findings on existing approaches and design of a novel model driven solution". PhD thesis. University of Münster, 2013.

[Hel+96] Abdelsalam Helal, Santosh Balakrishnan, Margaret Dunham, and Ramez Elmasri. *A Survey of Mobile Transaction Models*. Tech. rep. 1259. Purdue University – Department of Computer Science Technical Reports, 1996.

[Hir+01] Roberto Hirsch, Andres Coratella, Miguel Felder, and Eduardo Rodriguez. "A Framework for Analyzing Mobile Transaction Models". In: *Journal of Database Management* 12.3 (2001), p. 36.

[Hof+03] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. "Context-Awareness on Mobile Devices - the Hydrogen Approach". In: *36th Hawaii International Conference on System Sciences (HICSS-36 2003), CD-ROM / Abstracts Proceedings, January 6-9, 2003, Big Island, HI, USA*. IEEE Computer Society, 2003, p. 292.

[Hon+09] Jongyi Hong, Euiho Suh, and Sung-Jin Kim. "Context-aware systems: A literature review and classification". In: *Expert Systems with Applications* 36.4 (2009), pp. 8509–8522.

[HR04] David Harel and Bernhard Rumpe. "Meaningful Modeling: What's the Semantics of "Semantics"?" In: *IEEE Computer* 37.10 (2004), pp. 64–72.

[Hud96] Paul Hudak. “Building Domain-Specific Embedded Languages”. In: *ACM Computing Surveys (CSUR)* 28.4es (1996), p. 196.

[HV11a] Zef Hemel and Eelco Visser. “Declaratively programming the mobile web with Mobl”. In: *Proceedings of the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2011, part of SPLASH 2011, Portland, OR, USA, October 22 - 27, 2011*. Ed. by Cristina Videira Lopes and Kathleen Fisher. ACM, 2011, pp. 695–712.

[HV11b] Zef Hemel and Eelco Visser. “Mobl: the new language of the mobile web”. In: *Companion to the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2011, part of SPLASH 2011, Portland, OR, USA, October 22 - 27, 2011*. Ed. by Cristina Videira Lopes and Kathleen Fisher. ACM, 2011, pp. 23–24.

[IS03] Jadwiga Indulska and Peter Sutton. “Location Management in Pervasive Systems”. In: *ACSW Frontiers 2003, 2003 ACSW Workshops - the Australasian Information Security Workshop (AISW) and the Workshop on Wearable, Invisible, Context-Aware, Ambient, Pervasive and Ubiquitous Computing (WICAPUC), Adelaide, South Australia, February 2003*. Ed. by Chris W. Johnson, Paul Montague, and Chris Steketee. Vol. 21. CRPIT. Australian Computer Society, 2003, pp. 143–151.

[Ita+05] Ziad Itani, Hassan Diab, and Hassan Artail. “Efficient pull based replication and synchronization for mobile databases”. In: *Proceedings of the International Conference on Pervasive Services 2005, ICPS ’05, Santorini, Greece, July 11-14, 2005*. IEEE Computer Society, 2005, pp. 401–404.

[Jao+16] Imen Jaouadi, Raoudha Ben Djemaa, and Hanêne Ben-Abdallah. “A model-driven development approach for context-aware systems”. In: *Software & Systems Modeling* (2016), pp. 1–27.

[Jeo+07] Nam Joo Jeon, Choon Seong Leem, Min Hyung Kim, and Hyoun Gyu Shin. “A taxonomy of ubiquitous computing applications”. In: *Wireless Personal Communications* 43.4 (2007), pp. 1229–1239.

[Jeo+08] Yang-Jae Jeong, Ji-Hyeon Lee, and Gyu-Sang Shin. “Development process of mobile application SW based on Agile methodology”. In: *Advanced Communication Technology, 2008. ICACT 2008. 10th International Conference on*. IEEE Computer Society, 2008, pp. 362–366.

[JJ14] Christopher Jones and Xiaoping Jia. “The AXIOM Model Framework - Transforming Requirements to Native Code for Cross-platform Mobile Applications”. In: *ENASE 2014 - Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering, Lisbon, Portugal, 28-30 April, 2014*. Ed. by Joaquim Filipe and Leszek A. Maciaszek. SciTePress, 2014, pp. 26–37.

[Joo+13] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. “Real Challenges in Mobile App Development”. In: *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, Maryland, USA, October 10-11, 2013*. IEEE Computer Society, 2013, pp. 15–24.

[Kan+90] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. *Feature-oriented domain analysis (FODA) feasibility study*. Tech. rep. Defense Technical Information Center, 1990.

[Kap+08] Georgia M. Kapitsaki, Dimitrios A. Kateros, and Iakovos S. Venieris. “Architecture for provision of context-aware web applications based on web services”. In: *Proceedings of the IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC 2008, 15-18 September 2008, Cannes, French Riviera, France*. IEEE, 2008, pp. 1–5.

[Kap+09] Georgia M. Kapitsaki, Dimitrios A. Kateros, George N. Prezerakos, and Iakovos S. Venieris. “Model-driven development of composite context-aware web applications”. In: *Information & Software Technology* 51.8 (2009), pp. 1244–1260.

[Kar+14] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. "Design Guidelines for Domain Specific Languages". In: *The Computing Research Repository (CoRR)* abs/1409.2378 (2014).

[KEG12] Heather Kennedy-Eden and Ulrike Gretzel. "A Taxonomy of Mobile Applications in Tourism". In: *e-Review of Tourism Research* 10.2 (2012), pp. 47–50.

[Keh+13] Timo Kehrer, Udo Kelter, and Gabriele Taentzer. "Consistency-preserving edit scripts in model versioning". In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*. Ed. by Ewen Denney, Tevfik Bultan, and Andreas Zeller. IEEE, 2013, pp. 191–201.

[KH10] Maximilian Koegel and Jonas Helming. "EMFStore: a model repository for EMF models". In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*. Ed. by Jeff Kramer, Judith Bishop, Premkumar T. Devanbu, and Sebastián Uchitel. ACM, 2010, pp. 307–308.

[Kle+03] Anneke .G. Kleppe, Jos .B. Warmer, and Wim Bast. *MDA Explained, the Model Driven Architecture: Practice and Promise*. Boston (MA) USA: Addison-Wesley, 2003.

[KM00] Oliver Köth and Mark Minas. "Generating diagram editors providing free-hand editing as well as syntax-directed editing". In: *Proc. International Workshop on Graph Transformation (GraTra), Mar 25 - Apr 2, 2000, Berlin, Germany*. 2000.

[Kol+10] Dimitrios S. Kolovos, Louis M. Rose, Saad bin Abid, Richard F. Paige, Fiona A. C. Polack, and Goetz Botterweck. "Taming EMF and GMF Using Model Transformation". In: *Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*. Ed. by Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen. Vol. 6394. Lecture Notes in Computer Science. Springer, 2010, pp. 211–225.

[Kor+03] Panu Korpipää, Jani Mäntyjärvi, Juha Kela, Heikki Keränen, and Esko-Juhani Malm. "Managing Context Information in Mobile Devices". In: *IEEE Pervasive Computing* 2.3 (2003), pp. 42–51.

[Kra11] Frank Alexander Kraemer. "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*. Ed. by Jon Whittle, Tony Clark, and Thomas Kühne. Vol. 6981. Lecture Notes in Computer Science. Springer, 2011, pp. 183–197.

[Kri+17] Lars Kristensen, Gabriele Taentzer, and Steffen Vaupel. "Towards Verification of Connection-Aware Transactions Models for Mobile Applications". In: *Petri Nets and Software Engineering. International Workshop, PNSE'17, Zaragoza, Spain, June 25-26, 2017. Proceedings*. Ed. by Daniel Moldt, Lawrence Cabac, and Heiko Rölke. Vol. 1846. CEUR Workshop Proceedings. CEUR-WS.org, 2017, pp. 227–228.

[KS13] M. Keith and M. Schincariol. *Pro JPA 2*. New York City (NY) USA: Apress, 2013.

[KT08] S. Kelly and J.P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. New York City (NY) USA: Wiley, 2008.

[Kul+11] Vinay Kulkarni, Souvik Barat, and Uday Ramteerthkar. "Early Experience with Agile Methodology in a Model-Driven Approach". In: *Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings*. Ed. by Jon Whittle, Tony Clark, and Thomas Kühne. Vol. 6981. Lecture Notes in Computer Science. Springer, 2011, pp. 578–590.

[Küh05] Thomas Kühne. "What is a Model?" In: *Language Engineering for Model-Driven Software Development*. Ed. by Jean Bezivin and Reiko Heckel.

Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[Küs+13] Martin Küster, Benjamin Klatt, Eike Kohnert, Steffen Brandt, and Johannes Tysiak. "Apps aus Kästchen und Linien - Modellgetriebene Multi-Plattformentwicklung mobiler Anwendungen". In: *OBJEKTspektrum* 1 (2013). (in german), pp. 70 –75.

[Kön09] Birgitta König-Ries. "Challenges in Mobile Application Development". In: *it - Information Technology* 51.2 (2009), pp. 69–71.

[Lee+02] Kwanwoo Lee, Kyo Chul Kang, and Jaejoon Lee. "Concepts and Guidelines of Feature Modeling for Product Line Software Engineering". In: *Software Reuse: Methods, Techniques, and Tools, 7th International Conference, ICSR-7, Austin, TX, USA, April 15-19, 2002, Proceedings*. Ed. by Cristina Gacek. Vol. 2319. Lecture Notes in Computer Science. Springer, 2002, pp. 62–77.

[Lem+13] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley R. Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ronald J. Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Antónia Lopes, Jeff Magee, Sam Malek, Serge Mankovski, Raffaela Mirandola, John Mylopoulos, Oscar Nierstrasz, Mauro Pezzè, Christian Prehofer, Wilhelm Schäfer, Richard D. Schlichting, Dennis B. Smith, João Pedro Sousa, Ladan Tahvildari, Kenny Wong, and Jochen Wuttke. "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap". In: *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*. Ed. by Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw. Vol. 7475. Lecture Notes in Computer Science. Springer, 2013, pp. 1–32.

[Lia14] S. Liao. *Migrating to Swift from Android*. New York City (NY) USA: Apress, 2014.

[Lie+78] Bennet P. Lientz, E. Burton Swanson, and Gerry Edward Tompkins. "Characteristics of Applications Software Maintenance". In: *Communications of the ACM* 21.6 (1978), pp. 466–471.

[Lie83] Bennet P. Lientz. "Issues in Software Maintenance". In: *ACM Computing Surveys* 15.3 (1983), pp. 271–278.

[Lim+04] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, and Víctor López-Jaquero. "USIXML: A Language Supporting Multi-path Development of User Interfaces". In: *Engineering Human Computer Interaction and Interactive Systems, Joint Working Conferences EHCI-DSVIS 2004, Hamburg, Germany, July 11-13, 2004, Revised Selected Papers*. Ed. by Rémi Bastide, Philippe A. Palanque, and Jörg Roth. Vol. 3425. Lecture Notes in Computer Science. Springer, 2004, pp. 200–220.

[Lin+14] Stefan Lindner, Philippe Büttner, G Taenzer, Steffen Vaupel, and Nele Russwinkel. "Towards an efficient evaluation of the usability of android apps by cognitive models". In: *Proceedings 3. Interdisziplinärer Workshop Kognitive Systeme: Mensch, Teams, Systeme und Automaten, Magdeburg, 25.-27. März 2014*. Ed. by A. Wendemuth, M. Jipp, A. Kluge, and D. Söffker. Otto von Guericke Universität Magdeburg, 2014.

[LL09] Fritz Laux and Tim Lessner. "Escrow Serializability and Reconciliation in Mobile Computing using Semantic Properties". In: *International Journal On Advances in Telecommunications* 2.2 (2009), pp. 72–87.

[LM10] Jeff Linwood and Dave Minter. *Beginning Hibernate*. New York City (NY) USA: Apress, 2010.

[Lon03] K. C. London. *An Evaluation of Compuware OptimalJ Professional Edition as an MDA Tool*. Tech. rep. University of York, 2003.

[LS80] Bennet P. Lientz and E. Burton Swanson. "Impact of development productivity aids on application system maintenance". In: *DATA BASE* 11.3 (1980), pp. 114–120.

[LS81] Bennet P. Lientz and E. Burton Swanson. "Problems in Application Software Maintenance". In: *Communications of the ACM* 24.11 (1981), pp. 763–769.

[LS94] Qi Lu and Mahadev Satyanarayanan. "Isolation-Only Transactions for Mobile Computing". In: *Operating Systems Review* 28.2 (1994), pp. 81–87.

[LS97] John Lee and Keith Simpson. "A High Performance Transaction Processing Algorithm for Mobile Computing". In: *Intelligent Information Systems, 1997. IIS '97, December 8-10, 1997 Grand Bahama Island, Bahamas. Proceedings*. Ed. by H. Adeli. IEEE Computer Society, 1997, pp. 486–491.

[Lud03] Jochen Ludewig. "Models in software engineering - an introduction". In: *Software & Systems Modeling* 2.1 (2003), pp. 5–14.

[Lup+09] Dennis Lupiana, Ciaran O'Driscoll, and Fredrick Mtenzi. "Characterising Ubiquitous Computing Environments". In: *International Journal of Web Applications* 1.4 (2009), pp. 253–262.

[Mad+02] Sanjay Kumar Madria, Mukesh K. Mohania, Sourav S. Bhowmick, and Bharat K. Bhargava. "Mobile data and transaction management". In: *Information Sciences* 141.3-4 (2002), pp. 279–309.

[MB01] Sanjay Kumar Madria and Bharat Bhargava. "A Transaction Model to Improve Data Availability in Mobile Computing". In: *Distributed and Parallel Databases* 10.2 (2001), pp. 127–160.

[Med+12] Z. Mednieks, L. Dornin, G.B. Meike, and M. Nakamura. *Programming Android*. Sebastopol (CA) USA: O'Reilly, 2012.

[Mei+10] G. Meiselwitz, B. Wentz, and J. Lazar. *Universal Usability: Past, Present, and Future*. Boston (MA) USA; Delft Netherlands: Now Publishers, 2010.

[Mel+03] Stephen J. Mellor, Anthony N. Clark, and Takao Futagami. "Guest Editors' Introduction: Model-Driven Development". In: *IEEE Software* 20.5 (2003), pp. 14–18.

[Mer+05] Marjan Mernik, Jan Heering, and Anthony M Sloane. "When and how to develop domain-specific languages". In: *ACM Computing Surveys (CSUR)* 37.4 (2005), pp. 316–344.

[Mer12] M. Mernik. *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments: Recent Developments*. Hershey (PA), New York City (NY) USA: Information Science Reference, 2012.

[MG06] Tom Mens and Pieter Van Gorp. "A Taxonomy of Model Transformation". In: *Electronic Notes in Theoretical Computer Science* 152 (2006), pp. 125–142.

[Min02] Mark Minas. "Specifying Graph-like Diagrams with DIAGEN". In: *Electronic Notes in Theoretical Computer Science* 72.2 (2002), pp. 102–111.

[MN82] Daniel A Menascé and Tatuo Nakanishi. "Optimistic versus pessimistic concurrency control mechanisms in database management systems". In: *Information systems* 7.1 (1982), pp. 13–27.

[Mod+06] Martin Modahl, Bikash Agarwalla, T Scott Saponas, Gregory Abowd, and Umakishore Ramachandran. "UbiqStack: a taxonomy for a ubiquitous computing software stack". In: *Personal and Ubiquitous Computing* 10.1 (2006), pp. 21–27.

[MS13] B. Mutschler and G. Specht. *Mobile Datenbanksysteme: Architektur, Implementierung, Konzepte*. (in german). Berlin, Heidelberg GER: Springer, 2013.

[Muc+12] Henry Muccini, Antonio Di Francesco, and Patrizio Esposito. "Software testing of mobile applications: Challenges and future research directions". In: *7th International Workshop on Automation of Software Test,* *AST 2012, Zurich, Switzerland, June 2-3, 2012*. Ed. by Daniel Hoffman, John Hughes, and Dianxiang Xu. IEEE Computer Society, 2012, pp. 29–35.

[Mul+08] Pierre-Alain Muller, Frédéric Fondement, Franck Fleurey, Michel Hassenforder, Rémi Schnekenburger, Sébastien Gérard, and Jean-Marc Jézéquel. "Model-driven analysis and synthesis of textual concrete syntax". In: *Software & Systems Modeling* 7.4 (2008), pp. 423–441.

[Muñ+06] Javier Muñoz, Vicente Pelechano, and Carlos Cetina. "Implementing a Pervasive Meeting Room: A Model Driven Approach." In: *IWUC*. 2006, pp. 13–20.

[Nav+09] David Navarre, Philippe A. Palanque, Jean-François Ladry, and Eric Barboni. "ICOs: A model-based user interface description technique dedicated to interactive systems addressing usability, reliability and scalability". In: *ACM Transactions on Computer-Human Interaction (TOCHI)* 16.4 (2009), pp. 1–56.

[Nei80] James Milne Neighbors. "Software construction using components". PhD thesis. University of California, Irvine, 1980.

[Neu+14] Patrick Neubauer, Tanja Mayerhofer, and Gerti Kappel. "Towards Integrating Modeling and Programming Languages: The Case of UML and Java". In: *Proceedings of the 2nd International Workshop on The Globalization of Modeling Languages co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, GEMOC@Models 2014, Valencia, - Spain, September 28, 2014.* Ed. by Benoît Combemale, Julien DeAntoni, and Robert B. France. Vol. 1236. CEUR Workshop Proceedings. CEUR-WS.org, 2014, pp. 23–32.

[Nic+07] Robert C. Nickerson, Upkar Varshney, Jan Muntermann, and Henri Isaac. "Towards a Taxonomy of Mobile Applications". In: *Reaching New Heights. 13th Americas Conference on Information Systems, AMCIS 2007, Keystone, Colorado, USA, August 9-12, 2007*. Ed. by John A. Hoxmeier and Stephen Hayne. Association for Information Systems, 2007, p. 338.

[Nov+10] Jernej Novak, Andrej Krajnc, and Rok Zontar. "Taxonomy of static code analysis tools". In: *MIPRO, 2010 Proceedings of the International Convention on Information and Communication Technology, Electronics and Microelectronics , Opatija, Croatia, 24-28 May, 2010*. IEEE Computer Society, 2010, pp. 418–422.

[O'N86] Patrick E. O'Neil. "The Escrow Transactional Method". In: *ACM Transactions on Database Systems (TODS)* 11.4 (1986), pp. 405–430.

[OB11] D. Oehlman and S. Blanc. *Pro Android Web Apps: Develop for Android using HTML5, CSS3 & JavaScript*. New York City (NY) USA: Apress, 2011.

[OB88] Thomas J. Ostrand and Marc J. Balcer. "The Category-Partition Method for Specifying and Generating Functional Tests". In: *Communications of the ACM* 31.6 (1988), pp. 676–686.

[Ou+06] Shumao Ou, Nektarios Georgalas, Manooch Azmoodeh, Kun Yang, and Xiantang Sun. "A Model Driven Integration Architecture for Ontology-Based Context Modelling and Context-Aware Application Development". In: *Model Driven Architecture - Foundations and Applications, Second European Conference, ECMDA-FA 2006, Bilbao, Spain, July 10-13, 2006, Proceedings*. Ed. by Arend Rensink and Jos Warmer. Vol. 4066. Lecture Notes in Computer Science. Springer, 2006, pp. 188–197.

[Ouz+09] Mourad Ouzzani, Brahim Medjahed, and Ahmed K. Elmagarmid. "Correctness Criteria Beyond Serializability". In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Özsu. Springer US, 2009, pp. 501–506.

[Pae+03] Frauke Paetsch, Armin Eberlein, and Frank Maurer. "Requirements Engineering and Agile Software Development". In: *12th IEEE International Workshops on Enabling Technologies (WETICE 2003), Infrastructure for*

Collaborative Enterprises, 9-11 June 2003, Linz, Austria. IEEE Computer Society, 2003, pp. 308–313.

[Pai+12] Richard F. Paige, Dimitrios S. Kolovos, and Fiona A. C. Polack. "Metamodelling for Grammarware Researchers". In: *Software Language Engineering, 5th International Conference, SLE 2012, Dresden, Germany, September 26-28, 2012, Revised Selected Papers*. Ed. by Krzysztof Czarnecki and Görel Hedin. Vol. 7745. Lecture Notes in Computer Science. Springer, 2012, pp. 64–82.

[Pal+12] Manuel Palmieri, Inderjeet Singh, and Antonio Cicchetti. "Comparison of cross-platform mobile development tools". In: *16th International Conference on Intelligence in Next Generation Networks, ICIN 2012, Berlin, Germany, October 8-11, 2012*. IEEE, 2012, pp. 179–186.

[Pan+11] Prasanta Kumar Panda, Sujata Swain, and PK Pattnaik. "Review of Some Transaction Models used in Mobile Databases". In: *International Journal of Instrumentation, Control & Automation (IJICA)* 1.1 (2011), pp. 99–104.

[Pan04] Ray Pandya. *Mobile and Personal Communication Systems and Services*. New York City (NY) USA: Wiley, 2004.

[Par12] T. Parr. *The Definitive ANTLR 4 Reference*. Raleigh (NC) USA: Pragmatic Bookshelf, 2012.

[PB95] Evaggelia Pitoura and Bharat K. Bhargava. "Maintaining Consistency of Data in Mobile Distributed Environments". In: *Proceedings of the 15th International Conference on Distributed Computing Systems, Vancouver, British Columbia, Canada, May 30 - June 2, 1995*. IEEE Computer Society, 1995, pp. 404–413.

[PB99] Evaggelia Pitoura and Bharat Bhargava. "Data Consistency in Intermittently Connected Distributed Systems". In: *IEEE Transactions on Knowledge and Data Engineering* 11.6 (1999), pp. 896–915.

[PD90] Rubén Prieto-Díaz. "Domain analysis: An introduction". In: *ACM SIGSOFT Software Engineering Notes* 15.2 (1990), pp. 47–54.

[Pip10] I. Piper. *Learn Xcode Tools for Mac OS X and iPhone Development*. New York City (NY) USA: Apress, 2010.

[Pit96] Evaggelia Pitoura. "A Replication Schema to Support Weak Connectivity in Mobile Information Systems". In: *Database and Expert Systems Applications, 7th International Conference, DEXA '96, Zurich, Switzerland, September 9-13, 1996, Proceedings*. Ed. by Roland Wagner and Helmut Thoma. Vol. 1134. Lecture Notes in Computer Science. Springer, 1996, pp. 510–520.

[Pou16] Jacob Poushter. *Smartphone ownership and Internet usage continues to climb in emerging economies*. Tech. rep. Pew Research Center, 2016.

[PS12] Evaggelia Pitoura and George Samaras. *Data management for mobile computing*. Berlin, Heidelberg GER: Springer, 2012.

[Pud10] Arno Puder. "Cross-compiling Android applications to the iPhone". In: *Proceedings of the 8th International Conference on Principles and Practice of Programming in Java, PPPJ 2010, Vienna, Austria, September 15-17, 2010*. Ed. by Andreas Krall and Hanspeter Mössenböck. ACM, 2010, pp. 69–77.

[Pue+94] Angel R. Puerta, Henrik Eriksson, John H. Gennari, and Mark A. Musen. "Beyond Data Models for Automated User Interface Generation". In: *People and Computers IX, Proceedings of HCI '94, Glasgow, August 1994*. Ed. by Gilbert Cockton, Stephen W. Draper, and George R. S. Weir. Cambridge University Press, 1994, pp. 353–366.

[Puv+16] Abhinay Puvvala, Amitava Dutta, Rahul Roy, and Priya Seetharaman. "Mobile Application Developers' Platform Choice Model". In: *49th Hawaii International Conference on System Sciences, HICSS 2016, Koloa, HI, USA, January 5-8, 2016*. Ed. by Tung X. Bui and Ralph H. Sprague Jr. IEEE Computer Society, 2016, pp. 5721–5730.

[Ran+12] David Raneburger, Roman Popp, and Jean Vanderdonckt. "An automated layout approach for model-driven WIMP-UI generation". In: *ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS'12, Copenhagen, Denmark - June 25 - 28, 2012*. Ed. by Simone Diniz Junqueira Barbosa, José Creissac Campos, Rick Kazman, Philippe A. Palanque, Michael D. Harrison, and Steve Reeves. ACM, 2012, pp. 91–100.

[Rod+98] Tom Rodden, Keith Cheverst, K Davies, and Alan Dix. "Exploiting context in HCI design for mobile systems". In: *Workshop on human computer interaction with mobile devices, Glasgow, Scotland, 21-23rd May 1998*. Ed. by Chris Johnson. Department of Computing Science, University of Glasgow, Scotland – GIST Technical Report G98-1., 1998, pp. 21–22.

[Rom+00] Gruia-Catalin Roman, Gian Pietro Picco, and Amy L. Murphy. "Software engineering for mobility: a roadmap". In: *22nd International Conference on on Software Engineering, Future of Software Engineering Track, ICSE 2000, Limerick Ireland, June 4-11, 2000.* Ed. by Anthony Finkelstein. ACM, 2000, pp. 241–258.

[Rom+02] Manuel Román, Christopher K. Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. "A Middleware Infrastructure for Active Spaces". In: *IEEE Pervasive Computing* 1.4 (2002), pp. 74–83.

[Ros+10] Louis M. Rose, Dimitrios S. Kolovos, Richard F. Paige, and Fiona A. C. Polack. "Model Migration with Epsilon Flock". In: *Theory and Practice of Model Transformations, Third International Conference, ICMT 2010, Malaga, Spain, June 28-July 2, 2010. Proceedings*. Ed. by Laurence Tratt and Martin Gogolla. Vol. 6142. Lecture Notes in Computer Science. Springer, 2010, pp. 184–198.

[Roy+09] Chanchal Kumar Roy, James R. Cordy, and Rainer Koschke. "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach". In: *Science of Computer Programming* 74.7 (2009), pp. 470–495.

[Roy87] W. W. Royce. "Managing the Development of Large Software Systems: Concepts and Techniques". In: *Proceedings, 9th International Conference on Software Engineering, Monterey, California, USA, March 30 - April 2, 1987.* Ed. by William E. Riddle, Robert M. Balzer, and Kouichi Kishida. ACM Press, 1987, pp. 328–339.

[RR08] Vahid Rahimian and Raman Ramsin. "Designing an agile methodology for mobile software development: A hybrid method engineering approach". In: *Proceedings of the IEEE International Conference on Research Challenges in Information Science, RCIS 2008, Marrakech, Morocco, June 3-6, 2008*. Ed. by Oscar Pastor, André Flory, and Jean-Louis Cavarero. IEEE, 2008, pp. 337–342.

[RS12] André Ribeiro and Alberto Rodrigues da Silva. "Survey on Cross-Platforms and Languages for Mobile Apps". In: *8th International Conference on the Quality of Information and Communications Technology, QUATIC 2012, Lisbon, Portugal, 2-6 September 2012, Proceedings*. Ed. by João Pascoal Faria, Alberto Rodrigues da Silva, and Ricardo Jorge Machado. IEEE Computer Society, 2012, pp. 255–260.

[Rub+11] D. Rubel, J. Wren, and E. Clayberg. *The Eclipse Graphical Editing Framework (GEF)*. Boston (MA) USA: Addison-Wesley, 2011.

[Rus+10] Davide Di Ruscio, Ralf Lämmel, and Alfonso Pierantonio. "Automated Co-evolution of GMF Editor Models". In: *Software Language Engineering - Third International Conference, SLE 2010, Eindhoven, The Netherlands, October 12-13, 2010, Revised Selected Papers*. Ed. by Brian A. Malloy, Steffen Staab, and Mark van den Brand. Vol. 6563. Lecture Notes in Computer Science. Springer, 2010, pp. 143–162.

[Rát+10] István Ráth, András Ökrös, and Dániel Varró. "Synchronization of abstract and concrete syntax in domain-specific modeling languages -

By mapping models and live transformations". In: *Software & Systems Modeling* 9.4 (2010), pp. 453–471.

[SA+04] Patricia Serrano-Alvarado, Claudia Roncancio, and Michel Adiba. "A Survey of Mobile Transactions". In: *Distributed and Parallel Databases* 16.2 (2004), pp. 193–230.

[SA14a] Luis Miguel Pires Teixeira da Silva and Fernando Brito e Abreu. "A MDE Generative Approach for Mobile Business Apps". In: *9th International Conference on the Quality of Information and Communications Technology, QUATIC 2014, Guimaraes, Portugal, September 23-26, 2014*. IEEE Computer Society, 2014, pp. 312–317.

[SA14b] Luís Pires da Silva and Fernando Brito e Abreu. "Model-Driven GUI Generation and Navigation for Android BIS Apps". In: *MODELSWARD 2014 - Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development, Lisbon, Portugal, 7 - 9 January, 2014*. Ed. by Luís Ferreira Pires, Slimane Hammoudi, Joaquim Filipe, and Rui César das Neves. SciTePress, 2014, pp. 400–407.

[Sal+99] Daniel Salber, Anind K. Dey, and Gregory D. Abowd. "The Context Toolkit: Aiding the Development of Context-Enabled Applications". In: *Proceeding of the CHI '99 Conference on Human Factors in Computing Systems: The CHI is the Limit, Pittsburgh, PA, USA, May 15-20, 1999.* Ed. by Marian G. Williams and Mark W. Altom. ACM, 1999, pp. 434–441.

[Sar14] Robert G. Sargent. "Verifying and validating simulation models". In: *Proceedings of the 2014 Winter Simulation Conference, Savannah, GA, USA, December 7-10, 2014*. Ed. by Stephen J. Buckley and John A. Miller. IEEE/ACM, 2014, pp. 118–131.

[Sat96] Mahadev Satyanarayanan. "Fundamental Challenges in Mobile Computing". In: *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, Pennsylvania, USA, May 23-26, 1996*. Ed. by James E. Burns and Yoram Moses. ACM, 1996, pp. 1–7.

[Sch+94] Bill N. Schilit, Norman Adams, and Roy Want. "Context-Aware Computing Applications". In: *First Workshop on Mobile Computing Systems and Applications, WMCSA 1994, Santa Cruz, CA, USA, December 8-9, 1994*. IEEE Computer Society, 1994, pp. 85–90.

[Sch+99] Albrecht Schmidt, Kofi Asante Aidoo, Antti Takaluoma, Urpo Tuomela, Kristof Van Laerhoven, and Walter Van de Velde. "Advanced Interaction in Context". In: *Handheld and Ubiquitous Computing, First International Symposium, HUC'99, Karlsruhe, Germany, September 27-29, 1999, Proceedings*. Ed. by Hans-Werner Gellersen. Vol. 1707. Lecture Notes in Computer Science. Springer, 1999, pp. 89–101.

[Sei03] Ed Seidewitz. "What Models Mean". In: *IEEE Software* 20.5 (2003), pp. 26–32.

[Sel03] Bran Selic. "The Pragmatics of Model-Driven Development". In: *IEEE Software* 20.5 (2003), pp. 19–25.

[Ser+01] Patricia Serrano-Alvarado, Claudia Roncancio, and Michel E. Adiba. "Analyzing Mobile Transaction Supports for DBMS". In: *12th International Workshop on Database and Expert Systems Applications (DEXA 2001), 3-7 September 2001, Munich, Germany*. Ed. by A Min Tjoa and Roland Wagner. IEEE Computer Society, 2001, pp. 595–600.

[Ser+10] Estefanía Serral, Pedro Valderas, and Vicente Pelechano. "Towards the Model Driven Development of context-aware pervasive systems". In: *Pervasive and Mobile Computing* 6.2 (2010), pp. 254–280.

[SF13] Andreas Schuler and Barbara Franz. "Rule-Based Generation of Mobile User Interfaces". In: *Tenth International Conference on Information Technology: New Generations, ITNG 2013, 15-17 April, 2013, Las Vegas, Nevada, USA*. Ed. by Shahram Latifi. IEEE Computer Society, 2013, pp. 267–272.

[SG09] Andrea Sindico and Vincenzo Grassi. "Model driven development of context aware software systems". In: *International Workshop on Context-Oriented Programming, COP 2009, Genova, Italy, July 7, 2009*. Ed. by Pascal Costanza, Richard P. Gabriel, Robert Hirschfeld, and Jorge Vallejos. ACM, 2009, 7:1–7:5.

[Sha01] Mary Shaw. "The Coming-of-Age of Software Architecture Research". In: *Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, 12-19 May 2001, Toronto, Ontario, Canada*. Ed. by Hausi A. Müller, Mary Jean Harrold, and Wilhelm Schäfer. IEEE Computer Society, 2001, pp. 656–664.

[Shi+12] Qiao Shi, Dong Wen, and Xiaoyan Yang. "Study of logical transaction based data synchronization scheme for pervasive computing". In: *World Automation Congress (WAC), 24-28 June 2012, Puerto Vallarta, Mexico*. IEEE, 2012.

[Sho+08] Marwa Shousha, Lionel Briand, and Yvan Labiche. "A UML/SPT model analysis methodology for concurrent systems based on genetic algorithms". In: *Model Driven Engineering Languages and Systems* (2008), pp. 475–489.

[Sil+14] Luís Pires da Silva, Fernando Brito e Abreu, and Vasco Amaral. "A Model-Driven Approach for Mobile Business Information Systems Applications". In: *Proceedings of Doctoral Symposium co-located with 17th International Conference on Model Driven Engineering Languages and Systems (2014), Valencia, Spain, September 30, 2014.* Ed. by Benoit Baudry. Vol. 1321. CEUR Workshop Proceedings. CEUR-WS.org, 2014.

[SK12] Archana Sharma and Vineet Kansal. "An Analysis of Mobile Transaction Methods and Limitations in Execution of M-Commerce Transaction". In: *International Journal of Computer Applications* 43.21 (2012), pp. 14–19.

[SLP04] Thomas Strang and Claudia Linnhoff-Popien. "A context modeling survey". In: *First International Workshop on Advanced Context Modelling, Reasoning And Management at UbiComp 2004, Nottingham, England, September 7, 2004*. 2004.

[Spi01] Diomidis Spinellis. "Notable design patterns for domain-specific languages". In: *Journal of Systems & Software* 56.1 (2001), pp. 91–99.

[Sta73] Herbert Stachowiak. *Allgemeine Modelltheorie*. (in german). Berlin, Heidelberg GER: Springer, 1973.

[Ste+09] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. Boston (MA) USA: Addison-Wesley, 2009.

[Str+08] Detlef Streitferdt, Georg Wendt, Philipp Nenninger, Alexander Nyßen, and Horst Lichter. "Model Driven Development Challenges in the Automation Domain". In: *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2008, 28 July - 1 August 2008, Turku, Finland*. IEEE Computer Society, 2008, pp. 1372–1375.

[Str+15] Eleni Stroulia, Blerina Bazelli, Joanna W. Ng, and Tinny Ng. "WL++: Code Generation of Multi-platform Mobile Clients to Restful Back-Ends". In: *2nd ACM International Conference on Mobile Software Engineering and Systems, MOBILESoft 2015, Florence, Italy, May 16-17, 2015*. Ed. by Aharon Abadi, Danny Dig, and Yael Dubinsky. IEEE Computer Society, 2015, pp. 136–137.

[Str98] Susanne Strahringer. "Ein sprachbasierter Metamodellbegriff und seine Verallgemeinerung durch das Konzept des Metaisierungsprinzips". In: *Modellierung '98, Proceedings des GI-Workshops in Münster, 11.-13. März 1998*. Ed. by Klaus Pohl, Andy Schürr, and Gottfried Vossen. Vol. 9. CEUR Workshop Proceedings. CEUR-WS.org, 1998.

[SV03] Nathalie Souchon and Jean Vanderdonckt. "A Review of XML-compliant User Interface Description Languages". In: *Interactive Systems. Design,* *Specification, and Verification, 10th International Workshop, DSV-IS 2003, Funchal, Madeira Island, Portugal, June 11-13, 2003, Revised Papers*. Ed. by Joaquim A. Jorge, Nuno Jardim Nunes, and João Falcão e Cunha. Vol. 2844. Lecture Notes in Computer Science. Springer, 2003, pp. 377–391.

[Tae+07] Gabriele Taentzer, André Crema, René Schmutzler, and Claudia Ermel. "Generating Domain-Specific Model Editors with Complex Editing Commands". In: *Applications of Graph Transformations with Industrial Relevance, Third International Symposium, AGTIVE 2007, Kassel, Germany, October 10-12, 2007, Revised Selected and Invited Papers*. Ed. by Andy Schürr, Manfred Nagl, and Albert Zündorf. Vol. 5088. Lecture Notes in Computer Science. Springer, 2007, pp. 98–103.

[Tal10] A.K. Talukdar. *Mobile Computing, 2E*. New York City (NY) USA: McGraw-Hill Inc., 2010.

[Tan08] D. Taniar. *Mobile computing: concepts, methodologies, tools, and applications*. Hershey (PA) USA: IGI Global, 2008.

[Tes+10] Ricardo Tesoriero, José A. Gallud, María Dolores Lozano, and Victor M. Ruiz Penichet. "CAUCE: Model-driven Development of Context-aware Applications for Ubiquitous Computing Environments". In: *Journal of Universal Computer Science* 16.15 (2010), pp. 2111–2138.

[TG95] Rajiv Tewari and Peter Grillo. "Data Management for Mobile Computing on the Internet". In: *Proceedings of the 1995 ACM 23rd Annual Conference on Computer Science, CSC '95, Nashville, TN, USA, February 28 - March 2, 1995*. Ed. by Richard S. Brice, C. Jinshong Hwang, and Betty W. Hwang. ACM, 1995, pp. 246–252.

[Thü+11] Thomas Thüm, Christian Kästner, Sebastian Erdweg, and Norbert Siegmund. "Abstract Features in Feature Modeling". In: *Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, August 22-26, 2011*. Ed. by Eduardo Santana de Almeida, Tomoji Kishi, Christa Schwanninger, Isabel John, and Klaus Schmid. IEEE Computer Society, 2011, pp. 191–200.

[TK05] Juha-Pekka Tolvanen and Steven Kelly. "Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences". In: *Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings*. Ed. by J. Henk Obbink and Klaus Pohl. Vol. 3714. Lecture Notes in Computer Science. Springer, 2005, pp. 198–209.

[TR03] Juha-Pekka Tolvanen and Matti Rossi. "MetaEdit+: defining and using domain-specific modeling languages and code generators". In: *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2003, October 26-30, 2003, Anaheim, CA, USA*. Ed. by Ron Crocker and Guy L. Steele Jr. ACM, 2003, pp. 92–93.

[Tra+12] Vi Tran, Jean Vanderdonckt, Ricardo Tesoriero, and François Beuvens. "Systematic generation of abstract user interfaces". In: *ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS'12, Copenhagen, Denmark - June 25 - 28, 2012*. Ed. by Simone Diniz Junqueira Barbosa, José Creissac Campos, Rick Kazman, Philippe A. Palanque, Michael D. Harrison, and Steve Reeves. ACM, 2012, pp. 101–110.

[Træ02] Hallvard Trætteberg. "Using User Interface Models in Design". In: *Computer-Aided Design of User Interfaces III, Proceedings of the Fourth International Conference on Computer-Aided Design of User Interfaces, May, 15-17, 2002, Valenciennes, France*. Ed. by Christophe Kolski and Jean Vanderdonckt. Kluwer, 2002, pp. 131–142.

[Tuo00] Antti-Pekka Tuovinen. "Practical Error Handling in Parsing Visual Languages". In: *Journal of Visual Languages & Computing* 11.5 (2000), pp. 505–528.

[TV16] Gabriele Taentzer and Steffen Vaupel. "Model-Driven Development of Mobile Applications: Towards Context-Aware Apps of High Quality". In: *Proceedings of the International Workshop on Petri Nets and Software Engineering 2016, including the International Workshop on Biological Processes & Petri Nets 2016 co-located with the 37th International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2016 and the 16th International Conference on Application of Concurrency to System Design ACSD 2016, Toruń, Poland, June 20-21, 2016.* Ed. by Lawrence Cabac, Lars Michael Kristensen, and Heiko Rölke. Vol. 1591. CEUR Workshop Proceedings. CEUR-WS.org, 2016, pp. 17–29.

[UB16] Eric Umuhoza and Marco Brambilla. "Model Driven Development Approaches for Mobile Applications: A Survey". In: *Mobile Web and Intelligent Information Systems - 13th International Conference, MobiWIS 2016, Vienna, Austria, August 22-24, 2016, Proceedings*. Ed. by Muhammad Younas, Irfan Awan, Natalia Kryvinska, Christine Strauss, and Do Van Thanh. Vol. 9847. Lecture Notes in Computer Science. Springer, 2016, pp. 93–107.

[Usm+14] Muhammad Usman, Muhammad Zohaib Z. Iqbal, and Muhammad Uzair Khan. "A Model-Driven Approach to Generate Mobile Applications for Multiple Platforms". In: *21st Asia-Pacific Software Engineering Conference, APSEC 2014, Jeju, South Korea, December 1-4, 2014. Volume 1: Research Papers*. Ed. by Sungdeok (Steve) Cha, Yann-Gaël Guéhéneuc, and Gihwon Kwon. IEEE, 2014, pp. 111–118.

[Van05] Jean Vanderdonckt. "A MDA-Compliant Environment for Developing User Interfaces of Information Systems". In: *Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings*. Ed. by Oscar Pastor and João Falcão e Cunha. Vol. 3520. Lecture Notes in Computer Science. Springer, 2005, pp. 16–31.

[Vau+14] Steffen Vaupel, Gabriele Taentzer, Jan Peer Harries, Raphael Stroh, René Gerlach, and Michael Guckert. "Model-Driven Development of Mobile Applications Allowing Role-Driven Variants". In: *Model-Driven Engineering Languages and Systems - 17th International Conference, MODELS 2014, Valencia, Spain, September 28 - October 3, 2014. Proceedings*. Ed. by Jürgen Dingel, Wolfram Schulte, Isidro Ramos, Silvia Abrahão, and Emilio Insfrán. Vol. 8767. Lecture Notes in Computer Science. Springer, 2014, pp. 1–17.

[Vau+15] Steffen Vaupel, Daniel Strüber, Felix Rieger, and Gabriele Taentzer. "Agile Bottom-Up Development of Domain-Specific IDEs for Model-Driven Development". In: *Proceedings of the Workshop on Flexible Model Driven Engineering co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2015), Ottawa, Canada, September 29, 2015.* Ed. by Davide Di Ruscio, Juan de Lara, and Alfonso Pierantonio. Vol. 1470. CEUR Workshop Proceedings. CEUR-WS.org, 2015.

[Vau+16a] Steffen Vaupel, Damian Wlochowitz, and Gabriele Taentzer. "A generic architecture supporting context-aware data and transaction management for mobile applications". In: *Proceedings of the International Conference on Mobile Software Engineering and Systems, MOBILESoft '16, Austin, Texas, USA, May 14-22, 2016.* ACM, 2016, pp. 111–122.

[Vau+16b] Steffen Vaupel, Gabriele Taentzer, René Gerlach, and Michael Guckert. "Model-driven development of platform-independent mobile applications supporting role-based app variability". In: *Software Engineering 2016, Fachtagung des GI-Fachbereichs Softwaretechnik, 23.-26. Februar 2016, Wien, Österreich*. Ed. by Jens Knoop and Uwe Zdun. Vol. 252. LNI. GI, 2016, pp. 99–100.

[Vau+18a] Steffen Vaupel, Gabriele Taentzer, and Michael Guckert. "Model-Driven Design of Connectivity-Aware Mobile Applications". In: *Mobile Apps*

Engineering: Architecture, Design, Development and Testing. Ed. by Ghita Kouadri Mostefaoui, Mitul Shukla, and Faisal Tariq. (To appear). CRC Press, 2018.

[Vau+18b] Steffen Vaupel, Gabriele Taentzer, René Gerlach, and Michael Guckert. "Model-driven development of mobile applications for Android and iOS supporting role-based app variability". In: *Software and System Modeling* 17.1 (2018), pp. 35–63.

[VB07] Dániel Varró and András Balogh. "The model transformation language of the VIATRA2 framework". In: *Science of Computer Programming* 68.3 (2007), pp. 214–234.

[VD+00] Arie Van Deursen, Paul Klint, and Joost Visser. "Domain-specific languages: An annotated bibliography". In: *ACM Sigplan Notices* 35.6 (2000), pp. 26–36.

[VG14] Thomas Vogel and Holger Giese. "On Unifying Development Models and Runtime Models". In: *Proceedings of the 9th Workshop on Models@run.time co-located with 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), Valencia, Spain, September 30, 2014.* Ed. by Sebastian Götz, Nelly Bencomo, and Robert B. France. Vol. 1270. CEUR Workshop Proceedings. CEUR-WS.org, 2014, pp. 5–10.

[VH08] Samyr Vale and Slimane Hammoudi. "Context-aware model driven development by parameterized transformation". In: *Proceedings of the First International Workshop on Model Driven Interoperability for Sustainable Information Systems (MDISIS'08), June 16, 2008, Montpellier, France*. Ed. by Jean-Pierre Bourey and Reyes Grangel Seguer. Vol. 340. CEUR Workshop Proceedings. CEUR-WS.org, 2008.

[Vli98] J. Vlissides. *Pattern hatching: design patterns applied*. Boston (MA) USA: Addison-Wesley, 1998.

[VV01] Upkar Varshney and Ronald J. Vetter. "A Framework for the Emerging Mobile Commerce Applications". In: *34th Annual Hawaii International Conference on System Sciences (HICSS-34), January 3-6, 2001, Maui, Hawaii, USA*. IEEE Computer Society, 2001.

[Völ+13] M. Völter, T. Stahl, J. Bettin, A. Haase, S. Helsen, and K. Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. New York City (NY) USA: Wiley, 2013.

[Völ09] Markus Völter. "MD* Best Practices". In: *Journal of Object Technology* 8.6 (2009), pp. 79–102.

[Was10] Anthony I. Wasserman. "Software engineering issues for mobile application development". In: *Proceedings of the Workshop on Future of Software Engineering Research, FoSER 2010, at the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, Santa Fe, NM, USA, November 7-11, 2010*. Ed. by Gruia-Catalin Roman and Kevin J. Sullivan. ACM, 2010, pp. 397–400.

[WC97] Gary D. Walborn and Panos K. Chrysanthis. "PRO-MOTION : Management of Mobile Transactions". In: *Proceedings of the 1997 ACM symposium on Applied Computing, SAC'97, San Jose, CA, USA, February 28 - March 1*. Ed. by Barrett R. Bryant, Janice H. Carroll, Dave Oppenheim, Jim Hightower, and K. M. George. ACM, 1997, pp. 101–108.

[WK05] Manuel Wimmer and Gerhard Kramler. "Bridging Grammarware and Modelware". In: *Satellite Events at the MoDELS 2005 Conference, MoDELS 2005 International Workshops, Doctoral Symposium, Educators Symposium, Montego Bay, Jamaica, October 2-7, 2005, Revised Selected Papers*. Ed. by Jean-Michel Bruel. Vol. 3844. Lecture Notes in Computer Science. Springer, 2005, pp. 159–168.

[Wol+14] D. Wolber, H. Abelson, E. Spertus, and L. Looney. *App Inventor 2: Create Your Own Android Apps*. Sebastopol (CA) USA: O'Reilly, 2014.

[WV02] G. Weikum and G. Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Burlington (MA) USA: Morgan Kaufmann, 2002.

[Yen+14] M. Yener, A. Theedom, and R. Rahman. *Professional Java EE Design Patterns*. New York City (NY) USA: Wiley, 2014.

[Zap13] B.C. Zapata. *Android Studio Application Development*. Birmingham UK: Packt Publishing, 2013.

[ZP11] Yuefeng Zhang and Shailesh Patel. "Agile Model-Driven Development in Practice". In: *IEEE Software* 28.2 (2011), pp. 84–91.

Index

PIMAR ... 5, 66, 108, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291 system context ... 9, 11, 12, 13, 14, 17, 145, 148, 169, 170, 171, 172, 199, 230, 232

Model-driven development (MDD) of software systems has been a serious trend in different application domains over the last 15 years. While technologies, platforms, and architectural paradigms have changed several times since model-driven development processes were first introduced, their applicability and usefulness are discussed every time a new technological trend appears. Looking at the rapid market penetration of smartphones, software engineers are curious about how model-driven development technologies can deal with this novel and emergent domain of software engineering (SE).

Indeed, software engineering of mobile applications provides many challenges that model-driven development can address. Model-driven development uses a platform independent model as a crucial artifact. Such a model usually follows a domain-specific modeling language and separates the business concerns from the technical concerns. These platform-independent models can be reused for generating native program code for several mobile software platforms. However, a major drawback of model-driven development is that infrastructure developers must provide a fairly sophisticated model-driven development infrastructure before mobile application developers can create mobile applications in a model-driven way.

Hence, the first part of this thesis deals with designing a model-driven development infrastructure for mobile applications. We will follow a rigorous design process comprising a domain analysis, the design of a domain-specific modeling language, and the development of the corresponding model editors. To ensure that the code generators produce high-quality application code and the resulting mobile applications follow a proper architectural design, we will analyze several representative reference applications beforehand. Thus, the reader will get an insight into both the features of mobile applications and the steps that are required to design and implement a model-driven development infrastructure.

As a result of the domain analysis and the analysis of the reference applications, we identified context-awareness as a further important feature of mobile applications. Current software engineering tools do not sufficiently support designing and implementing of context-aware mobile applications. Although these tools (e.g., middleware approaches) support the definition and the collection of contextual information, the adaptation of the mobile application must often be implemented by hand at a low abstraction level by the mobile application developers.

Thus, the second part of this thesis demonstrates how context-aware mobile applications can be designed more easily by using a model-driven development approach. Techniques such as model transformation and model interpretation are used to adapt mobile applications to different contexts at design time or runtime. Moreover, model analysis and model-based simulation help mobile application developers to evaluate a designed mobile application (i.e., app model) prior to its generation and deployment with respected to certain contexts.

We demonstrate the usefulness and applicability of the model-driven development infrastructure we developed by seven case examples. These showcases demonstrate the designing of mobile applications in different domains. We demonstrate the scalability of our model-driven development infrastructure with several performance tests, focusing on the generation time of mobile applications, as well as their runtime performance. Moreover, the usability was successfully evaluated during several hands-on training sessions by real mobile application developers with different skill levels.

Philipps-Universität Marburg, Deutschland
Fachbereich Mathematik und Informatik
2018